МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра інформаційно-обчислювальних систем і управління

**ВАСЕНКО Світозар Павлович**

# Програмний модуль адаптивної поведінки суперника в комп'ютерній грі з використанням штучного інтелекту/ Software module for adaptive opponent behavior in a computer game using artificial intelligence

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Штучний інтелект

Кваліфікаційна робота

Виконав студент групи КНШІ-41
С.П. Васенко

Науковий керівник:
к.т.н., доцент П.Є. Биковий

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

**Тернопіль – 2025**

**Факультет комп'ютерних інформаційних технологій**
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Штучний інтелект

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2025 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
**Васенко Світозар Павлович**

1. Тема кваліфікаційної роботи: Програмний модуль адаптивної поведінки суперника в комп'ютерній грі з використанням штучного інтелекту / Software module for adaptive opponent behavior in a computer game using artificial intelligence керівник роботи Биковий П.Є. к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 р. № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
   - проаналізувати існуючі сучасні методи моделювання поведінки неігрових персонажів;
   - сформувати концепцію побудови програмного модуля, який дозволяє реалізувати адаптивну поведінку супротивника шляхом навчання нейронної мережі через алгоритм PPO;
   - спроєктувати архітектуру програмного рішення, що складається з взаємопов'язаних Blueprint-компонентів: Agent, Interactor, Trainer і Manager для забезпечення гнучкості, модульності, адаптації до різних сценаріїв поведінки, а також можливості масштабування системи;
   - реалізувати механізмів збору спостережень та нарахування винагород, а також інтеграцію PPO-моделі у рушій Unreal Engine через плагін Learning Agents;
   - провести експериментальні дослідження ефективності модуля.
5. Перелік графічного матеріалу в роботі:
   - структурна схема архітектури програмного модуля;
   - Blueprint-компоненти програмного модуля;
   - графіки навчання агентів програмного модуля.

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів кваліфікаційної роботи | Примітка |
|---|---|---|---|
| 1 | Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи | до 01.01. 2025 р. | |
| 2 | Написання 1 розділу кваліфікаційної роботи | до 01.02. 2025 р. | |
| 3 | Написання 2 розділу кваліфікаційної роботи | до 01.04.2025 р. | |
| 4 | Написання 3 розділу кваліфікаційної роботи | до 01.05. 2025 р. | |
| 5 | Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником | до 15.05.2025 р. | |
| 6 | Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів | до 25.05.2025 р. | |
| 7 | Перевірка кваліфікаційної роботи на оригінальність тексту | до 30.05.2025 р. | |
| 8 | Оформлення кваліфікаційної роботи та отримання допуску до захисту | до 10.06.2025 р. | |
| 9 | Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії | до 10.06. 2025 р. | |

Студент_____ Васенко С.П.
( підпис ) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Биковий П.Є.
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи: 68 с., 26 рис., 3 додатки, 38 джерела.

Метою роботи є розробка програмного модуля адаптивної поведінки супротивника в комп'ютерній грі з використанням методів машинного навчання, зокрема алгоритму навчання з підкріпленням PPO (Proximal Policy Optimization).

У роботі використано технології ігрового рушія Unreal Engine 5.4.4, плагін Learning Agents та Blueprint-візуальне програмування. Реалізовано повноцінний цикл самонавчання NPC: збір спостережень, виконання дій, нарахування винагород і оновлення політики через нейронну мережу. Розроблено дві команди агентів з різними можливостями керування, створено систему винагород, визначено простір спостережень та дій.

Проведено серію симуляцій та експериментів для порівняння ефективності навчання. Результати показали перевагу агентів із повним контролем (рух, поворот, стрільба) над тими, чия стрільба реалізована автоматично. Модуль демонструє стабільну динаміку навчання, гнучкість архітектури та придатність до масштабування.

Розроблена система має практичну цінність для створення інтелектуальних супротивників у відеоіграх, досліджень в галузі навчання з підкріпленням та застосування нейронних мереж у симуляційних середовищах.

Ключові слова: Unreal Engine, Learning Agents, PPO, NPC, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ШТУЧНИЙ ІНТЕЛЕКТ, ПОВЕДІНКА СУПРОТИВНИКА, НЕЙРОННА МЕРЕЖА, Blueprint.

# ANNOTATION

Explanatory note to the qualification work: 68 p., 26 fig., 3 appendices, 38 sources.

The aim of the work is to develop a software module for adaptive enemy behavior in a computer game using machine learning methods, in particular the PPO (Proximal Policy Optimization) reinforcement learning algorithm.

The work uses the technologies of the Unreal Engine 5.4.4 game engine, the Learning Agents plugin and Blueprint-visual programming. A full cycle of NPC self-learning has been implemented: collecting observations, performing actions, accruing rewards and updating the policy via a neural network. Two teams of agents with different control capabilities have been developed, a reward system has been created, and the space of observations and actions has been defined.

A series of simulations and experiments have been conducted to compare the effectiveness of training. The results have shown the advantage of agents with full control (movement, rotation, shooting) over those whose shooting is implemented automatically. The module demonstrates stable learning dynamics, architectural flexibility, and scalability.

The developed system has practical value for creating intelligent opponents in video games, research in the field of reinforcement learning, and the application of neural networks in simulation environments.

Keywords: Unreal Engine, Learning Agents, PPO, NPC, REINFORCEMENT LEARNING, ARTIFICIAL INTELLIGENCE, OPPONENT BEHAVIOR, NEURAL NETWORK, Blueprint.

ЗМІСТ

# ВСТУП

Сфера розробки комп’ютерних ігор стрімко розвивається, інтегруючи сучасні технології штучного інтелекту для створення реалістичних, динамічних та захопливих ігрових світів. Одним з ключових елементів таких світів є неігрові персонажі (NPC — Non-Player Characters), поведінка яких значною мірою визначає рівень занурення, інтересу та виклику гравця [1, 2]. Традиційні скриптові підходи до реалізації логіки NPC є статичними та обмеженими, оскільки не забезпечують адаптації до стилю гри користувача. Тому зростає потреба у впровадженні адаптивних моделей поведінки, здатних до навчання та вдосконалення в процесі гри [3].

Актуальність теми обумовлена впровадженням нових підходів до проектування NPC з використанням навчання з підкріпленням, що дозволяє створювати більш реалістичну, варіативну та гнучку поведінку суперників. У даній роботі реалізація поведінки суперника здійснюється в ігровому рушії Unreal Engine 5.4.4 із використанням плагіну Learning Agents, що забезпечує можливість застосування сучасних алгоритмів навчання з підкріпленням [4]. В основі навчання лежить метод Proximal Policy Optimization (PPO) — один з найефективніших підходів до тренування агентів у складних середовищах [2, 5].

Метою роботи є розробка програмного модуля адаптивної поведінки суперника в комп’ютерній грі з використанням методу навчання з підкріпленням PPO у середовищі Unreal Engine.

Завдання дослідження:

1. Проаналізувати існуючі підходи до реалізації адаптивної поведінки NPC у комп’ютерних іграх.
2. Дослідити принципи роботи алгоритму PPO.
3. Розробити архітектуру модуля адаптивної поведінки з використанням плагіну Learning Agents.

4. Реалізувати навчання суперника у віртуальному середовищі.

5. Провести тестування ефективності адаптивної поведінки у порівнянні зі стандартними підходами.

Об’єкт дослідження — процес прийняття рішень NPC у динамічному ігровому середовищі.

Предмет дослідження — моделювання та навчання адаптивної поведінки суперника з використанням методу навчання з підкріпленням.

Методи дослідження: аналіз літературних джерел, алгоритмічне моделювання, машинне навчання (PPO), програмна реалізація у середовищі Unreal Engine, тестування.

Функціональність модуля полягає у забезпеченні інтелектуальної та адаптивної поведінки суперника залежно від дій гравця. Це підвищує динамічність гри, її реалістичність та загальний рівень геймерського досвіду.

Результати кваліфікаційної роботи можуть бути використані при розробці комерційних ігор, симуляторів, тренажерів, а також як основа для подальших досліджень у галузі поведінкових моделей на базі машинного навчання.

# 1 АНАЛІЗ ТЕХНОЛОГІЙ АДАПТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ В ІГРАХ

## 1.1 Роль штучного інтелекту у формуванні поведінки неігрових персонажів

Адаптивна поведінка суперника в комп'ютерних іграх є одним із ключових аспектів сучасного ігрового штучного інтелекту. Вона дозволяє NPC змінювати свою тактику залежно від дій гравця, що покращує загальний ігровий досвід та підвищує реіграбельність гри. Традиційні методи створення AI суперників часто передбачають статичні алгоритми або передбачувані скрипти, що можуть швидко втратити свою ефективність, особливо для досвідчених гравців. Впровадження адаптивного AI дозволяє створювати динамічніших та реалістичніших суперників, які можуть навчатися, аналізувати та реагувати на стиль гри користувача [6].

AI у комп'ютерних іграх може реалізовуватися через різні підходи, кожен з яких має свої переваги та обмеження [1]. Одним із найпростіших і найраніших підходів до створення поведінки штучного інтелекту в іграх є система, побудована на заздалегідь прописаних скриптах. Такий AI діє суворо за наперед визначеними правилами: якщо гравець знаходиться в певному місці або виконує певну дію, NPC запускає задану послідовність реакцій. Наприклад, у грі Doom (1993) вороги реагували на гравця доволі просто: помітили — атакують, втратили з поля зору — повертаються на початкову позицію. У Half-Life (1998) така логіка стала трохи складнішою: супротивники активували напад тільки після перетину невидимого тригера.

Подібна система дозволяє розробникам мати повний контроль над поведінкою супротивників, проте вона практично позбавлена гнучкості. З часом гравець починає помічати шаблони, і дії NPC стають передбачуваними, що негативно впливає на реіграбельність.

Згодом розробники почали використовувати так звані скінченні автомати станів (FSM). У таких системах NPC здатен змінювати свою поведінку залежно

від внутрішнього стану, в якому він перебуває. Наприклад, якщо ворог має низький рівень здоров'я, він переходить у режим втечі або оборони. У тому ж Half-Life це реалізується через стани патрулювання, атаки чи відступу — зокрема, солдати HECU можуть тікати, залишившись без підтримки. У The Legend of Zelda дії супротивників варіюються залежно від відстані до гравця. Така модель вже забезпечує базову адаптацію та є гнучкішою за скриптові рішення. Але вона має і свої обмеження — кількість станів зазвичай обмежена, а з ускладненням логіки проектування автомат стає важким для масштабування.

Ще одним поширеним підходом стали поведінкові дерева (Behavior Trees). Вони дозволяють створювати ієрархічну структуру прийняття рішень, де NPC реагує на зміни у середовищі відповідно до заданих умов. Наприклад, у Halo 2 супротивники оцінюють ситуацію та можуть обрати атаку, укриття або пошук підкріплення. У Alien: Isolation AI Чужого навіть запам'ятовує місця, де бачив гравця, і змінює тактику. Поведінкові дерева є дуже модульними й зручними для поетапної розробки. Проте з глибиною дерева зростає складність налаштувань, що вимагає більшої уважності при балансуванні логіки.

Значний стрибок в еволюції AI відбувся із приходом машинного навчання, зокрема методів на кшталт Q-learning та глибоких нейронних мереж. Такі системи не потребують жорстко прописаних сценаріїв — вони здатні навчатися, адаптуватися до поведінки гравця і змінювати свою тактику в реальному часі. У прикладі OpenAI Five (Dota 2) агент вчиться передбачати дії суперника, а AlphaStar від DeepMind демонструє вдосконалення тактик у Starcraft II шляхом безперервного самонавчання. Це дозволяє створювати абсолютно непередбачуваних супротивників, які кидають гравцю новий виклик щоразу. Але така гнучкість приходить із ціною — системи машинного навчання потребують потужних обчислювальних ресурсів і складної інфраструктури для тренування та контролю.

Паралельно з Q-learning, у відеоіграх почали активно використовувати нейронні мережі, які імітують роботу людського мозку. Вони здатні навчатися

на великих обсягах даних і прогнозувати дії гравця. Наприклад, у Forza Motorsport система Drivatar аналізує стиль водіння гравця та створює віртуального опонента, що імітує цю поведінку. У грі AI Dungeon мережа GPT-3 генерує сюжетні сценарії на льоту, реагуючи на введені тексти гравця. Такі системи створюють враження глибокої взаємодії та інтелектуального супротивника. Проте, як і у випадку з машинним навчанням, нейронні мережі вимагають багато ресурсів і не завжди передбачувані — вони можуть вивчити неочікувану або навіть шкідливу поведінку, якщо дані неякісні.

Ще одним підходом є генетичні алгоритми (GA) — методи, що базуються на принципах еволюції та природного відбору. У цьому випадку AI розвиває стратегії через мутації, відбір найефективніших рішень і поступову еволюцію. У грі Creatures (1996) віртуальні істоти здобувають унікальні навички у взаємодії з середовищем, а в Black & White (2001) тварини-боги навчаються відповідно до поведінки гравця. Крім того, метод NEAT (NeuroEvolution of Augmenting Topologies) дозволяє агентам еволюціонувати структуру нейромережі під час симуляцій перегонів, де кожна наступна генерація AI керує автомобілем ефективніше. Генетичні алгоритми дозволяють створити складну поведінку без необхідності вручну прописувати правила, але навчання зазвичай займає багато часу і не завжди передбачуване.

Загалом, адаптивна поведінка AI відіграє критично важливу роль у створенні глибокого ігрового досвіду. NPC, які реагують на дії гравця і змінюють свою поведінку залежно від ситуації, підвищують рівень реалізму та динамічності. Це також дозволяє тонко балансувати складність гри, роблячи її водночас цікавою для новачка та викликом для досвідченого гравця. Крім того, адаптивний AI значно покращує реіграбельність — оскільки поведінка противників щоразу нова, гравець отримує унікальний досвід з кожним проходженням.

## 1.2 Огляд і аналіз існуючих рішень

Сучасні комп’ютерні ігри використовують широкий спектр підходів до створення адаптивного штучного інтелекту (AI), який дозволяє неігровим персонажам (NPC) реагувати на дії гравця, підвищуючи реалістичність і динаміку геймплею. У цьому підрозділі розглядаються найвідоміші рішення, їхні технічні основи, переваги, недоліки та вплив на ігровий досвід. Аналіз включає як традиційні методи (скрипти, станові автомати, поведінкові дерева), так і передові технології (машинне навчання, генеративний AI), які демонструють еволюцію адаптивної поведінки суперників. Декілька популярних рішень наведені нижче [7].

### 1.2.1 Гра F.E.A.R

Гра F.E.A.R., випущена в 2005 році студією Monolith Productions, стала однією з перших у жанрі FPS, де штучний інтелект ворогів поводився так, ніби він дійсно аналізує навколишнє середовище й адаптується до дій гравця. Це стало можливим завдяки комбінації скінченних автоматів (Finite State Machines, FSM) та планувальника дій, орієнтованого на цілі (Goal-Oriented Action Planning, GOAP) [8].

FSM використовується для управління базовими станами ворогів, такими як патрулювання, атака, укриття або втеча. GOAP же дозволяє NPC не просто виконувати заздалегідь визначені скрипти, а динамічно будувати послідовність дій на основі цілей і доступних ресурсів. Наприклад, якщо ворог отримує наказ атакувати, він не просто біжить до гравця, а аналізує, як краще це зробити: атакувати напряму, обійти з флангу чи використати гранату, якщо гравець сховався за укриттям. Це створює враження "розумного" AI, оскільки NPC можуть перегруповуватися, закликати підкріплення та використовувати тактичні маневри.

Ключовий приклад цієї системи можна побачити в бою: якщо гравець заховається за ящиком і почне обстріл, вороги можуть вирішити його обійти або використати гранати, щоб змусити гравця вийти. Вони також координуються між собою — один може відволікати увагу, поки інший підкрадається збоку. Це забезпечує відчуття складного, але реалістичного бою, де вороги не просто "підставляються", а активно намагаються виграти.

### 1.2.2 Alien: Isolation — Чужий, що "мислить"

Гра Alien: Isolation, випущена в 2014 році студією Creative Assembly, встановила новий стандарт для AI у жанрі survival horror. Головний ворог — Чужий — не слідує заздалегідь визначеним маршрутами, а використовує поведінкові дерева (Behavior Trees) та додаткові алгоритми, що роблять його максимально непередбачуваним [9].

Система AI у цій грі складається з двох рівнів. "Директор AI" відстежує дії гравця та вирішує, коли і як Чужий повинен діяти. Наприклад, якщо гравець часто ховається у шафах або під столами, Чужий починає частіше перевіряти ці місця.

Другий рівень — безпосередньо AI Чужого, який визначає його поведінку залежно від почутого звуку, побаченого руху чи інших факторів. Спочатку ворог може легко піддатися обману, наприклад, на шумовий пристрій, але після кількох таких випадків він "вчиться" не реагувати на нього, якщо не бачить самого гравця. Завдяки цьому кожна зустріч із Чужим унікальна, а гравець ніколи не може повністю передбачити його поведінку.

### 1.2.3 Halo — Інтелект загонів

Серія Halo, починаючи з першої частини 2001 року, стала прикладом того, як можна ефективно реалізувати командний AI. У Halo NPC поводяться як справжні бойові загони, використовуючи скінченні стани (FSM) у поєднанні з логікою групової поведінки [10].

Кожен NPC має набір базових станів (наприклад, атака, укриття, відступ), а ворожі групи можуть координувати свої дії. Наприклад, якщо в сутичці гине лідер загону Ковенанту, інші можуть почати панікувати або відійти на безпечніші позиції. Це створює динамічні бої, де вороги не діють як окремі одиниці, а працюють разом, підтримуючи та змінюючи тактику.

У Halo 2 це стало ще помітнішим: грунти (молодші солдати Ковенанту) часто ховаються за щитами своїх союзників, а еліти (командири) можуть ухилятися від гранат або змінювати позицію, якщо їх атакують. Такі механіки змушують гравця продумувати тактику та не покладатися лише на грубу силу.

### 1.2.4 Гра The Last of Us Part II — Емоційний та адаптивний AI

Гра The Last of Us Part II (2020), створена студією Naughty Dog, підняла планку в розвитку поведінкового AI ще вище. Тут використовується поведінкове дерево (Behavior Trees) у поєднанні з динамічною адаптацією до стилю гри гравця [11].

Одна з ключових особливостей AI у грі — емоційність та індивідуальність NPC. Вороги не просто атакують гравця, а й реагують на події навколо. Наприклад, якщо гравець уб'є одного з супротивників, інші можуть вигукнути його ім’я, що робить сутички більш особистими. Якщо гравець використовує стелс, NPC починають активніше обшукувати територію, змінюючи маршрути патрулювання.

Вороги також діють як команда, координуючи свої дії. Якщо гравець ховається за укриттям, один NPC може піти на обхід, поки інший буде прикривати його. Завдяки цьому кожна битва відчувається живою та змушує гравця постійно адаптуватися.

### 1.2.5 Nemesis System — Вороги, які запам’ятовують гравця

Система Nemesis, створена Monolith Productions для Middle-earth: Shadow of Mordor (2014) та Shadow of War (2017), стала проривом у розробці AI [12].

На відміну від традиційних NPC, вороги тут не просто з’являються й зникають, а розвиваються у відповідь на дії гравця. Якщо гравець переможе одного з ватажків орків, але не доб'є його, той може повернутися сильнішим, отримавши нові здібності, або навіть запам’ятати попередні битви. Це робить кожного противника унікальним і створює динамічний світ, де гравець будує особисту історію з ворогами.

Ця система вплинула на багатьох розробників, хоча її використання обмежене патентом Warner Bros., який діятиме до 2036 року.

### 1.2.6 NVIDIA ACE

NVIDIA ACE (Avatar Cloud Engine) — це новий етап у розвитку штучного інтелекту для ігор. Використовуючи генеративний AI, ACE дозволяє NPC не тільки взаємодіяти з гравцем у реальному часі, але й адаптувати свою поведінку залежно від голосових команд та контексту гри.

Для прикладу, у грі Mecha BREAK NPC можуть отримувати інструкції від гравця і змінювати тактику на ходу [13]. Це відкриває нові можливості для створення автономних NPC, які спілкуються з гравцем не лише через заздалегідь прописані репліки, а й генерують відповіді на основі контексту ситуації.

В таблиці 1.1 представлено порівняльний аналіз ключових критеріїв розглянутих систем. Проведений аналіз дозволяє показати основні відмінності між цими системами.

Системи на кшталт Nemesis і Q-learning по-різному максимізують реіграбельність: перша через процедурність, друга через навчання. Системи NVIDIA ACE і Alien: Isolation фокусуються на зануренні, але потребують значних ресурсів. Системи F.E.A.R. і Halo пропонують баланс між простотою та ефективністю, але поступаються в адаптивності.

Таблиця 1.1 – Порівняльний аналіз сучасних комп’ютерних ігор

| Система | Адаптив-ність | Обчислюваль-ні ресурси | Складність реалізації | Вплив на реіграбельність |
|---|---|---|---|---|
| F.E.A.R. (FSM+GOAP) | Середня | Середня | Висока | Середній |
| Alien: Isolation (Behavior Trees + ML) | Висока | Високі | Висока | Високий |
| Halo (Squad AI + FSM) | Середня | Низькі | Середня | Середній |
| The Last of Us II **(**Behavior Trees + Dynamic AI) | Середня | Середні | Висока | Середній |
| NVIDIA ACE | Висока | Дуже високі | Дуже високий | Високий |
| Nemesis System | Висока | Низькі | Середня | Дуже високий |
| Q-learning | Дуже висока | Дуже високі | Дуже висока | Дуже високий |

## 1.3 Вибір алгоритму машинного навчання для адаптивної поведінки

У процесі моделювання поведінки віртуального супротивника в комп’ютерній грі ключову роль відіграє вибір ефективного методу навчання, який дозволяє агенту адаптувати свою стратегію до змін у поведінці гравця та умовах ігрового середовища. Одним із найбільш перспективних підходів у цьому контексті є навчання з підкріпленням (Reinforcement Learning, RL) — клас методів машинного навчання, у яких агент навчається шляхом взаємодії з середовищем, отримуючи винагороди за певні дії.

У рамках даної роботи розглядалися такі поширені алгоритми навчання з підкріпленням:

Q-learning — табличний метод, що добре працює для невеликих дискретних просторів станів і дій, але погано масштабується при збільшенні складності середовища [1, 17, 27, 30];

Deep Q-Network (DQN) — глибоке Q-навчання, що використовує нейронні мережі замість Q-таблиці, але має обмеження при роботі в середовищах з неперервними діями [5, 26, 28];

Actor-Critic методи, серед яких PPO (Proximal Policy Optimization) є одним із найбільш сучасних і стабільних [2, 16, 29].

З урахуванням специфіки проєкту було вирішено використати алгоритм Proximal Policy Optimization (PPO) — один із найефективніших методів у родині градієнтних політик. Цей підхід чудово поєднує в собі високу стабільність навчання, ефективне використання даних і водночас залишається доволі простим у реалізації. На відміну від багатьох інших методів, PPO забезпечує надійне оновлення політики завдяки механізму, що обмежує занадто різкі зміни параметрів. Такий механізм, відомий як кліпінг функції втрат, дозволяє уникнути "катастрофічних" оновлень і втрати накопиченого досвіду [2].

Однією з ключових причин вибору саме PPO стала його підтримка неперервного простору дій. Це особливо важливо в контексті розробки NPC, поведінка яких у грі повинна бути плавною, варіативною та наближеною до дій реального гравця. Ще одним визначальним фактором стала повна сумісність PPO з плагіном Learning Agents у середовищі Unreal Engine 5.4.4. Цей плагін надає розробнику всі необхідні інструменти для створення агентів, визначення спостережень, дій та винагород, а також для запуску й моніторингу процесу тренування [4].

PPO також зарекомендував себе як надзвичайно стабільний алгоритм, який демонструє хорошу узагальненість результатів навіть на тривалих епізодах тренування. Це дозволяє агенту не тільки швидко адаптуватися до поточних умов, а й зберігати навички протягом змін середовища [5].

Важливою перевагою цього підходу є і можливість інкрементального донавчання. Тобто, агент може оновлювати свою політику навіть під час симуляції або роботи в реальному часі, без необхідності повного перезапуску навчання. Крім того, PPO добре масштабується в умовах паралельних середовищ, що дає змогу тренувати агентів одночасно в кількох симуляціях. Це суттєво пришвидшує процес оптимізації та дозволяє ефективніше використовувати обчислювальні ресурси.

У підсумку, обраний підхід на основі PPO забезпечує гнучке, масштабоване та стійке рішення для реалізації адаптивної поведінки NPC. Такий алгоритм дозволяє навчати агентів без заздалегідь визначених сценаріїв — виключно на основі досвіду, який вони здобувають у процесі симульованої гри. Це відкриває можливість створення поведінки, що динамічно адаптується до дій гравця та складності середовища.

## 1.4 Постановка задачі

Розробка адаптивної поведінки віртуального супротивника у комп’ютерній грі починається з чіткого розуміння того, які завдання повинен виконувати інтелектуальний агент під час ігрового процесу. Особливо це актуально в контексті змагального середовища, де в грі беруть участь дві команди NPC і незалежний гравець. У такій ситуації супротивник повинен не просто виконувати базові дії на кшталт руху чи стрільби, а й динамічно реагувати на зміну умов — розташування ворогів, поведінку гравця, особливості геометрії карти тощо [1, 2].

У грі симулюється сценарій, у якому дві команди — червона та синя — мають по вісім агентів, а також є один незалежний гравець сірого кольору. Завдання кожної команди — набрати якомога більше очок шляхом знищення ворожих агентів, водночас мінімізуючи власні втрати. Усі агенти мають

можливість переміщуватися по карті, змінювати напрямок погляду та, залежно від налаштувань команди, стріляти самостійно або автоматично.

Це означає, що інтелектуальні агенти повинні володіти кількома ключовими навичками. Насамперед — орієнтуватися в частково спостережуваному середовищі, де не всі об’єкти доступні у кожен момент. Вони мають уміти приймати оптимальні рішення в реальному часі, швидко змінювати тактику залежно від ігрової ситуації, а також намагатися уникати негативних подій, таких як зіткнення зі стінами, наведення на союзника чи смерть. І навпаки — максимізувати корисну активність: прицілювання на ворогів чи гравця, ефективне переміщення, зайняття вигідних позицій тощо [1, 3].

Усе це дозволяє сформулювати завдання розробки як проблему навчання з підкріпленням. У такій постановці агент перебуває в певному стані, що формується на основі спостережень із навколишнього середовища — таких як координати та напрямки власні, ворожі та гравця, а також об’єкти на карті. У відповідь на цей стан агент виконує певну дію з доступного набору (рух, обертання камери, стрільба), отримує винагороду залежно від її наслідків, переходить у новий стан — і врешті-решт прагне максимізувати сумарну винагороду протягом епізоду [1, 2]. Для вирішення поставленого завдання необхідно виконати наступні задачі:

1. Реалізувати у червоній команді повнофункціональне керування: усі три аспекти поведінки — переміщення, поворот камери та стрільба — контролюються нейронною мережею, яка навчається за алгоритмом PPO (Proximal Policy Optimization) [2]. Система винагород у цій команді повинна бути налаштована так, щоб стимулювати обережну, але рішучу поведінку: +10 балів нараховується за прицілювання на ворожого агента, +20 — на гравця, тоді як -1 бал знімається за наведення на союзника або зіткнення зі стіною. Найбільш серйозне покарання передбачене за загибель агента — мінус 2.5 бала.

2. Реалізувати у синій команді агенти що мають обмежений контроль — вони мають відповідати лише за переміщення та орієнтацію в просторі, а постріл

виконується автоматично, коли відбувається наведення на ворожу ціль за допомогою трасування каналу (trace by channel) [4]. Система винагород тут також налаштована більш агресивно: +10 балів за наведення на червоного агента, +20 — на гравця, -5 балів за зіткнення зі стіною, -2.5 — за загибель.

# 2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

## 2.1 Архітектура ігрового AI-модуля у середовищі Unreal Engine

Розроблена ігрова система моделює багатокористувацький змагальний сценарій між двома командами NPC (червона і синя) та одним гравцем. Уся реалізація виконана виключно за допомогою Blueprints-системи в середовищі Unreal Engine 5.4.4. Blueprints — візуальний метод програмування, вбудований в рушій Unreal Engine [4, 34]. Blueprint дозволяє створювати ігрову логіку, обробляти події, керувати штучним інтелектом, взаємодією між об'єктами, навчанням агентів та анімаціями без написання коду. Це досягається завдяки візуальним блокам (вузлам), які з’єднуються в логічні графи, аналогічно до блок-схем, приклад вигляду Blueprint зображено на рисунку 2.1. Це дозволило досягти гнучкості в реалізації, швидкої інтеграції з плагіном Learning Agents і спрощення експериментування з логікою навчання [4, 6].

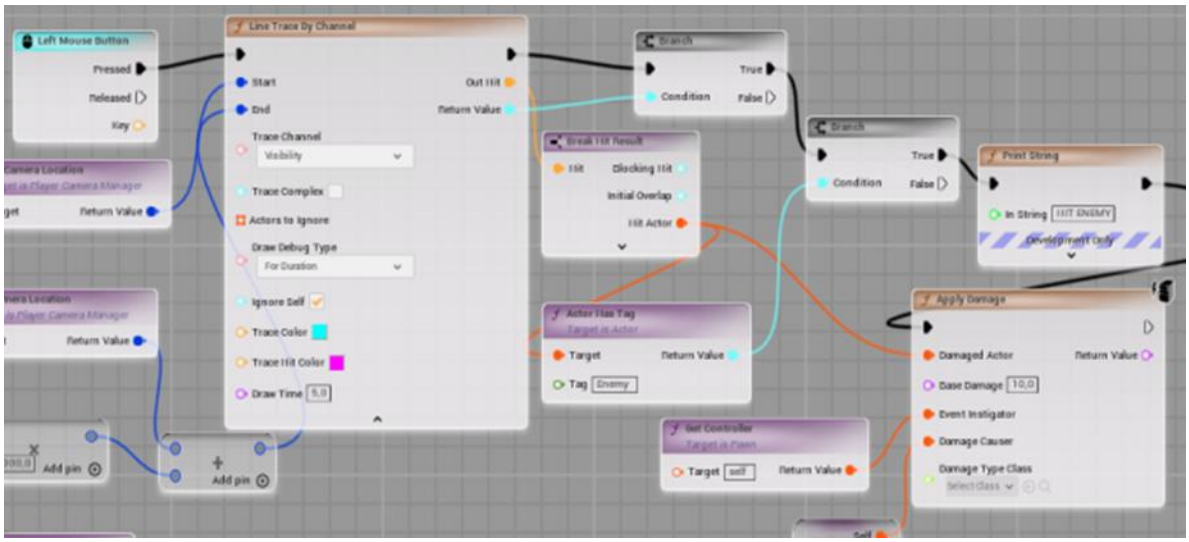



Рисунок 2.1 – Приклад вигляду Blueprint

Кожна з команд у проєкті має власну сукупність Blueprint-компонентів, які разом забезпечують повний цикл навчання з підкріпленням на основі алгоритму PPO [2]. Ці компоненти реалізують усі необхідні функції для створення

адаптивної поведінки NPC — від збору спостережень до виконання дій та обчислення винагород.

У першу чергу, основу кожного агента складає Agent Blueprint. Саме цей компонент визначає фізичну модель персонажа, його здатність рухатись, обертатись, стріляти, а також пов’язану з цим анімацію. Усе управління поведінкою NPC на рівні ігрового світу відбувається саме через цю структуру.

Наступним критичним елементом є Agent Interactor — компонент, який відповідає за спілкування агента з навчальною системою. Він формує спостереження (observations), що подаються на вхід нейронної мережі, та отримує у відповідь обрані дії. У межах цього блоку реалізовано ключові функції: Gather Agent Observation — для збору інформації про світ, і Specify Observation Schema — для опису структури цих даних.

Тренувальний процес конфігурується через Agent Trainer. Це Blueprint, що визначає спосіб навчання агента: обраний алгоритм (у нашому випадку PPO), систему винагород, гіперпараметри, простір дій і тип політики. Саме Trainer відповідає за оновлення політики на основі накопиченого досвіду та за контроль навчального циклу.

Ще один важливий компонент — Agent Manager. Він координує роботу всієї системи, включаючи створення та респаун NPC, моніторинг їх стану, управління командним рахунком і перезапуск раундів. Цей менеджер фактично пов’язує всі інші компоненти в єдину цілісну структуру.

Варто зазначити, що для кожної команди — червоної та синьої — описані компоненти реалізовано окремо, з урахуванням особливостей поведінки. У червоної команди контроль над агентами здійснюється повністю через нейронну мережу: вона приймає рішення про рух, поворот камери та стрільбу. У той же час у синьої команди мережа керує лише переміщенням та орієнтацією, а стрільба виконується автоматично, щойно ворог опиняється в полі зору — це реалізовано через механізм trace by channel [14].

У типовому циклі взаємодії агентів усе відбувається так. Спочатку Interactor збирає поточні спостереження про стан гри — це можуть бути координати агентів, напрямки їх руху, положення ворогів, гравця або об'єкти середовища (наприклад, стіни). Потім Learning Agent на основі цих даних визначає дію, використовуючи модель, натреновану з допомогою PPO. Далі агент виконує відповідну дію в грі: рухається, повертає камеру або стріляє. Після цього Trainer нараховує винагороду згідно з конфігурацією. Нарешті, Agent Manager фіксує зміну стану агента — зокрема, якщо NPC загинув — і, за потреби, ініціює новий раунд.

Уся ця система завдячує гнучкості Blueprint-реалізації, яка дала змогу швидко експериментувати зі змінами логіки, візуалізувати структуру спостережень і винагород, а також зберігати повну відокремленість між конфігураціями різних команд. Завдяки такій архітектурі не виникала потреба змінювати код рушія, що значно прискорило розробку [1, 5].

Створена модульна структура є масштабованою: за потреби можна легко змінювати кількість агентів, адаптувати гіперпараметри тренування, підключати нові алгоритми або експериментувати з альтернативною логікою поведінки на базі інших підходів до навчання з підкріпленням.

## 2.2 Алгоритм навчання з підкріпленням

Proximal Policy Optimization (PPO) — це алгоритм навчання з підкріпленням, який належить до сімейства методів на основі політики (policy-based methods). PPO був запропонований у 2017 році дослідниками з OpenAI як ефективний і стабільний спосіб тренування агентів у задачах, де потрібно оптимізувати поведінку через взаємодію з середовищем [2]. Алгоритм широко використовується в задачах ШІ, зокрема для тренування NPC у відеоіграх.

PPO базується на ідеї навчання з підкріпленням, де агент (наприклад, NPC) взаємодіє з середовищем, отримуючи винагороди за свої дії, і прагне

максимізувати сумарну винагороду [1]. PPO є вдосконаленням попередніх алгоритмів, таких як TRPO (Trust Region Policy Optimization), і має такі ключові особливості:

– PPO безпосередньо оптимізує функцію політики (policy), яка визначає ймовірність вибору певної дії в заданому стані;

– алгоритм обмежує оновлення політики, щоб уникнути занадто різких змін, які можуть дестабілізувати навчання4

– PPO поєднує простоту реалізації з високою ефективністю, що робить його популярним для задач із великими просторами дій і станів.

Основні компоненти PPO:

1. Політика (Policy)— це функція, яка визначає, яку дію обрати в заданому стані. PPO зазвичай використовує нейронну мережу для представлення політики, яка видає ймовірності дій [15].

2. Функція цінності (Value Function) – оцінює наскільки "вигідним" є певний стан. Вона допомагає агенту зрозуміти, чи приведе поточна дія до довгострокової винагороди. У PPO функція цінності також представлена нейронною мережею і тренується разом із політикою [1].

3. Середовище (Environment) - це ігровий світ, у якому NPC взаємодіє з іншими об’єктами.

4. Винагорода (Reward) - це числова оцінка дії NPC. PPO прагне максимізувати сумарну винагороду за епізод.

Псевдокод роботи алгоритму представлений нижче:

```
# 1. Ініціалізація
ініціалізувати параметри політики θ та функції цінності ϕ
встановити гіперпараметри:
ε (clipping), γ (discount),
λ (GAE), α (коеф. навчання),
N (розмір батчу), K (кількість епох)

while не досягнуто кінця навчання:
    trajectories = []

    # 2. Збір даних взаємодії агента зі середовищем
```

```
        for episode in range(N):
            зібрати (s_t, a_t, r_t, s_{t+1}, done) протягом T кроків
            зберегти до trajectories

        # 3. Обчислення переваг та цілей
        для кожного (s_t, a_t) в trajectories:
            обчислити Â_t = advantage estimate (наприклад, GAE)
            обчислити R_t = сумарна дисконтована винагорода

        # 4. Оновлення політики та функції цінності
        для кожної з K епох:
            розділити trajectories на міні-батчі
            для кожного міні-батчу:
                обчислити:
                    r(θ) = π_θ(a_t | s_t) / π_θ_old(a_t | s_t)
                    L^CLIP = мін(r(θ) * Â_t, clip(r(θ), 1 - ε, 1 +
ε) * Â_t)
                    L^VF = (V_ϕ(s_t) - R_t)^2
                    L^S = ентропія[π_θ(.|s_t)]

                обчислити загальну функцію втрат:
                    L_total = -L^CLIP + c1 * L^VF - c2 * L^S

                оновити параметри θ та ϕ за допомогою градієнтного
спуску

        # 5. Перевірка завершення
        якщо середня винагорода > порогу або досягнуто макс.
ітерацій:
            break

    # 6. Збереження моделі
    зберегти навчені параметри політики θ
```

Опис алгоритму PPO:

1. Ініціалізація. На даному кроці створюється початкова політика (наприклад, випадкові ймовірності вибору дій) і функція цінності. Визначаються гіперпараметри: розмір оновлення політики (clipping parameter), кількість епізодів для тренування, розмір батчу тощо.

2. Збір даних. NPC взаємодіє з середовищем протягом кількох епізодів, обираючи дії на основі поточної політики. Для кожної дії записуються: стан, обрана дія, отримана винагорода, і оцінка цінності стану.

3. Обчислення переваги (Advantage Estimation). Даний крок показує, наскільки обрана дія краща за середню в даному стані.

4. Оновлення політики. В даному кроці PPO використовує спеціальну функцію втрат (loss function), яка включає три компоненти:

– Clipped Surrogate Objective: Основна частина, яка обмежує оновлення політики, щоб уникнути занадто різких змін [2, 17, 29].

– Value Function Loss: Для оновлення функції цінності.

– Entropy Bonus: Для забезпечення різноманітності дій [15].

5. Оновлення функції цінності. Дана тренується, щоб мінімізувати різницю між передбаченою цінністю стану та реальною сумарною винагородою.

6. Повторення. Вест процес повторюється протягом заданого числа ітерацій або доки NPC не досягне бажаної поведінки (наприклад, максимальної сумарної винагороди).

До переваги PPO слід віднести його стабільнісь - обмеження на оновлення політики дозволяє уникнути "стрибків", які часто псують навчання в інших алгоритмах. Він також ефективний у складних середовищах, особливо тих, де дуже багато можливих дій і станів [2, 6]. До того ж PPO простий у реалізації — його легше запрограмувати й налаштувати порівняно з іншими сучасними методами, такими як TRPO.

Попри свої переваги, PPO не позбавлений і недоліків. Він вимагає багато обчислювальних ресурсів через використання нейронних мереж і великі обсяги даних. Навчання може тривати досить довго, особливо в складних сценаріях. Крім того, результат значною мірою залежить від гіперпараметрів — якщо вибрати їх невдало, агент може так і не навчитися ефективно діяти [16, 18].

## 2.3 Інформаційні моделі

Всі спостереження, які передаються до нейронної мережі агентів червоної команди, формуються динамічно в рамках логіки події Event Tick у Blueprint-системі, як показано в додатку А. Саме тут, у реальному часі, здійснюється обробка інформації про навколишнє середовище, щоб агент мав актуальну

картину гри перед прийняттям рішення. Логіка поділяється на дві основні гілки: одна відповідає за обробку даних про супротивників (агентів синьої команди), інша — за дані про гравця.

У першій гілці реалізовано механізм збору інформації про живих агентів синьої команди. На початку кожного тіку очищуються всі пов'язані змінні AliveBlueActor що відповідає за посилання на останнього виявленого живого супротивника, змінна AliveBlueLocation відповідає за координати цього агента, AliveBlueDirection відповідає вектор напрямку (forward vector), що вказує, куди він дивиться.

Після цього запускається цикл For Each, який перебирає масив BlueActor[], що містить усіх агентів синьої команди. Якщо виявляється, що конкретний агент мертвий (тобто змінна IsDead має значення true), система переходить до наступного агента. Але якщо супротивник живий, то відповідні змінні оновлюються: в AliveBlueActor зберігається посилання на об'єкт, у AliveBlueLocation — його координати через GetActorLocation, а у AliveBlueDirection — напрямок погляду за допомогою GetActorForwardVector. Варто зазначити, що в результаті цього процесу зберігається інформація про останнього в масиві живого супротивника, тобто при наявності кількох живих агентів дані будуть перезаписані на останнього знайденого.

У другій частині логіки, реалізується аналогічний механізм, але вже для актора-гравця (PlayerActor). Спочатку очищуються змінні AlivePlayerActor, AlivePlayerLocation, AlivePlayerDirection.

Далі система запускає For Each цикл для масиву PlayerActor[]. Хоча зазвичай гравець у грі один, структура передбачає можливість підтримки кількох екземплярів (наприклад, у багатокористувацьких режимах або для масштабування). Якщо певний гравець виявляється невалідним або мертвим — він ігнорується. У випадку, якщо гравець валідний, система зберігає посилання на нього у змінну AlivePlayerActor, його координати у AlivePlayerLocation, а напрямок руху — у AlivePlayerDirection, аналогічно до логіки з агентами [4].

Усі ці операції виконуються миттєво, що гарантує постійну актуальність переданої інформації до нейронної мережі. Незалежно від змін ситуації на полі бою — загибелі супротивників чи зникнення гравця — логіка Event Tick автоматично забезпечує подачу тільки актуальних даних. Завдяки цьому агенти червоної команди отримують достовірні спостереження навіть у динамічному середовищі, де склад об’єктів може змінюватися щосекунди. UML-діаграма активності, яка повністю охоплює логіку представлена на рисунку 2.2.

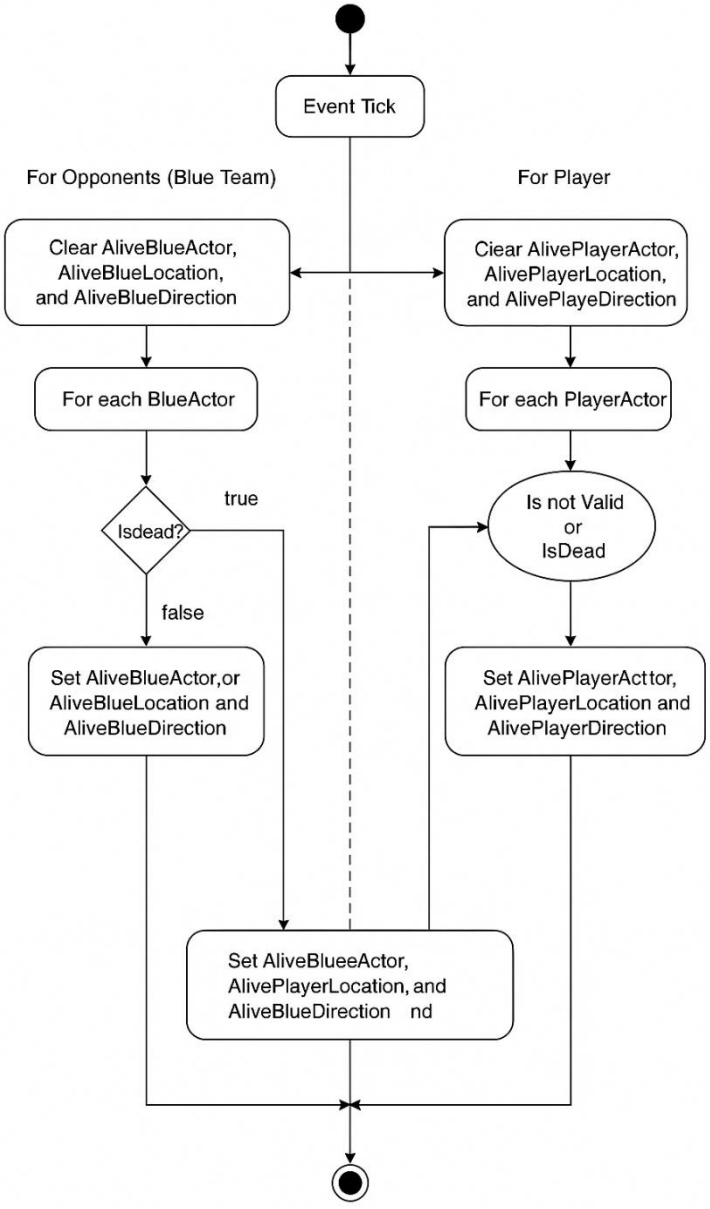



Рисунок 2.2 – UML-діаграма активності

Такий підхід дозволяє забезпечити стабільність у навчанні моделі PPO, оскільки дані, що надходять на її вхід, завжди структуровані, очищені від неактуальних об’єктів і містять лише релевантну інформацію про оточення.

## 2.4 Blueprint-компоненти інтелектуальної системи

Архітектура модуля адаптивної поведінки побудована з урахуванням принципів модульності, повторного використання компонентів та повної реалізації логіки через Blueprint-систему Unreal Engine 5.4.4.

Загальна структура модуля складається з чотирьох основних компонентів (Blueprint-класів), які реалізовано окремо для кожної команди (червона та синя), призначення яких зображено в таблиці 2.1.

Таблиця 2.1 – Компоненти структури модуля і їх призначення

| Компонент | Призначення |
|---|---|
| Agent Blueprint | Фізична оболонка NPC. Містить колізії, анімації, інтерфейс руху та стрільби. |
| Interactor Blueprint | Формує спостереження, приймає дії, виконує трансляцію рішень агента. |
| Trainer Blueprint | Налаштовує параметри PPO, визначає нагороди, обчислює політику. |
| Agent Manager Blueprint | Спавн агентів, контроль стану команди, респаун, командні лічильники. |

Для обох команд ці компоненти реалізовані незалежно, що дозволяє задавати різну логіку спостережень, винагород і контролю дій. Нижче представлено загальну архітектуру модуля у вигляді схеми на рисунку 2.3.

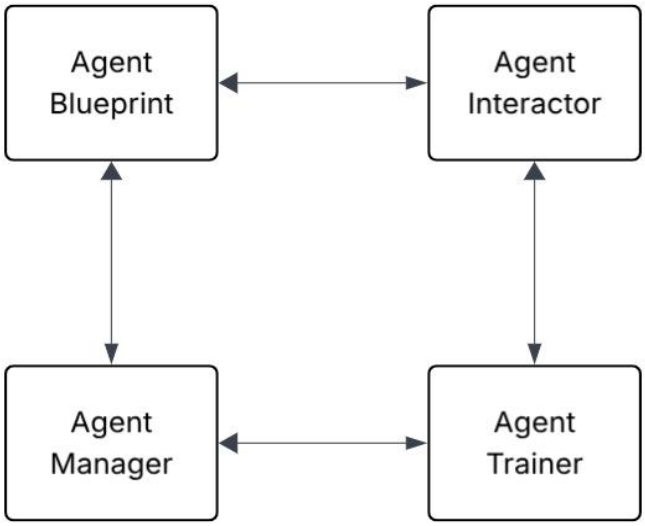



Рисунок 2.3 – Схема архітектури компонентів агентів

У межах архітектури навчання з підкріпленням важливу роль відіграє Agent Manager — компонент, що відповідає за ініціалізацію агентів у ігровому світі, контроль їх появи, координацію дій і відстеження повного життєвого циклу NPC. Саме цей Blueprint стежить за тим, щоб кожен агент був правильно створений, коректно функціонував у межах гри, а також щоб система могла реагувати на події, як-от смерть або завершення епізоду.

Фізичну присутність неігрового персонажа у світі гри реалізовано через Agent Blueprint. Цей компонент містить не лише модель персонажа, а й визначає базові можливості агента: його рух, взаємодію з фізичним середовищем, орієнтацію та анімацію. Це — оболонка, через яку агент "взаємодіє" з ігровим простором.

Дані для навчання та ухвалення рішень проходять через Interactor Blueprint. Він відповідає за збір усієї необхідної інформації про середовище — координати, напрямки, позиції ворогів і гравця. Ця логіка реалізується через функцію Gather Agent Observation. Окрім того, Interactor також застосовує дії, які обирає агент: наприклад, запускає рух, зміну напрямку камери або стрільбу.

Нарешті, Trainer Blueprint — це компонент, який з’єднує усе воєдино (див. додаток Б). Він приймає спостереження від Interactor’а, оцінює наслідки дій агента через систему винагород, і навчає PPO-модель на основі отриманого досвіду.

У цьому блоці реалізовано основну логіку тренування: збирання досвіду, обчислення втрат і оновлення політики відповідно до алгоритму Proximal Policy Optimization [2, 4].

Завдяки цій взаємодії між компонентами система стає повноцінною — від створення агента, до його навчання і застосування поведінки у грі.

Робочий цикл агента роботи одного агента виглядає наступним чином:

1. На кожному тіку (Tick) Interactor формує Observation — координати, напрямки, ворожі агенти, гравець, стіни.
2. Передає спостереження до PPO-моделі через Trainer.
3. Отримує зворотну дію (Action) від політики:

Червоні агенти: рух, поворот, стрільба.

Сині агенти: рух, поворот; стрільба — автоматична.

4. Виконує дію через Blueprint-команди руху та повороту.
5. Trainer оцінює наслідки дії й видає Reward, що впливає на подальше навчання.
6. Manager оновлює загальний стан команди, рахунок, та перевіряє на завершення раунду.

Усі компоненти логічно відокремлені, що дозволяє легко масштабувати систему. Агент може бути замінений або перестворений без зміни загальної архітектури.

Trainer реалізовано з можливістю гнучкого налаштування:

- Rewards для конкретних подій (вбивство, смерть, зіткнення);
- кількість Training Steps на епізод;
- гіперпараметри PPO: learning rate, clip epsilon, gamma тощо [2, 4].

Архітектура тісно інтегрована з PPO через Trainer. Навчання проходить у симульованих епізодах (реальному або прискореному часі). Після кожного епізоду накопичується досвід (trajectories), який використовується для оновлення параметрів нейронної мережі.

Завдяки модульній структурі, систему можна адаптувати під інші алгоритми (наприклад, A2C чи DQN) або додати нові типи дій без необхідності переписування всієї архітектури [23, 29].

## 2.5 Побудова навчального середовища та генерація даних

У системах навчання з підкріпленням якість результатів моделі безпосередньо залежить від способу формування та структури навчальних даних. У відмінність від традиційного машинного навчання, де дані зазвичай є наперед зібраними, у RL-моделях агент самостійно генерує досвід під час взаємодії зі середовищем. Цей процес називається rollout (розгортання політики) [4].

У проекті, реалізованому на рушії Unreal Engine 5.4.4, з використанням плагіну Learning Agents, генерація досвіду відбувається автоматизовано в процесі симуляції. Нижче описано логіку та технічну реалізацію збору даних.

Агенти взаємодіють із середовищем в реальному часі. Кожна така взаємодія утворює кортеж даних:

$$(s_t, a_t, r_t, s_{t+1}),$$

де $s_t$ — вектор спостережень у момент часу t,

$a_t$ — дія, обрана агентом відповідно до поточної політики,

$r_t$ — винагорода, яку отримав агент після виконання дії,

$s_{t+1}$ — новий стан після взаємодії з середовищем.

Ці кортежі формують trajectories (шляхи), які зберігаються у внутрішній буфер та використовуються під час оновлення моделі PPO [2, 4].

У системі навчання агентів важливу роль відіграє механізм збору спостережень, реалізований у компоненті Blueprint Interactor. Саме через нього агент отримує інформацію про стан навколишнього середовища, що слугує вхідними даними для нейронної мережі.

Цей процес виконується регулярно — на кожному тіку (Tick) або кроці симуляції (Step) викликаються спеціальні функції. Основна з них — Gather Agent Observation — відповідає за зчитування положення NPC, напрямків їх руху, а також перевірку, чи залишаються об’єкти (агенти супротивника або гравець) живими. Паралельно з цим функція Specify Agent Observation задає точний формат переданих даних: визначається тип векторів, кількість числових компонентів, структура спостереження тощо (див. додаток Б).

Серед усієї множини доступної інформації, найважливішими параметрами для навчання агента є такі:

– Просторові характеристики, а саме координати NPC та гравця, а також вектори їхнього напрямку.

– Геометричні обмеження, зокрема розташування статичних стін, які враховуються під час прийняття рішень, особливо для агентів синьої команди.

– Статус об’єктів, що дає змогу визначити, чи об’єкт ще активний (живий), чи його вже слід виключити зі спостережень [4, 5].

Завдяки цій логіці агент отримує лише релевантну та структуровану інформацію, що забезпечує стабільну роботу навчального процесу навіть у динамічному ігровому середовищі.

Після того як агент отримує спостереження про навколишнє середовище, на основі цих даних нейронна мережа, що функціонує в компоненті Trainer, генерує відповідну політику поведінки. Саме ця політика визначає, яку дію слід виконати у поточному стані. Обрана дія передається у Blueprint Interactor, який, у свою чергу, застосовує її до фізичної моделі агента, реалізованої в Agent Blueprint.

Залежно від того, до якої команди належить агент, список можливих дій відрізняється. Червоні агенти мають повний контроль, що надається нейронною мережею. Вони здатні переміщуватися, обертати камеру та виконувати стрільбу — усі ці дії регулюються безпосередньо алгоритмом PPO.

У випадку з синіми агентами контроль частково обмежений. Вони так само здійснюють переміщення та зміну орієнтації у просторі, однак стрільба у них відбувається автоматично. Це реалізовано за допомогою функції Trace by Channel, яка спрацьовує, коли агент самостійно наводиться на суперника [5].

Таким чином, логіка виконання дій відрізняється між командами, що дозволяє моделювати різні рівні автономності та досліджувати вплив ступеня контролю на результати навчання.

У процесі навчання з підкріпленням одним із ключових елементів є система нарахування винагород (reward-сигналів). У даному проєкті ця система реалізована через Blueprint-компонент Trainer, де за допомогою спеціальних Reward функцій визначається, скільки очок отримає агент за певну дію чи подію в грі.

На кожному етапі симуляції, протягом окремого епізоду, тренер обраховує сумарну винагороду агента на основі того, що відбулося під час гри. До прикладу, агент може отримати негативну винагороду за зіткнення зі стіною, позитивну — за влучання у ворога, або штраф — за наведення прицілу на союзника чи власну загибель. Усі ці події відстежуються у режимі реального часу завдяки логіці, закладеній у Event Tick та Agent Manager, які слідкують за станом агента, його діями та подіями, що з ним відбуваються.

У результаті зібрані винагороди зберігаються у буфері досвіду, де накопичуються до моменту оновлення політики нейронної мережі за алгоритмом PPO. Саме на основі цих даних модель навчання аналізує, які дії були ефективними, а які — ні, і відповідно коригує поведінку агента [4]. Такий підхід дозволяє забезпечити гнучке, динамічне навчання в середовищі з постійно змінними умовами.

Кожен навчальний епізод розпочинається з того, що агенти з’являються у середовищі, а середовище ініціалізується. На кожному кроці кожен агент формує спостереження, виконує дію та отримує винагороду. Епізод триває до тих пір, поки не буде досягнуто граничної кількості кроків, не станеться знищення всіх агентів однієї з команд або не закінчиться час (timeout). Після завершення епізоду дані траєкторій зберігаються для алгоритму PPO, виконується крок оновлення ваг політики, а потім запускається новий епізод із новими стартовими умовами [2].

Learning Agents дозволяє одночасне тренування кількох агентів. У грі таких агентів 16 (по 8 у кожній команді), тому на кожному тіку формується масив досвіду, що значно прискорює навчання і підвищує узагальнення [4].

# 3 РЕАЛІЗАЦІЯ І ТЕСТУВАННЯ МОДУЛЯ АДАПТИВНОЇ ПОВЕДІНКИ

## 3.1 Реалізація агента в Unreal Engine

Програмну реалізацію модуля адаптивної поведінки віртуального супротивника виконано у середовищі Unreal Engine 5.4.4 із використанням плагіну Learning Agents, що підтримує інтеграцію алгоритмів навчання з підкріпленням (RL), зокрема Proximal Policy Optimization (PPO) [4]. Уся логіка реалізована на Blueprint-системі (див. додаток Б).

Для кожної команди (червона і синя) реалізовано чотири основні компоненти у вигляді Blueprint-класів зображених в таблиці 3.1:

Таблиця 3.1 - Чотири основні компоненти

| Компонент | Призначення |
|---|---|
| Agent Blueprint | Фізичне тіло NPC, обробка руху, повороту, анімацій, стрільби. |
| Agent Interactor | Формування спостережень, отримання дій, їх застосування. |
| Agent Trainer | Визначення простору дій, налаштування PPO, винагород. |
| Agent Manager | Створення агентів, контроль станів, завершення епізодів, запуск тренування. |

Усі компоненти мають окрему реалізацію для обох команд, хоча налаштування PPO (модель, гіперпараметри, архітектура мережі, ігрові параметри) ідентичні [2].

Спостереження (Observation Space) задаються у функції Specify Agent Observation, а оновлюються в Gather Agent Observation у Blueprint Agent Interactor. Червона команда отримує координати та напрям власного тіла, а також позиції та напрямки живих синіх агентів, які визначаються через масив BlueActor[], і позицію та напрямок живого гравця, що отримуються через

PlayerActor[]. Синя команда, крім того, враховує положення стін, зокрема координати найближчих перешкод, оскільки стрільба у них відбувається автоматично. Збір спостережень виконується в Event Tick через послідовну перевірку життєздатності кожного агента чи гравця, а останній виявлений живий об’єкт зберігається у змінні типу Actor або Vector.

Дії агента формуються нейронною мережею PPO і передаються через функцію Specify Agent Action, а потім реалізуються за допомогою Perform Agent Action. Для червоної команди передбачені такі дії: Move Forward (значення типу float від -100 до 100), Turn Camera (значення типу float) і Should Shoot (значення типу bool), яке використовується для виклику події стрільби. Для синьої команди доступні дії Move Forward і Turn Camera, як і в червоної команди, але стрільба реалізується автоматично через Trace by Channel при наведенні на ворога. Усі ці дії застосовуються до Agent Blueprint за допомогою вузлів Add Movement Input, Add Controller Yaw Input і Fire Weapon.

У Agent Trainer реалізовано функцію Compute Reward, яка обчислює винагороди на основі подій, що відбуваються під час гри. Для червоної команди винагороди розподіляються наступним чином: +10 нараховується за наведення на ворога, +20 — за наведення на гравця, -1 — за наведення на союзника, -1 — за зіткнення зі стіною, а -2.5 — за смерть. Для синьої команди винагороди визначаються так: +10 за наведення на червоного бота, +20 за наведення на гравця, -5 за зіткнення зі стіною і -2.5 за смерть. Події, які впливають на ці винагороди, фіксуються за допомогою Line Trace, Overlap та внутрішньої логіки в Blueprint Agent Manager.

У Blueprint Agent Manager реалізовано ряд вузлів: Make Interactor, Make Policy, Make Critic і Make Trainer. До вузла Run Training підключено наступні налаштування: Trainer Settings із параметрами MaxEpisodeSteps = 512, MaxEpisodesPerIter = 1000 і MaxStepsPerIter = 10000; Trainer Training Settings із запланованою кількістю 1 мільйон ітерацій і параметрами PPO, такими як $\varepsilon = 0.2$, $\gamma = 0.99$ і $\lambda = 0.95$; а також Trainer Game Settings, які включають фіксований

таймстеп (60 FPS), відключений VSync і фонові обмеження [2, 4]. Архітектура моделі PPO складається з політики (Policy) із одним прихованим шаром із 128 нейронами, активацією ELU та увімкненою пам’яттю розміром 64, і критика (Critic) з одним шаром із 128 нейронами, активацією ELU, але без пам’яті. Усі обчислення виконуються на GPU [2].

## 3.2 Взаємодія між агентом та середовищем гри

У системах із навчанням з підкріпленням взаємодія між агентом (NPC) та середовищем є критично важливою складовою навчального циклу. Вона визначає, як агент сприймає стан гри, приймає рішення, впливає на оточення та отримує відповідну винагороду. Реалізовано повноцінну інтеграцію агентів у ігрове середовище Unreal Engine 5.4.4, де NPC діють у реальному часі, спираючись на обчислення нейронної мережі PPO, навчену через плагін Learning Agents [4].

Ігрове середовище являє собою арену, де змагаються дві команди агентів — червона та синя, кожна з яких складається з восьми NPC, а також один гравець, позначений як сірий, якого може керувати користувач або сценарій. Карта арени включає перешкоди у вигляді стін, відкриті зони та визначені стартові позиції для кожної з команд. Середовище характеризується такими особливостями: частковою спостережуваністю, що означає, що NPC не можуть одночасно бачити всіх ворогів, динамічною зміною станів, яка включає рух об’єктів і смерті агентів, обмеженням видимості, через що NPC не мають доступу до повної карти, а також автоматизованим завершенням епізоду, яке відбувається за таймером.

Агент сприймає своє оточення через функцію Gather Agent Observation, реалізовану у Blueprint Agent Interactor. Щоразу під час оновлення, яке відбувається на кожному тику або кроці симуляції, агент отримує інформацію про власне положення та напрямок, координати та напрямки живих

супротивників, тобто інших NPC, а також положення живого гравця. Для агентів синьої команди додатково враховуються координати найближчих стін. Усі ці дані перетворюються у нормалізований числовий вектор і передаються до моделі PPO, яка на основі отриманих значень обчислює оптимальну дію.

Після отримання вектора спостереження нейронна мережа PPO у Trainer обчислює дію агента. Далі ця дія передається через Perform Agent Action до Agent Blueprint, де вона реалізується наступним чином: рух виконується за допомогою вузла Add Movement Input, обертання здійснюється через Add Controller Yaw Input, стрільба для червоних агентів активується шляхом виклику спеціального Custom Event типу Fire Weapon, а для синіх агентів стрільба відбувається автоматично через Trace by Channel на ворога, що перебуває в полі зору [1, 2].

Агенти пересуваються по мапі, наводяться на ворогів, здійснюють постріли, влучають у суперників, змінюють загальний рахунок своєї команди та запускають події, які призводять до завершення епізоду. У свою чергу, середовище реагує на дії агентів через фізичну взаємодію, зокрема колізії, логіку пошкоджень, події смерті, механізм респауну за та системи підрахунку очок [23, 31].

За результатами дій агента система винагороди, реалізована у Agent Trainer, обчислює числове значення Reward, яке використовується моделлю PPO для оновлення політики. Наприклад, постріл по ворогу приносить +10, загибель NPC призводить до штрафу -2.5, влучання по союзнику карається -1, а наведення на гравця нагороджується +20. Такий підхід створює замкнутий цикл зворотного зв’язку, у якому кожен NPC самостійно вдосконалює свою поведінку, спираючись на досвід взаємодії зі світом гри [1].

Коли завершуються умови раунду — усі агенти гинуть, настає таймаут або раунд завершується вручну, — функція End Episode викликається в Agent Manager. Після цього всі агенти респавняються на своїх стартових позиціях, очищаються змінні, стани та лічильники, а модель PPO зберігає траєкторії у буфер для подальшого навчання. Цей цикл повторюється протягом визначеної

кількості ітерацій, зазначених в параметрах Trainer Settings, забезпечуючи поступове вдосконалення адаптивної стратегії поведінки NPC [4].

### 3.3 Тестування адаптивної поведінки агентів та аналіз ефективності

Розглядаються метрики навчання з підкріпленням, де є показники досвіду, градієнти для різних частин нейронної мережі (критик, декодер, енкодер, політика) та функції втрат [19, 25].

Графік метрики досвіду experience/avg_return (Середня сукупна винагорода за епізод) зображено на рисунку 3.2. TrainerB (Сині боти, бірюзова лінія): Починає з вищих значень (близько -7), але демонструє велику волатильність, значне падіння, а потім коливання. До кінця тренування (кроки 1200-1440) спостерігається тенденція до зниження (більш негативні значення). Згладжене значення на кроці 1440: -50,3544. TrainerR (Червоні боти, сіра лінія): Починає з нижчих значень (близько -20), але показує більш стабільне зростання (менш негативні значення), особливо після кроку 800. До кінця тренування стабільно покращується. Згладжене значення на кроці 1440: -29,3109.

Червона команда демонструє кращий та більш стабільний прогрес у другій половині тренування і закінчує з кращим (менш негативним) середнім доходом.

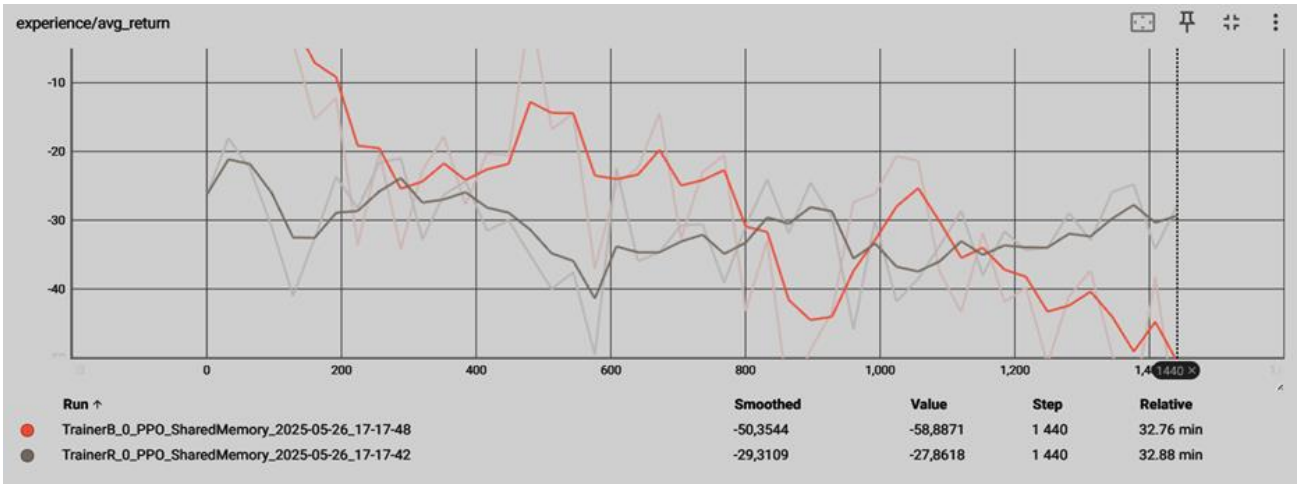



Рисунок 3.2 – Графік середньої сукупної винагороди за епізод

Графік experience/avg_reward (Середня винагорода за крок) зображено на рисунку 3.3: TrainerB (Сині): Схожа динаміка, як і в avg_return. Починає вище, але більш волатильна і з тенденцією до зниження наприкінці. Згладжене значення: -0,507. TrainerR (Червоні): Починає нижче, з чіткою тенденцією до покращення (менш негативні значення) у другій половині. Згладжене значення: -0,2448. Червона команда показує кращу середню винагороду за крок наприкінці тренування.

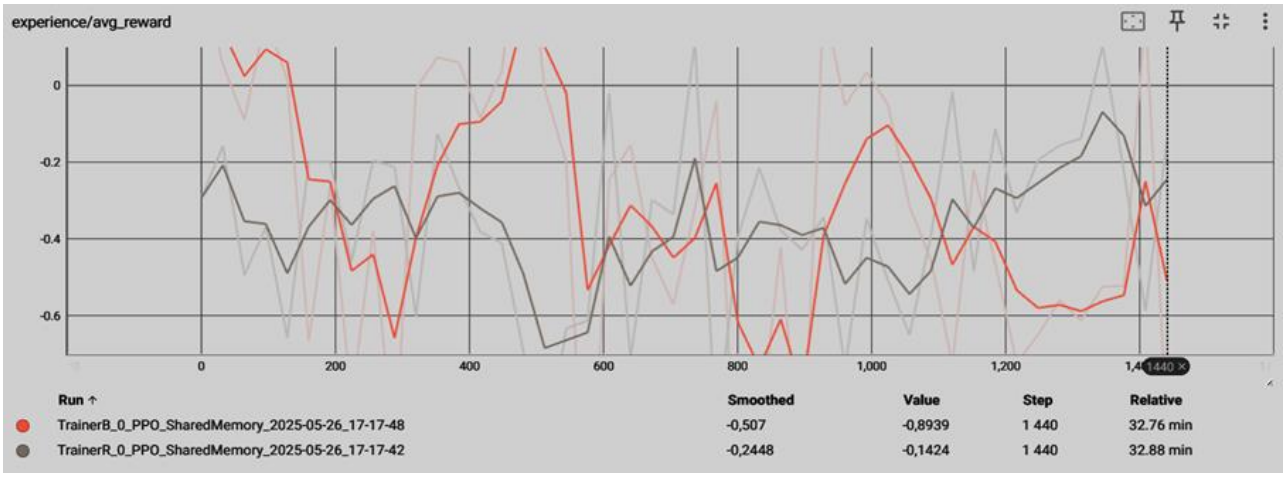



Рисунок 3.3 – Графік середньої винагороди за крок

Графік experience/avg_reward_sum (Сума середніх винагород) зображений на рисунку 3.4. TrainerB (Сині): Дуже схожа картина на avg_return, але з більш негативними значеннями по осі Y. Висока початкова волатильність, потім значне погіршення. Згладжене значення: -238,5515. TrainerR (Червоні): Також схоже на avg_return. Поступове покращення і стабілізація на кращих (менш негативних) значеннях. Згладжене значення: -121,2564. Червона команда значно перевершує синю за сумою середніх винагород наприкінці тренування.

Червона команда демонструє значно кращі результати навчання з точки зору стабільності та кінцевих показників винагороди. Синя команда хоч і може показувати короткочасні сплески, загалом її продуктивність знижується або залишається нестабільною.

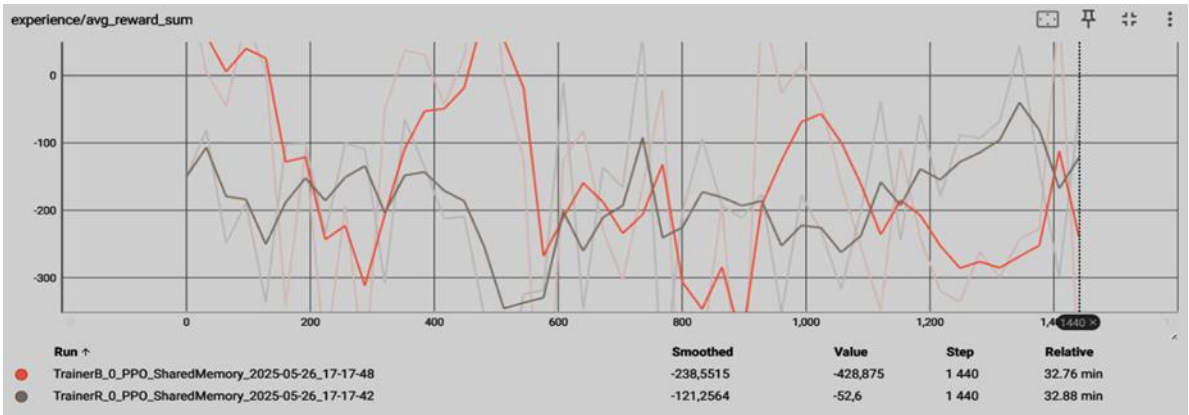



Рисунок 3.4 – Графік суми середніх значень винагород

Градієнти (Gradients)показують, наскільки сильно змінюються ваги мережі під час навчання. Дуже великі або дуже малі (зникаючі) градієнти вказують на проблеми. Графік grads/critic (Градієнти критика) зображено на рисунку 3.5. TrainerB (Сині): Має значно вищі піки градієнтів протягом усього тренування. TrainerR (Червоні): Демонструє набагато менші та стабільніші градієнти.

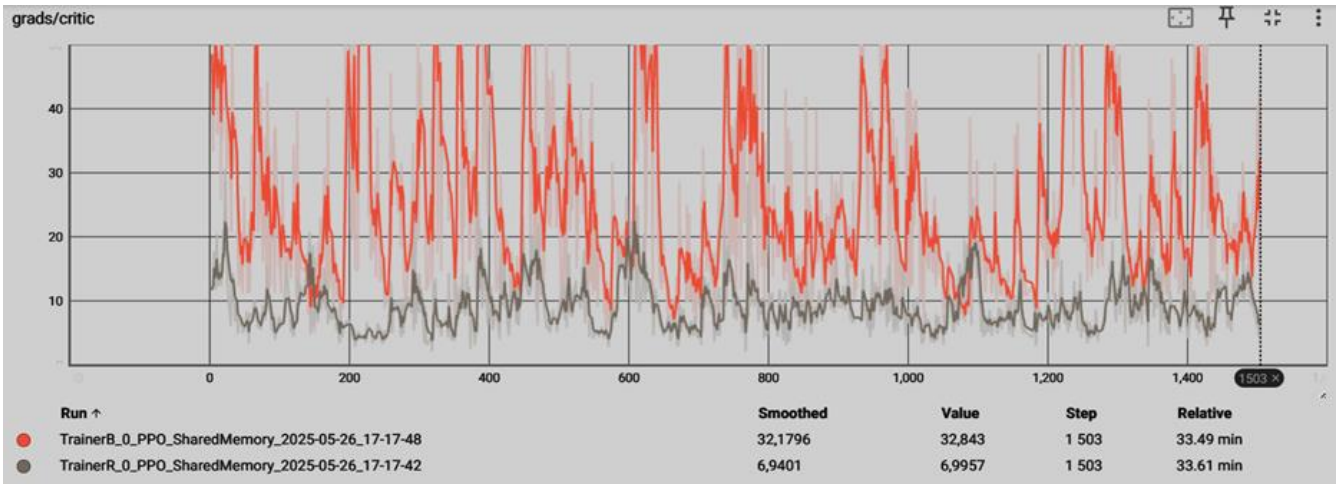



Рисунок 3.5 – Графік градієнтів критики

Градієнти червоної команди виглядають "здоровішими". Високі градієнти синьої команди можуть свідчити про нестабільність навчання або занадто великий крок навчання для критика [21, 32].

Графік grads/decoder (Градієнти декодера) зображено на рисунку 3.6. TrainerB (Сині): Показує вищі та більш "колючі" градієнти. TrainerR (Червоні): Градієнти нижчі та стабільніші.

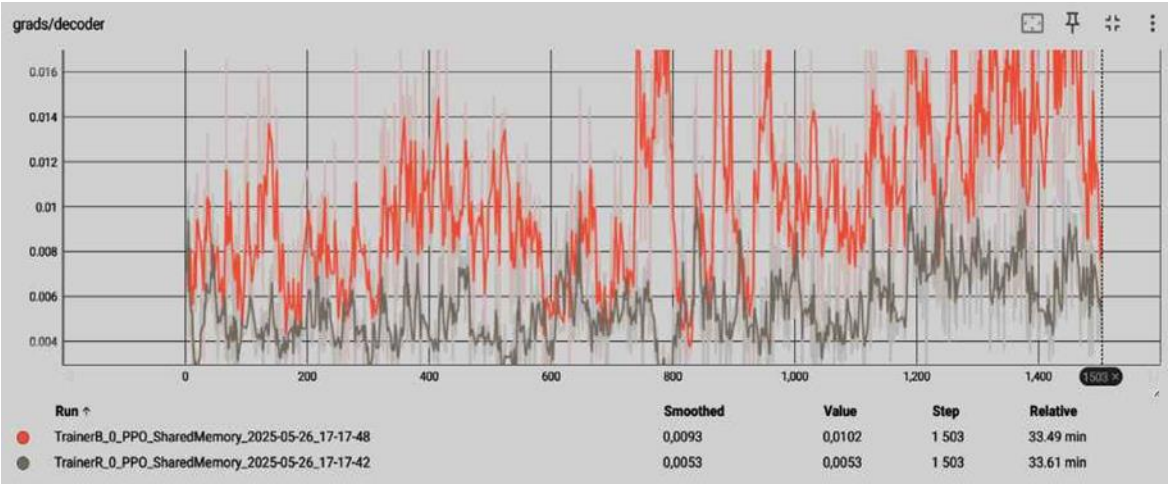



Рисунок 3.6 – Графік градієнтів декодера

Червона команда має стабільніші градієнти декодера. Графік grads/encoder (Градієнти енкодера) зображено на рисунку 3.7. TrainerB (Сині): Початково градієнти нижчі, але приблизно після кроку 600 вони стають значно більшими та нестабільними порівняно з червоною командою. TrainerR (Червоні): Початково градієнти вищі, але з часом стабілізуються і в другій половині тренування загалом нижчі та стабільніші, ніж у синьої команди.

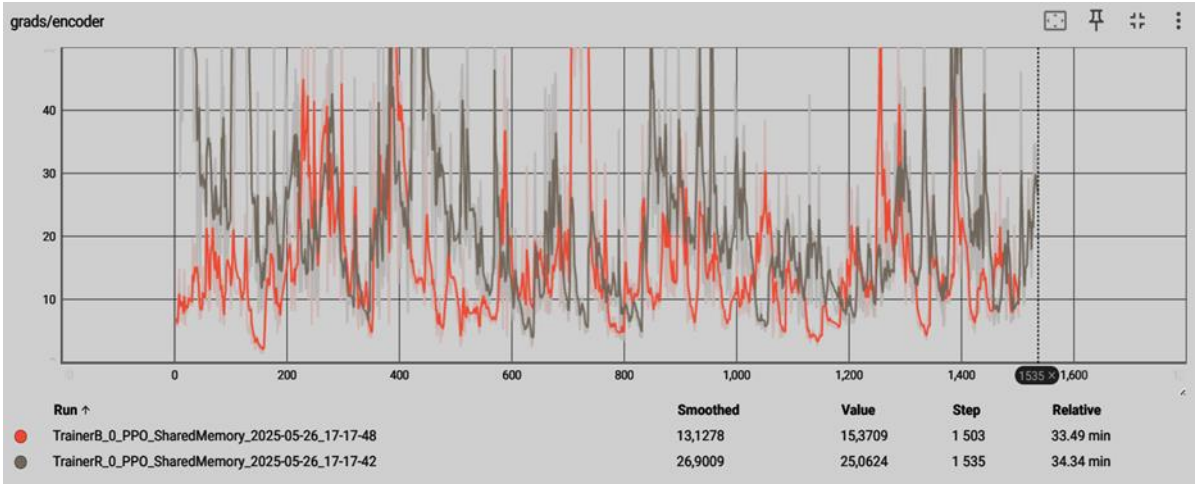



Рисунок 3.7 – Графік градієнтів енкодера

У довгостроковій перспективі червона команда показує стабільніші градієнти енкодера. Пізні високі градієнти синьої команди викликають занепокоєння.

Графік grads/policy (Градієнти політики) зображено на рисунку 3.8. TrainerB (Сині): Постійно демонструє вищі та більш мінливі градієнти політики. TrainerR (Червоні): Градієнти політики нижчі та стабільніші.

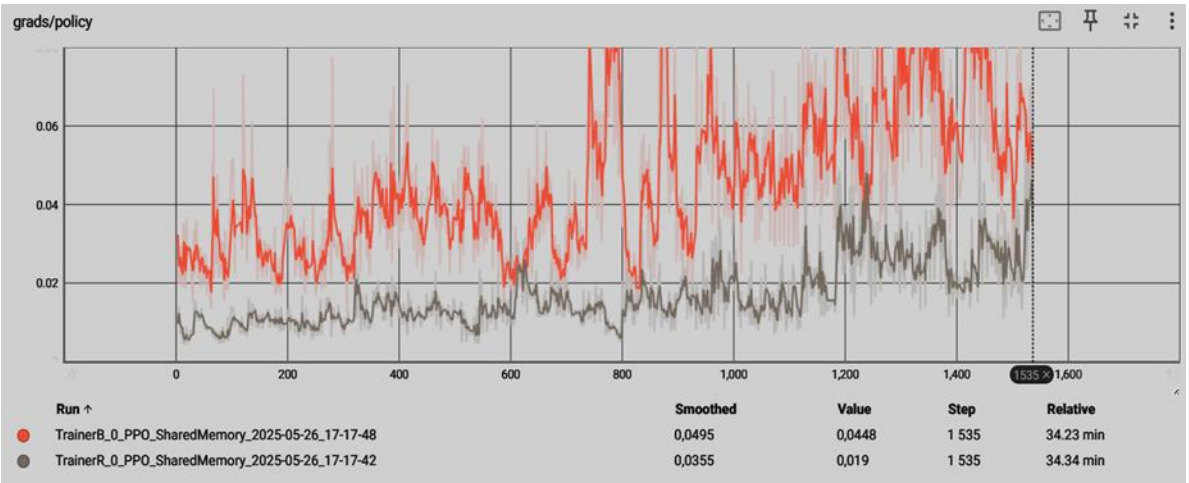



Рисунок 3.8 – Графік градієнтів політики

Градієнти політики червоної команди виглядають більш стабільними. Червона команда загалом демонструє набагато стабільніші та менші за амплітудою градієнти для всіх компонентів мережі. Це свідчить про більш плавний та ефективніший процес навчання. Великі та нестабільні градієнти синьої команди вказують на проблеми з навчанням.

Функція втрат (Loss Functions) показує, наскільки "погано" модель виконує завдання. Нижчі значення зазвичай кращі. Графіки loss/critic та loss/critic_reg (Втрати критика / Втрати критики якості навчання) зображено на рисунку 3.9 і 3.10: відповідно. Ці два графіки виглядають практично ідентичними. TrainerB (Сині): Має вищі піки втрат і, схоже, дещо вище середнє значення втрат. Коливання сильніші. TrainerR (Червоні): Демонструє нижчі піки втрат і, здається, дещо нижче середнє значення. Коливання менш виражені.

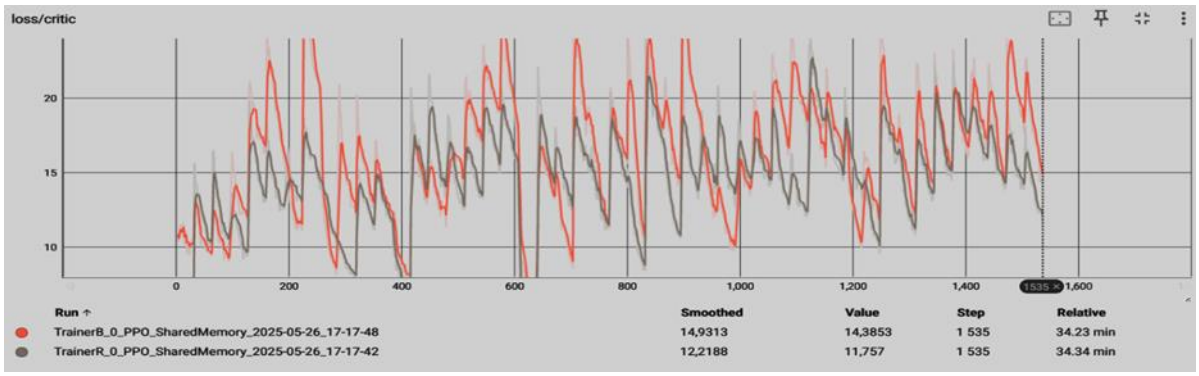



Рисунок 3.9 – Графік втрати критики

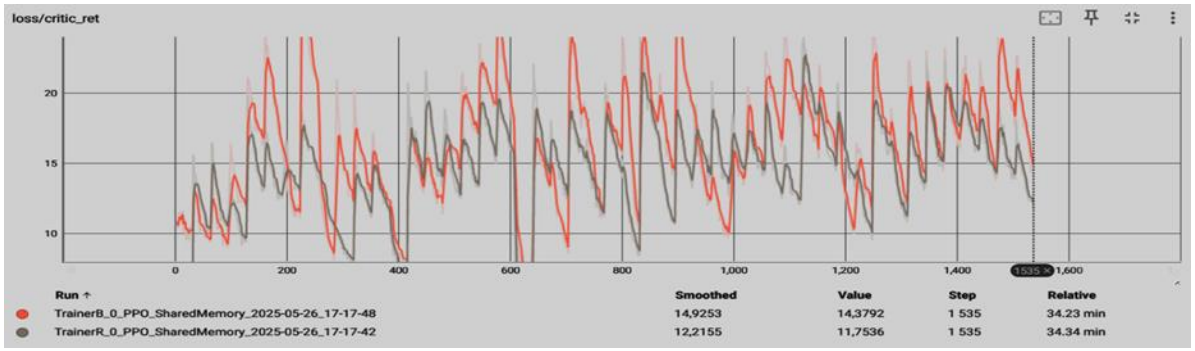



Рисунок 3.10 – Графік втрати критики якості навчання

Червона команда має трохи кращі (нижчі та більш обмежені) втрати критика. Обидві команди показують коливальний характер втрат, що є типовим для алгоритмів RL.

Графік loss/critic_reg (Регуляризаційні втрати критика) зображено на рисунку 3.11. TrainerB (Сині): Має значно вищі регуляризаційні втрати, особливо наприкінці тренування, де спостерігається різкий сплеск. TrainerR (Червоні): Регуляризаційні втрати набагато нижчі та стабільніші [19, 33].

Червона команда тут значно краща. Це свідчить про те, що її критик менш схильний до перенавчання, або його ваги менші.

Червона команда демонструє кращі показники втрат, особливо щодо регуляризації критика. Це підсилює висновок про більш стабільне та ефективне навчання.

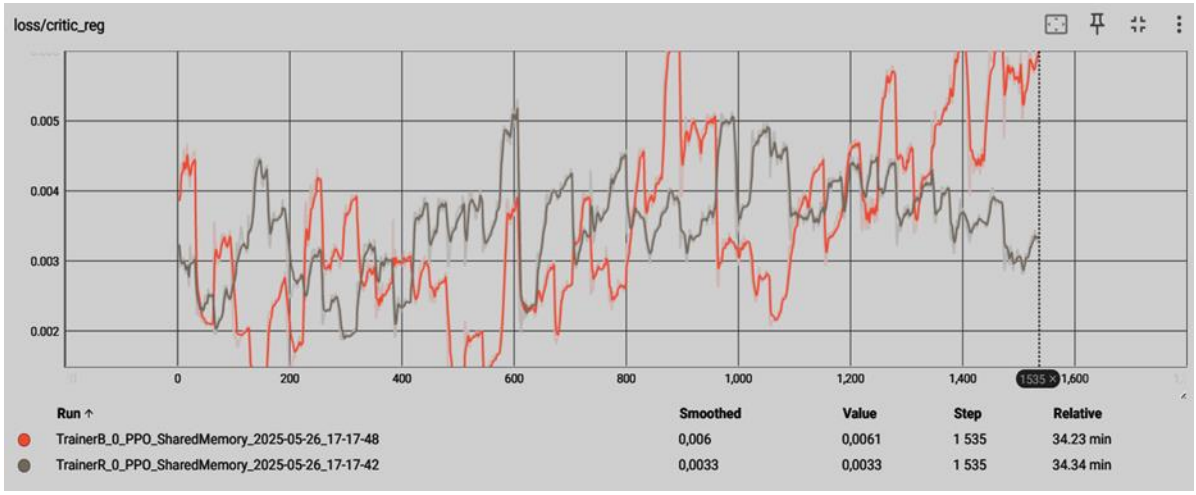



Рисунок 3.11 – Графік регуляційної втрати критики

Аналізуючи всі надані графіки, можна зробити наступні висновки:

– Продуктивність (Винагорода). Червона команда показує значно кращу та стабільнішу продуктивність. Її середні винагороди та сукупний дохід стабільно зростають або утримуються на кращому рівні, особливо в другій половині тренування. Синя команда хоч і може мати кращі стартові показники або короткочасні сплески, її продуктивність часто нестабільна і має тенденцію до погіршення.

– Стабільність Навчання (Градієнти). Червона команда демонструє набагато стабільніші градієнти для всіх компонентів моделі. Це свідчить про більш збалансований та контрольований процес навчання. Великі та непередбачувані градієнти синьої команди (TrainerB) можуть вказувати на проблеми, такі як занадто високий темп навчання або інші гіперпараметри, що потребують налаштування.

– Ефективність Навчання (Втрати). Червона команда також показує кращі характеристики функцій втрат, особливо значно нижчі регуляризаційні втрати критика. Це може вказувати на кращу генералізацію моделі та меншу схильність до перенавчання.

Червоні NPC контролюють стрільбу через власну дію Should Shoot, що дає конкретний зворотний зв’язок винагороди; у синіх стрільба автоматизована,

тому агент має менший вплив на ключову подію (влучання) і отримує шуми в сигналі винагороди [20 , 22, 24, 32].

Таким чином, червоні агенти демонструють кращу адаптивну поведінку та стабільність навчання порівняно з синіми агентами, незважаючи на однакові гіперпараметри PPO.

# ВИСНОВКИ

У межах даної кваліфікаційної роботи було розроблено, реалізовано та протестовано програмний модуль адаптивної поведінки супротивника в комп’ютерній грі з використанням штучного інтелекту, заснований на алгоритмі PPO (Proximal Policy Optimization) та вбудованому фреймворку Learning Agents рушія Unreal Engine 5.4.4. В роботі було:

1. Розроблено повнофункціональну систему штучного агента (NPC), здатного адаптивно реагувати на ситуацію в грі. Агент навчався взаємодіяти з іншими персонажами та середовищем з метою максимізації нагороди.

2. Сформовано інформаційну модель агента, що включає: простір спостережень (розташування, напрямки інших об’єктів), простір дій (рух, поворот, стрільба), функцію винагороди, яка відображає успішність дій агента.

3. Здійснено програмну реалізацію системи навчання на основі PPO: модулі Interactor, Trainer, Policy, Critic, Agent Manager створено повністю на Blueprints [4, 34].

4. Реалізовано дві команди NPC (червону та синю), які тренувалися незалежно при однакових гіперпараметрах PPO. Червоні агенти мали повний контроль над дією стрільби, тоді як у синіх вона активувалася автоматично при наведенні.

5. Проведено серію експериментів, під час яких зібрано й проаналізовано понад 10 метрик (avg_return, avg_reward, градієнти, функції втрат тощо). Це дало змогу порівняти якість та стабільність навчання обох систем.

Результати показали, що червона команда демонструє суттєво кращі характеристики навчання — стабільніші градієнти, нижчі втрати та вищі винагороди; Вплив NPC на результат має ключове значення — у разі, коли агент не контролює важливі дії (наприклад, стрільбу), сигнал винагороди стає шумним і менш ефективним для навчання; Blueprint-реалізація повністю підтримує

складні RL-сценарії, і дозволяє без кодування створювати адаптивну поведінку для NPC.

Отримані результати можуть бути застосовані в розробці комп’ютерних ігор, де важливо забезпечити непередбачувану, але розумну поведінку супротивника чи в розширенні системи: додати нові дії, змінити простір спостережень або переналаштувати політику.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. – 2nd ed. – MIT Press, 2018.

2. Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal Policy Optimization Algorithms. – arXiv:1707.06347 [cs.LG], 2017. – [Електронний ресурс]. – https://arxiv.org/abs/1707.06347

3. Champandard A. J. AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. – New Riders Publishing, 2003.

4. Unreal Engine Documentation. Learning Agents Plugin. – [Електронний ресурс]. – https://docs.unrealengine.com/5.4/en-US/learning-agents-overview/

5. Arulkumaran K., Deisenroth M. P., Brundage M., Bharath A. A. A Brief Survey of Deep Reinforcement Learning. – IEEE Signal Processing Magazine, 2017. – Vol. 34, No. 6. – P. 26–38. – [Електронний ресурс]. – Режим доступу: https://ieeexplore.ieee.org/document/8099231

6. Millington, I., & Funge, J. (2009). Artificial Intelligence for Games (2nd ed.). CRC Press.

7. Yannakakis, G. N., & Togelius, J. (2018). Artificial Intelligence and Games. Springer.

8. F.E.A.R. – Monolith Productions. F.E.A.R. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/F.E.A.R.).

9. Alien: Isolation – Creative Assembly. Alien: Isolation [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Alien:_Isolation.

10. Halo – Bungie, 343 Industries. Halo (серія ігор) [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Halo_(franchise)

11. The Last of Us Part II – Naughty Dog. The Last of Us Part II [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/The_Last_of_Us_Part_II

12. Nemesis System (Shadow of Mordor, Shadow of War) – Monolith Productions. Nemesis System [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Shadow_of_Mordor#Nemesis_System

13. NVIDIA ACE – NVIDIA Corporation. NVIDIA Avatar Cloud Engine (ACE) [Електронний ресурс] – Режим доступу: https://developer.nvidia.com/ace.

14. Unreal Engine Docs. Line Tracing and Hit Detection. – [Електронний ресурс]. – https://docs.unrealengine.com/

15. Mnih, V. et al. Playing Atari with Deep Reinforcement Learning // arXiv preprint arXiv:1312.5602, 2013. – [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/1312.5602

16. Lillicrap T. et al. Continuous Control with Deep Reinforcement Learning. – arXiv:1509.02971, 2015.

17. OpenAI. OpenAI Baselines: PPO2 Implementation. – [Електронний ресурс]. – GitHub, 2018. – URL: https://github.com/openai/baselines

18. Silver D. et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. – Nature, 2016.

19. Levine S. et al. End-to-End Training of Deep Visuomotor Policies. – Journal of Machine Learning Research, 2016.

20. Gleave A. et al. Adversarial Policies: Attacking Deep Reinforcement Learning. – NeurIPS, 2020.

21. Heess N. et al. Emergence of Locomotion Behaviours in Rich Environments. – arXiv:1707.02286, 2017.

22. Spaan M. Partially Observable Markov Decision Processes. – 2012.

23. Schmidhuber J. Deep Learning in Neural Networks: An Overview. – Neural Networks, 2015.

24. Sutton R. The Bitter Lesson. – [Електронний ресурс]. – URL: http://www.incompleteideas.net/IncIdeas/BitterLesson.html

25. Hafner D. et al. Dream to Control: Learning Behaviors by Latent Imagination. – ICLR, 2020.

26. Lilith Games. Warpath AI Developer Blog. – [Електронний ресурс]. – URL: https://www.warpath.com/devlog

27. Salimans T. et al. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. – arXiv:1703.03864, 2017.

28. Tremblay J. et al. Training Deep Networks with Synthetic Data. – CVPR Workshops, 2018.

29. Pathmind. PPO vs DDPG: Comparison of Reinforcement Learning Algorithms. – [Електронний ресурс]. – URL: https://pathmind.com/wiki/ppo

30. Rusu A. A. et al. Progressive Neural Networks. – arXiv:1606.04671, 2016.

31. OpenAI. AI and Compute. – [Електронний ресурс]. – URL: https://openai.com/research/ai-and-compute

32. Melis G. et al. On the State of the Art of Evaluation in Reinforcement Learning. – arXiv:1707.06887, 2017.

33. Rahmatizadeh R. et al. Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration. – ICRA, 2018.

34. Unreal Engine Forums. Blueprint vs C++ for AI in UE5. – [Електронний ресурс]. – URL: https://forums.unrealengine.com

35. GitHub. UnrealPPOTrainer Plugin Source Code Examples. – [Електронний ресурс]. – URL: https://github.com

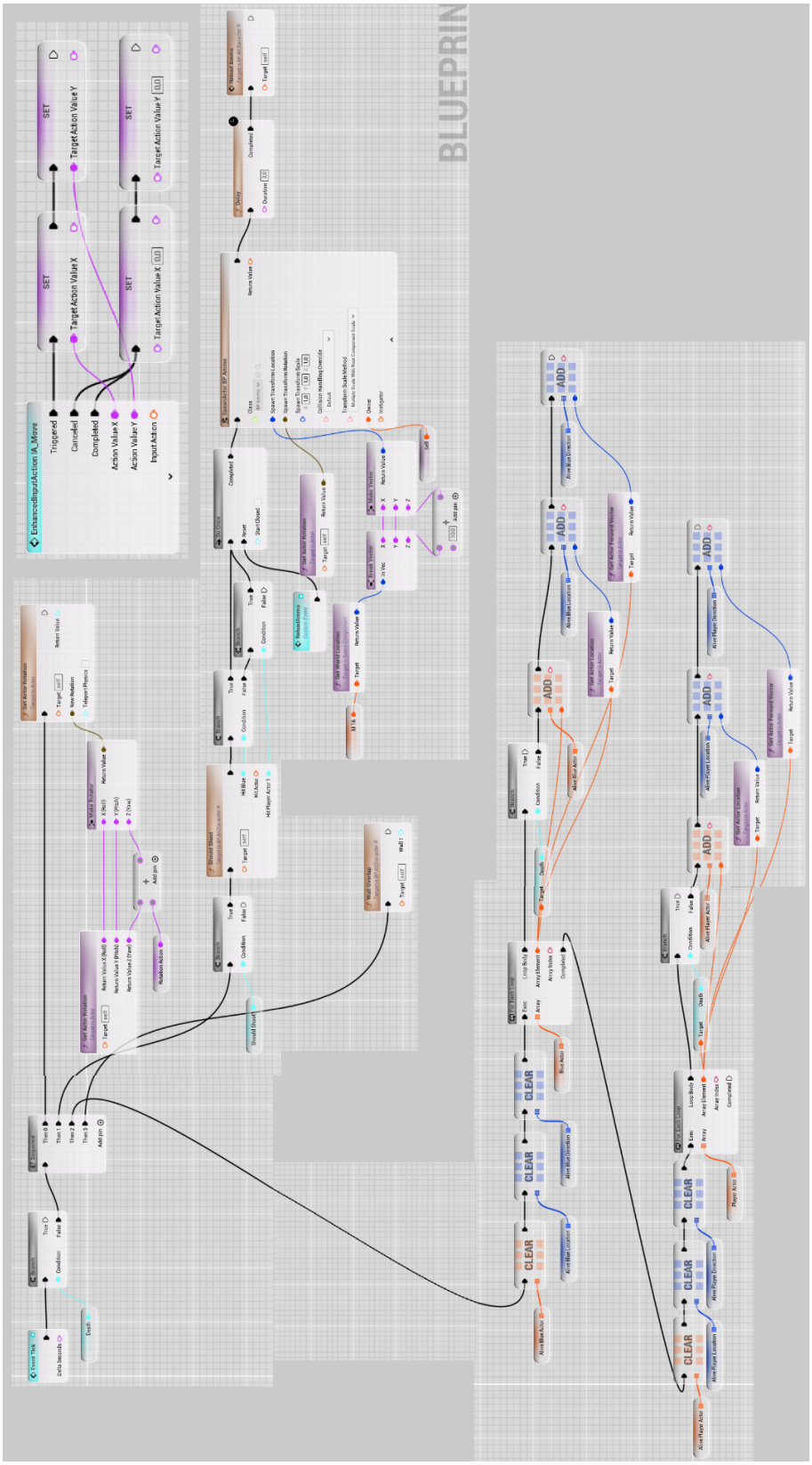
36. Васенко С.П., Биковий П.Є. Використання підходу behavior tree для створення адаптивних неігрових персонажів у навчальних відеоіграх. Інтелектуальні інформаційні технології в прикладних дослідженнях (IITAR-2025): збірник тез доповідей студентської науково-практичної конференції (Тернопіль, 27-29 травня 2025 р.). Тернопіль: ЗУНУ, 2025. С. 120-122 с.

37. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

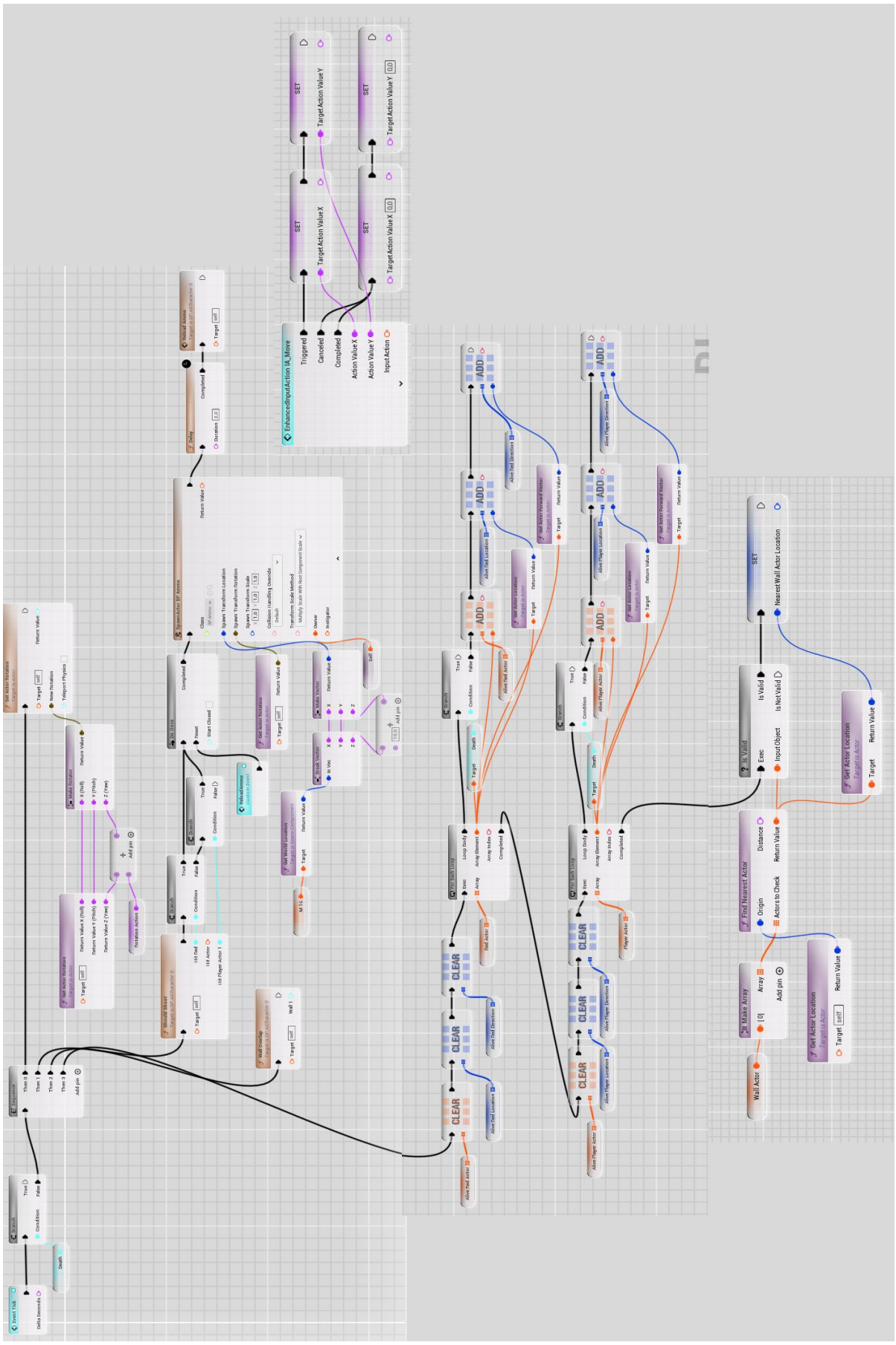
38. В. М. Островерхов, Л. І. Біловус, К. З. Возьний, О. О. Луцишин, Г. Л. Монастирський, С. А. Надвиничний, С. В. Питель,С. К. Шандрук., Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

Додаток А

Алгоритми оновлення змінних та поведінки агентів



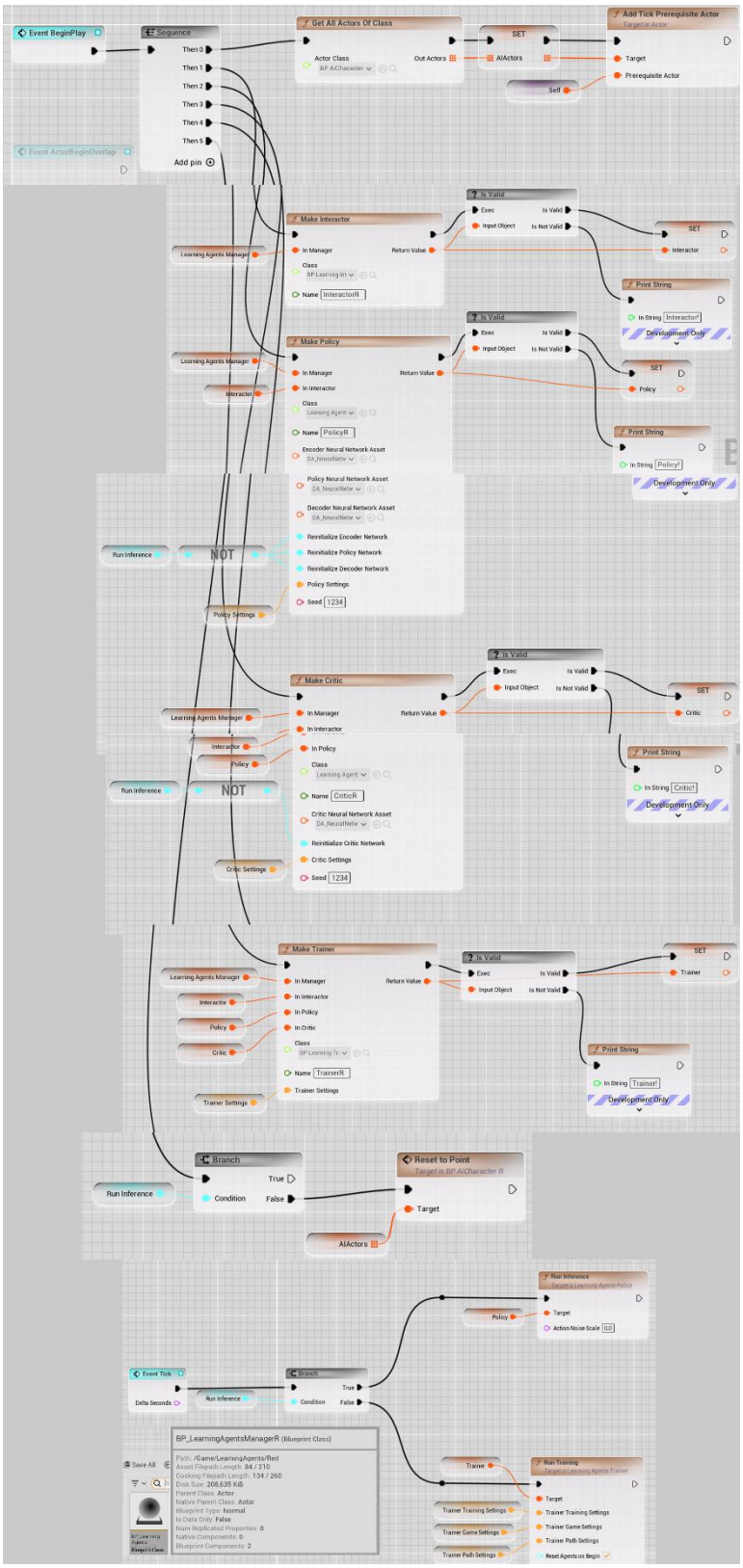
Blueprint червоної команди



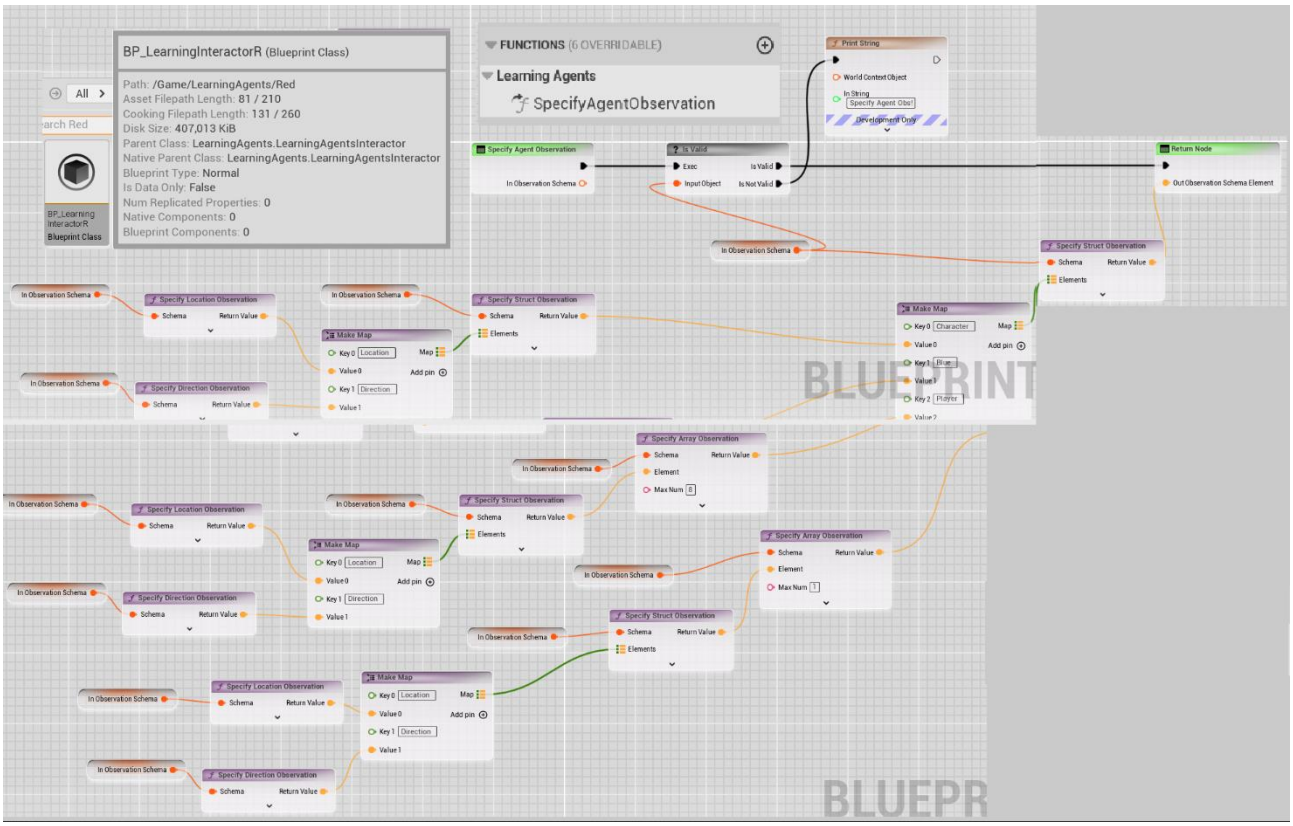
Blueprint синьої команди

Додаток Б

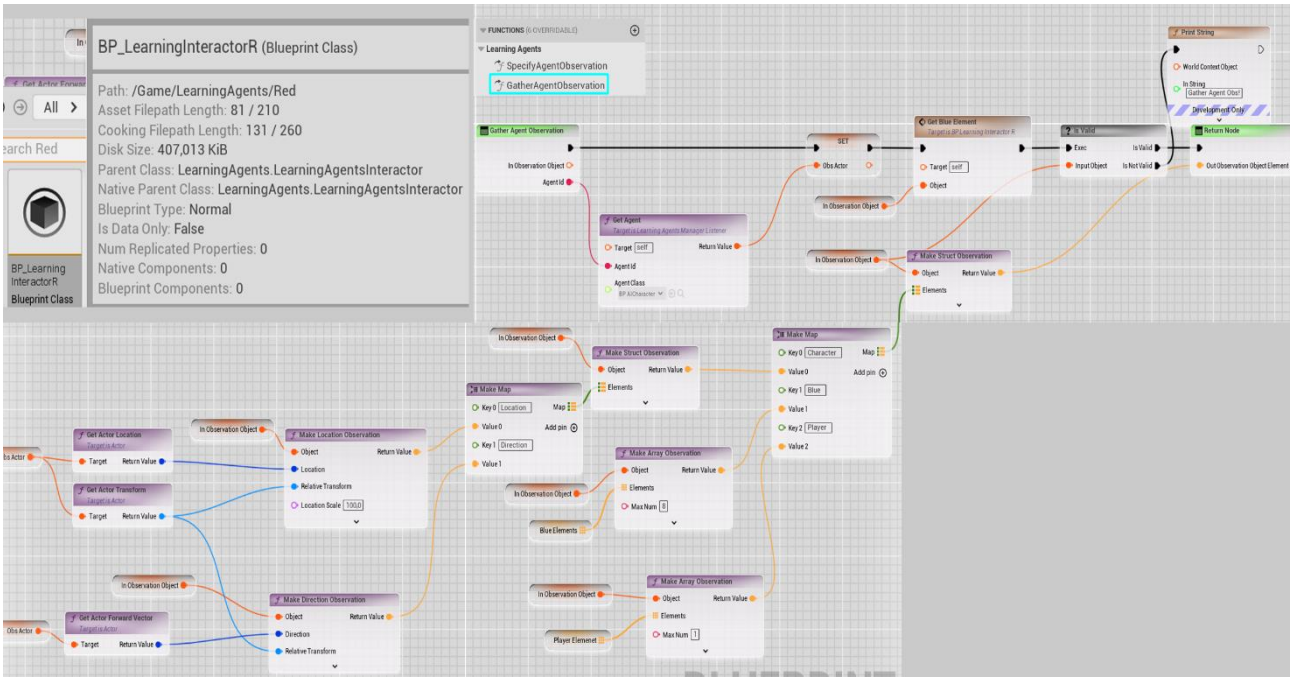
Реалізація програмного модуля



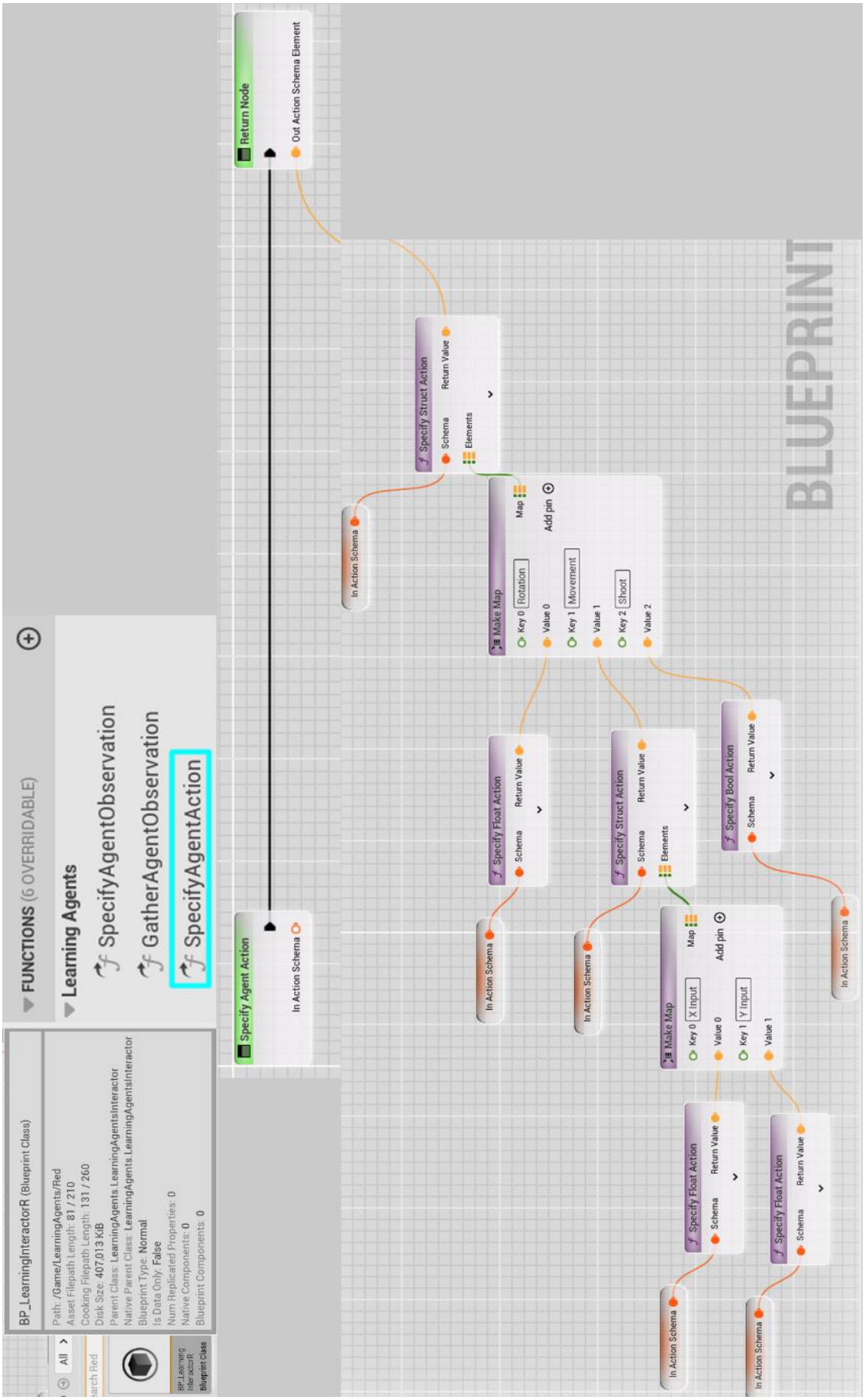

Blueprint Agent Manager червоної команди



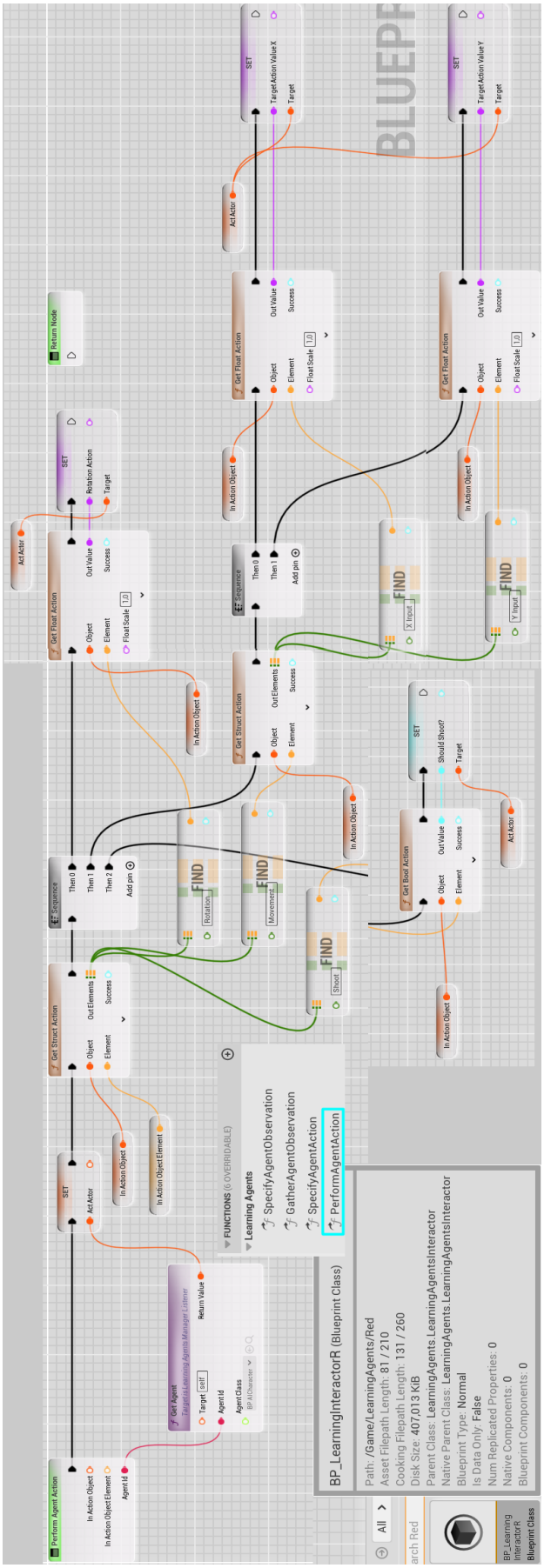

Blueprint Agent Interactor, функція Specify Agent Observation червоної команди



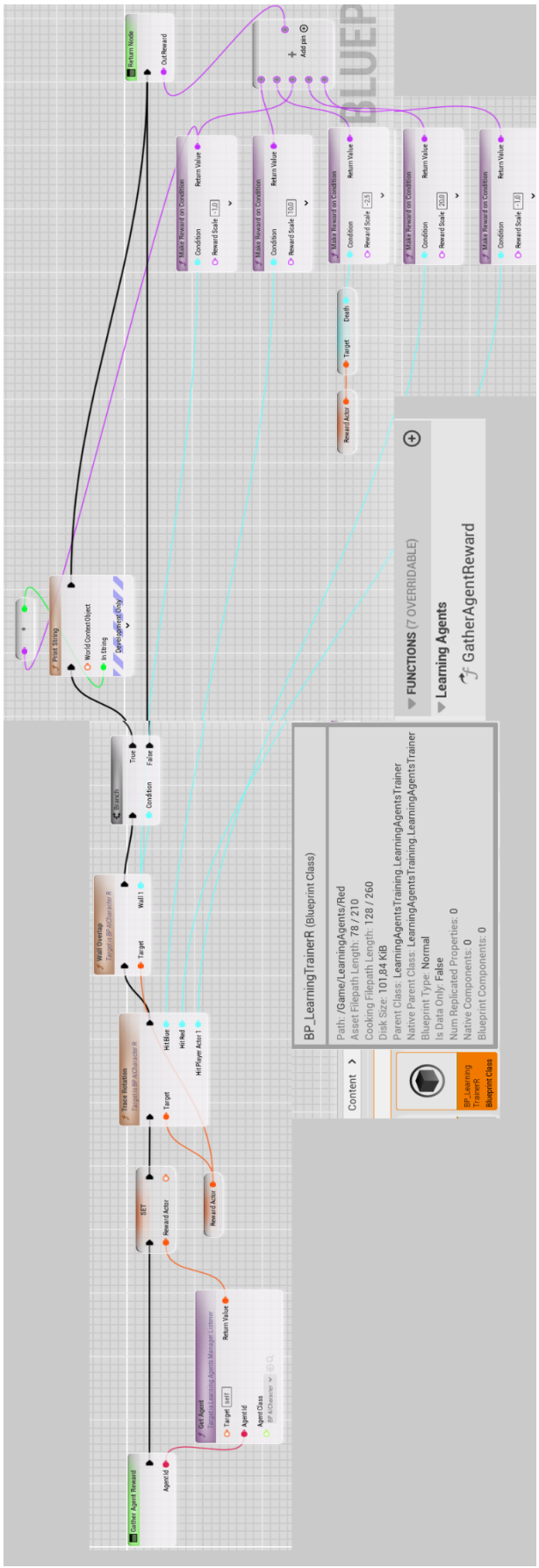

Blueprint Agent Interactor, функція Gather Agent Observation червоної команди



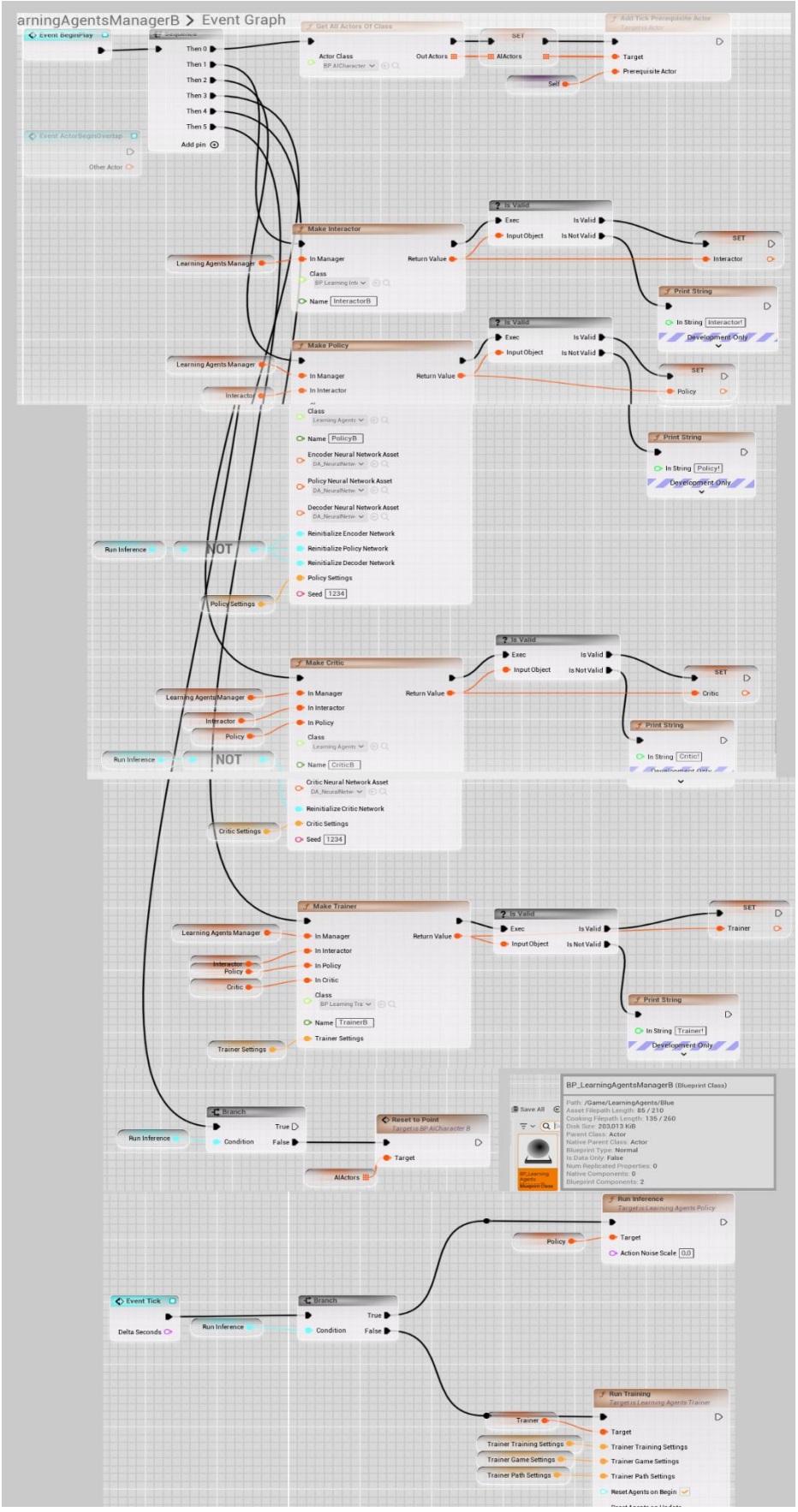

Blueprint Agent Interactor, функція Specify Agent Action червоної команди



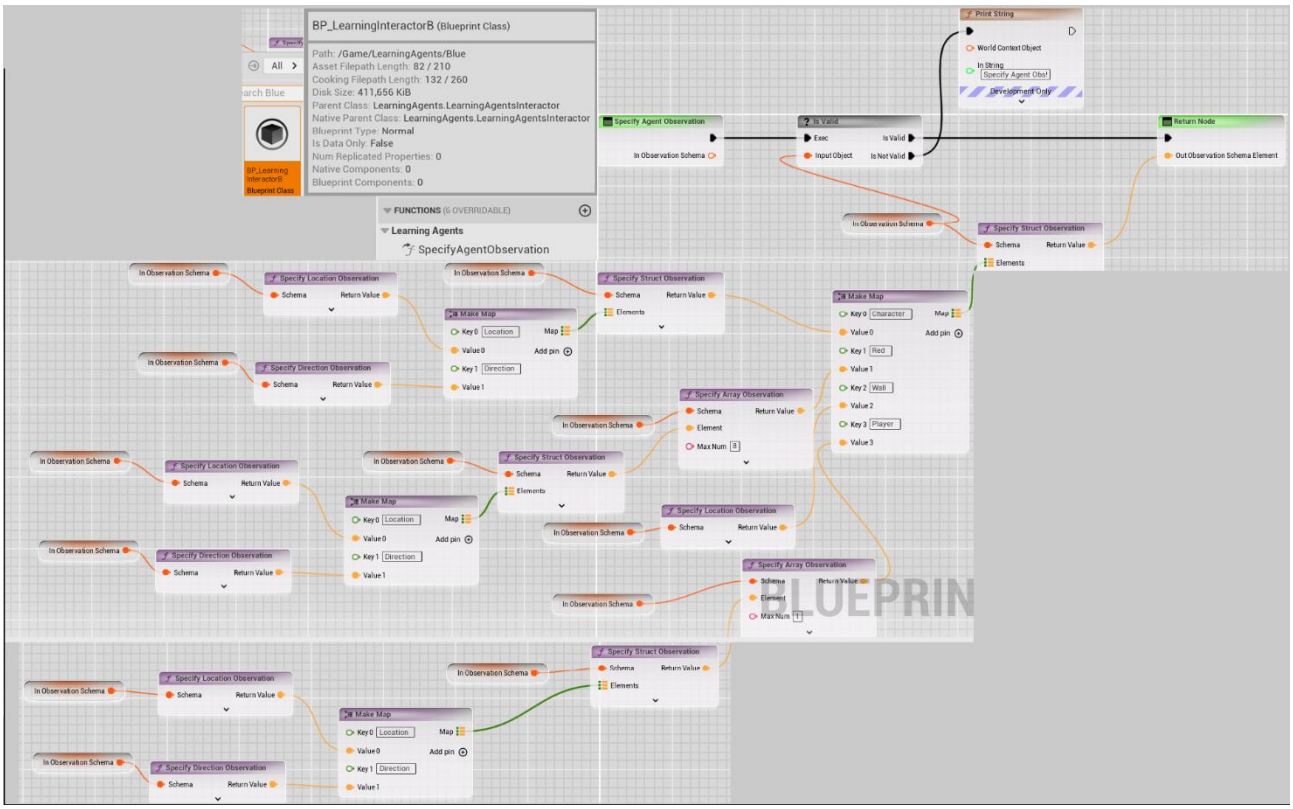

Blueprint Agent Interactor, функція Perform Agent Action червоної команди



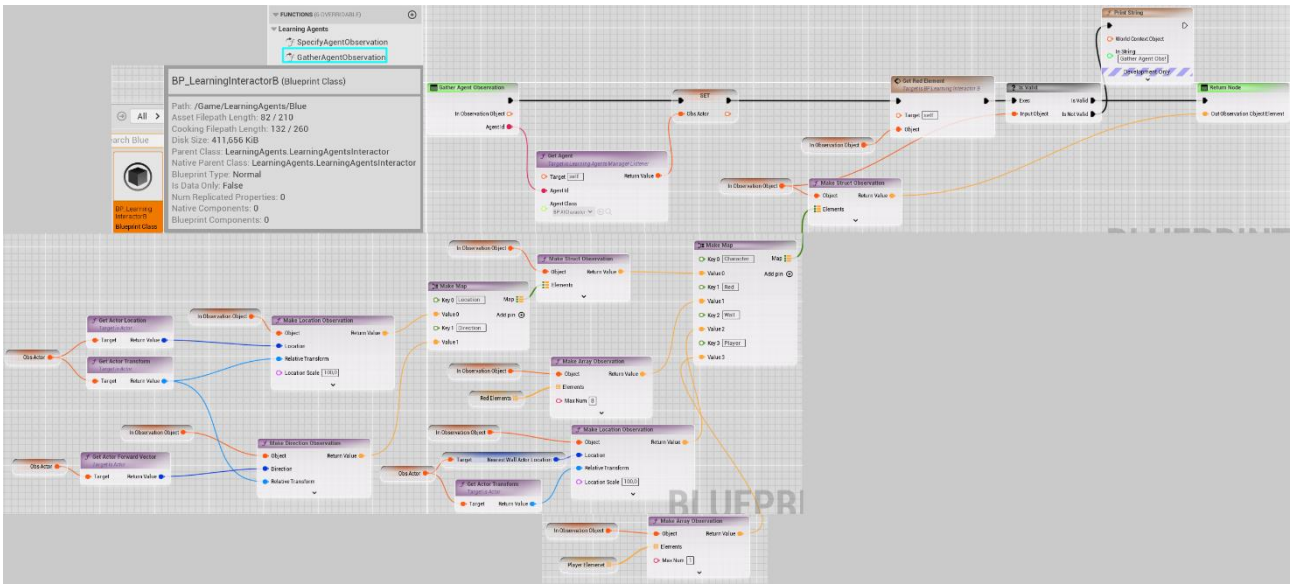

Blueprint Agent Trainer червоної команди



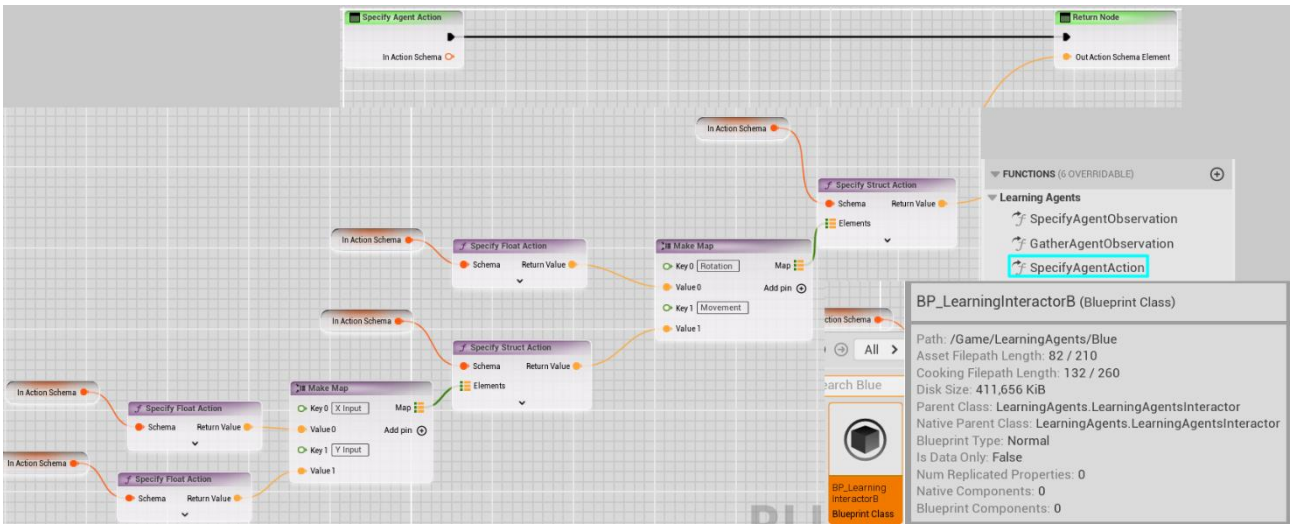

Blueprint Agent Manager синьої команди



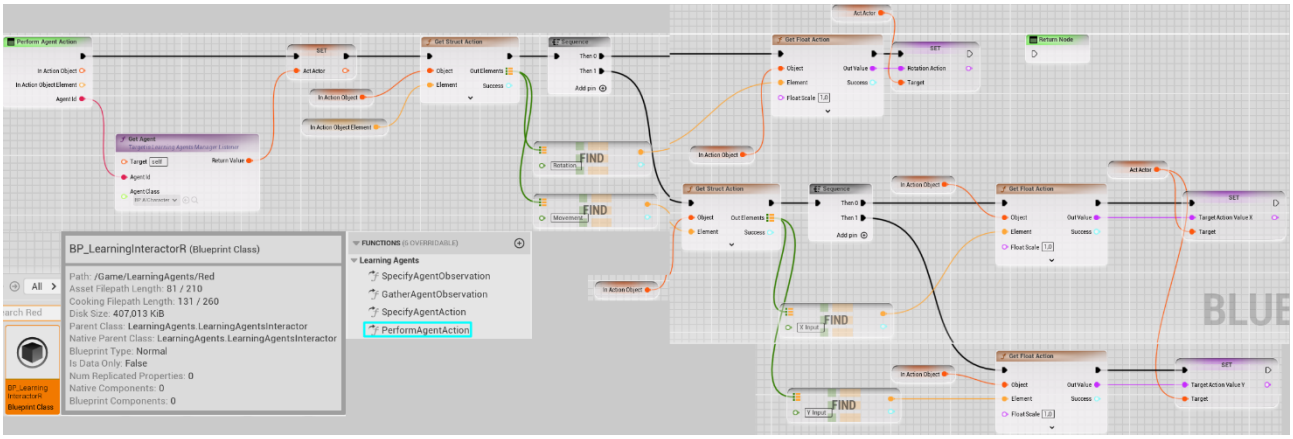

Blueprint Agent Interactor, функція Specify Agent Observation синьої команди



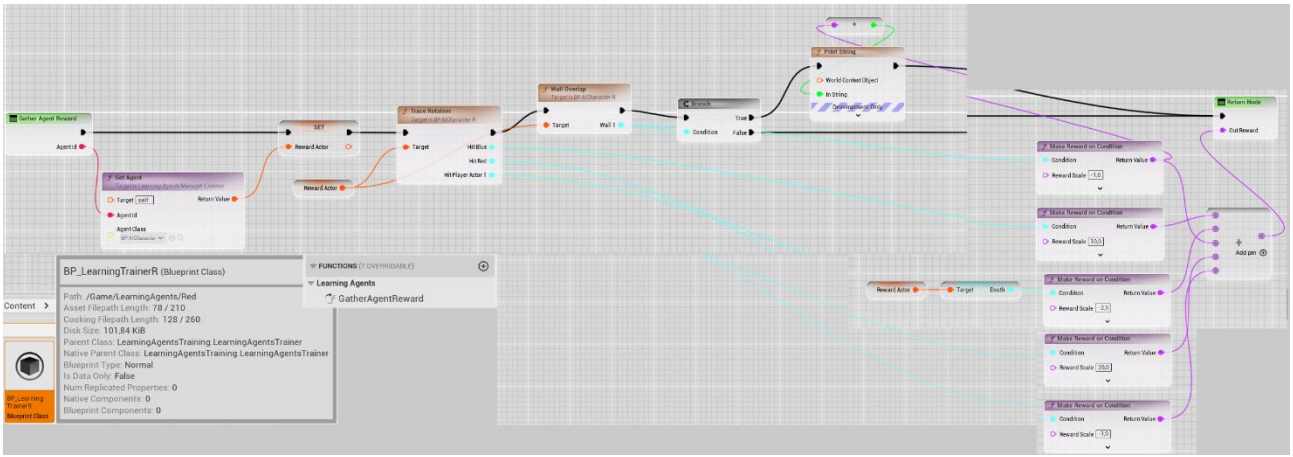

Blueprint Agent Interactor, функція Gather Agent Observation синьої команди




Blueprint Agent Interactor, функція Specify Agent Action синьої команди




Blueprint Agent Interactor, функція Perform Agent Action синьої команди




Blueprint Agent Trainer синьої команди

Додаток В

Апробація отриманих результатів

Західноукраїнський національний університет

Факультет комп'ютерних інформаційних технологій

Кафедра інформаційно-обчислювальних систем і управління


ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

Студентської науково-практичної конференції

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРИКЛАДНИХ ДОСЛІДЖЕННЯХ

(IITAR-2025)

27-29 травня 2025 року

Тернопіль

2025

***Васенко Світозар***
*студент групи КНШІ-41*
*svitozar0101@gmail.com*
***Павло Биковий***
*к.т.н., доцент*
*pb@wunu.edu.ua*
*Західноукраїнський національний університет*
*Тернопіль, Україна*

# ВИКОРИСТАННЯ ПІДХОДУ BEHAVIOR TREE ДЛЯ СТВОРЕННЯ АДАПТИВНИХ НЕІГРОВИХ ПЕРСОНАЖІВ У НАВЧАЛЬНИХ ВІДЕОІГРАХ

Сучасні інформаційні технології відіграють значну роль у розвитку освітніх методик, зокрема через використання відеоігор як інструменту навчання. Навчальні відеоігри сприяють розвитку стратегічного мислення, навичок вирішення проблем і командної роботи, а інтеграція штучного інтелекту у такі ігри підвищує їхню ефективність завдяки створенню адаптивних NPC (Неігрових персонажів) [1]. Одним із поширених підходів до створення такої поведінки є використання Behavior Tree (Дерева поведінки), яке дозволяє розробляти гнучку та передбачувану логіку дій для NPC.

Дерева поведінки є потужним інструментом для створення поведінки NPC, дозволяючи моделювати різноманітні дії залежно від змінних стану, таких як відстань до гравця, рівень здоров'я чи наявність укриття [2]. У навчальних відеоіграх NPC можуть виступати як інтелектуальні противники, які реагують на дії гравця, створюючи динамічне та захоплююче середовище. Наприклад, NPC може обирати між атакою, схованкою, наближенням до гравця або відступом, що змушує гравця адаптувати свою стратегію та розвивати навички планування й аналізу. Такий підхід може бути реалізований на платформах розробки ігор, таких як Unreal Engine, де Behavior Tree інтегрується з системами управління ШІ, зокрема через AI Controller (Контролер штучного інтелекту) і Blackboard (Чорна дошка) [3].

У контексті навчальних ігор Behavior Tree дозволяє створювати передбачувану поведінку NPC, що є важливим для забезпечення зрозумілого ігрового досвіду для учнів. Наприклад, NPC може бути налаштований таким чином, щоб атакувати, коли гравець перебуває на близькій відстані, або ховатися, якщо його здоров'я падає нижче певного рівня. Для вибору укриття можна використовувати порівняння відстаней до різних точок, що робить поведінку NPC більш реалістичною та сприяє розвитку стратегічного мислення учнів [4]. Такі ігри можуть бути застосовані в освітніх закладах для учнів різного

віку, а також у корпоративному навчанні для тренування командної роботи та прийняття рішень у стресових умовах [1].

Порівняно з іншими методами створення адаптивної поведінки, такими як Q-learning, Behavior Tree має низку переваг. Q-learning, засноване на навчанні з підкріпленням, дозволяє NPC навчатися оптимальних дій через систему винагород, але потребує значного часу на тренування та може бути складним у реалізації [5]. Натомість Behavior Tree забезпечує швидку розробку та передбачуваність, що є критично важливим для навчальних ігор, де поведінка NPC має бути контрольованою та відповідати освітнім цілям. Водночас Behavior Tree поступається за адаптивністю, оскільки не дозволяє NPC навчатися з досвіду, як це можливо з методами машинного навчання. Для наочності порівняння Behavior Tree та Q-learning наведено в таблиці 1.

Таблиця 1 - Порівняння Behavior Tree і Q-learning для створення адаптивних NPC у навчальних відеоіграх

| Критерій | Behavior Tree | Q-learning |
|---|---|---|
| Швидкість розробки | Висока: швидке налаштування | Низька: потребує тренування |
| Передбачуваність | Висока: чітка логіка дій | Низька: залежить від тренування |
| Адаптивність | Низька: немає навчання | Висока: NPC вчиться з досвіду |
| Складність реалізації | Низька: просте налаштування | Висока: потребує налаштування |
| Застосування в освіті | Ефективне для передбачуваності | Обмежене через складність |

Одним із ключових аспектів використання Behavior Tree у навчальних відеоіграх є можливість створення сценаріїв, які стимулюють учнів до активного навчання. Наприклад, NPC, який ховається при низькому здоров’ї, може спонукати гравця шукати нові стратегії для подолання перешкод, що розвиває аналітичні навички. Інтеграція таких NPC у навчальні ігри сприяє підвищенню залученості учнів, адже динамічна поведінка персонажів робить ігровий процес більш захоплюючим і мотивуючим [4].

Використання Behavior Tree також пов’язане з певними викликами. Наприклад, створення складної поведінки може вимагати детального налаштування умов і дій, що збільшує час розробки. Крім того, у динамічних ігрових середовищах поведінка NPC може бути обмеженою через відсутність адаптивності, що характерна для методів машинного навчання. Для подолання цих обмежень у майбутньому можна розглядати гібридні підходи, які поєднують Behavior Tree із методами навчання з підкріпленням, такими як Q-learning, для створення NPC, які є водночас передбачуваними та здатними адаптуватися до

нових умов [5].

Таким чином, Behavior Tree є ефективним інструментом для створення адаптивних NPC у навчальних відеоіграх, демонструючи значний потенціал інтелектуальних інформаційних технологій в освіті. Цей підхід дозволяє розробляти ігри, які розвивають стратегічні, аналітичні та командні навички учнів, підвищуючи їхню мотивацію до навчання. У майбутньому інтеграція Behavior Tree з іншими методами ШІ може відкрити нові можливості для створення більш гнучких і адаптивних навчальних ігор, що сприятиме подальшому розвитку освітніх технологій.

## Список використаних джерел


1. Prensky, M. Digital Game-Based Learning / M. Prensky. – New York: McGraw-Hill, 2001. – 442 p.
2. Millington, I. Artificial Intelligence for Games / I. Millington, J. Funge. – 2nd ed. – CRC Press, 2009. – 896 p.
3. Unreal Engine Documentation. Unreal Engine 5.4: Blueprints Visual Scripting [Electronic resource] / Epic Games. – 2024. – Access mode: https://docs.unrealengine.com/5.4/en-US/blueprints-visual-scripting-in-unreal-engine/
4. Gee, J. P. What Video Games Have to Teach Us About Learning and Literacy / J. P. Gee. – New York: Palgrave Macmillan, 2003. – 256 p.
5. Sutton, R. S. Reinforcement Learning: An Introduction / R. S. Sutton, A. G. Barto. – 2nd ed. – MIT Press, 2018. – 548 p.