МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

АНАЛІЗ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ У СТВОРЕННІ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІГРОВОГО ЗАСТОСУНКУ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Микита КУБИЦЬКИЙ**

«__» _____ 2025 р.

Керівник роботи
канд. техн. наук,
доцент кафедри ІПЗ _____ **Гліб ГОРБАНЬ**

«__» _____ 2025 р.

Миколаїв – 2025

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Другий (магістерський)
Освітній ступінь	Магістр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну магістерську роботу здобувача вищої освіти

Кубицького Микити

1. Тема кваліфікаційної роботи Аналіз ефективності використання методів машинного навчання у створенні штучного інтелекту для ігрового застосунку затверджена наказом ректора ЧНУ ім. Петра Могили № 182 від «2» липня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Очікуваним результатом є декілька типів поведінки для противників у ігровому застосунку та аналіз їх ефективності.

4. Перелік питань, що підлягають розробці:

− аналіз предметної галузі;

- аналіз методів машинного навчання;
- виконати проєктування та моделювання поведінки штучного інтелекту противника;
- реалізувати поведінку в ігровому застосунку;
- протестувати поведінку противника в ігровому застосунку;
- провести аналіз розроблених типів поведінки.

5. Перелік графічних матеріалів:

- презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «02» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Аналіз ефективності використання методів машинного навчання у створенні штучного інтелекту для ігрового застосунку**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	**01.09.2025**	**02.09.2025**	**виконано**
2.	Огляд літератури за темою роботи	**03.09.2025**	**04.09.2025**	**виконано**
3.	Складання календарного плану КМР	**05.09.2025**	**05.09.2025**	**виконано**
4.	Аналіз предметної області	**08.09.2025**	**12.09.2025**	**виконано**
5.	Розробка проєктних рішень	**15.09.2025**	**19.09.2025**	**виконано**
6.	Моделювання та конструювання ПЗ	**22.09.2025**	**26.09.2025**	**виконано**
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	**29.09.2025**	**10.11.2025**	**виконано**
8.	Відгук керівника КМР	**11.11.2025**	**17.11.2025**	**виконано**
9.	Оформлення КМР та презентації	**18.11.2025**	**23.11.2025**	**виконано**
10.	Попередній захист	**24.11.2025**	**24.11.2025**	**виконано**
11.	Рецензування	**25.11.2025**	**06.12.2025**	**виконано**
12.	Завершення оформлення КМР та презентації	**07.12.2025**	**10.12.2025**	**виконано**
13.	Захист кваліфікаційної роботи	**22.12.2025**	**23.12.2025**	**виконано**

Здобувач _____ **Микита КУБИЦЬКИЙ**

«05» вересня 2025 р.

Керівник роботи

канд. техн. наук,

доцент кафедри ІПЗ _____ **Гліб ГОРБАНЬ**

«__» _____ 2025 р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи

Аналіз ефективності використання методів машинного навчання у створенні штучного інтелекту для ігрового застосунку

Здобувач 608м гр.: Кубицький Микита

Керівник: канд. техн. наук, доцент Горбань Гліб

Актуальність обраної теми полягає в тому, що на теперішній час ігрова індустрія у більшості жанрів розвиває ШІ (штучний інтелект), як для створення більш реалістичного середовища, де NPC (non-playable character) не буде виконувати роль декорації та буде займатись певними діями самостійно та не прописано, так і для створення більш розумного противника з не прописаними діями та реакцією на поточне становище, так і розумних NPC, що будуть слугувати помічниками гравця або досить розумним співрозмовником. Є багато аспектів де ШІ використовується чи планується використовуватись і це не тільки для створення NPC так і для генерації світу, подій, завдань, середовища. Доречність описаної теми у тому, щоб перевірити чи достатньо та обґрунтовано витрачати стільки сил, ресурсів та часу для створення агентів штучного інтелекту за допомогою методів машинного навчання, замість створення звичайного та прописаного інтелекту.

Об’єктом кваліфікаційної роботи є процес створення агентів штучного інтелекту в ігровому застосунку.

Предметом кваліфікаційної роботи є методи машинного навчання для формування агентів штучного інтелекту в ігровому застосунку.

Метою кваліфікаційної роботи є дослідження ефективності використання методів машинного навчання до створення агентів штучного інтелекту в ігрових застосунках, для визначення потрібності отриманих витрат у часі та ресурсах при створенні штучного інтелекту.

Для досягнення мети треба виконати наступні завдання:

- виконати аналіз методів машинного навчання;
- виконати аналіз ігрового застосунку;

- виконати порівняльний аналіз методів машинного навчання;
- сформулювати вимоги до агентів ШІ, що розробляється;
- виконати проєктування та моделювання агентів ШІ;
- реалізувати агентів ШІ для ігрового застосунку;
- протестувати розроблений ШІ агентів для ігрового застосунку;
- виконати аналіз витрачених ресурсів та ефективність роботи агентів ШІ.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі описано тему, мету, предмет, об’єкт та завдання магістерської роботи, та наведено деяку додаткову інформацію щодо використаного матеріалу.

У першому розділі описано аналіз обраної сфери, використання методів машинного навчання в гейм індустрії, аналіз аналогів використання, актуальність використання методів ML (machine learning) та постановку задач на виконання.

Другий розділ присвячено процесу моделювання агентів штучного інтелекту із описом використаного інструментарію, існуючих методів для машинного навчання та методів для аналізу і навчання агентів, оформлення специфікації вимог.

У третьому розділі проєктування системи поведінки противника в ігровому застосунку за допомогою методів машинного навчання де описано сценарії використання, UML (Unified Modeling Language) діаграми, mockup, опис технологій, компонентів, мов програмування.

У четвертому розділі описано розробку, навчання, тестування штучного інтелекту та аналіз ефективності використання його в середовищі.

У висновках описується аналіз отриманих результатів з дослідження та виконанні задачі.

Кваліфікаційна робота викладена на 80 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 30 найменувань та 5 додатків. Праця містить 5 таблиць та 47 рисунків.

Ключові слова: машинне навчання, аналіз даних, ігровий застосунок, штучний інтелект, C#, Unity2D.

ABSTRACT

to the qualifying master's thesis

Analysis of the Efficiency of Using Machine Learning Methods in the Development of Artificial Intelligence for a Game Application

Student of 608m group: Kubytskyi Mykyta

Supervisor: PhD, Associate Professor Horban Hlib

The relevance of the chosen topic lies in the fact that currently, the gaming industry in most genres is developing AI (artificial intelligence) both to create a more realistic environment where NPCs (non-playable characters) do not serve as mere decorations but perform certain actions independently and are not scripted, as well as to create more intelligent opponents with unscripted actions and reactions to the current situation, and intelligent NPCs that will serve as the player's assistants or sufficiently intelligent interlocutors. There are many aspects where AI is used or planned to be used, not only for creating NPCs, but also for generating worlds, events, tasks, and environments. The relevance of the described topic is to verify whether it is sufficient and reasonable to spend so much effort, resources, and time to create AI agents using machine learning methods, instead of creating conventional and scripted intelligence.

The object of the qualification work is the process of creating artificial intelligence agents in a game application.

The subject of the qualification work is machine learning methods for forming artificial intelligence agents in a game application.

The purpose of the qualification work is to study the effectiveness of using machine learning methods to create artificial intelligence agents in gaming applications, to determine the necessity of the time and resources spent on creating artificial intelligence.

To achieve the goal, the following tasks should be completed:

- analyze machine learning methods;
- analyze the game application;
- perform a comparative analysis of machine learning methods;
- formulate requirements for the AI agents being developed;
- design and model the AI agents;

- implement the AI agents for the game application;
- test the developed AI agents within the game application;
- analyze resource expenditures and the efficiency of the AI agents performance.

The thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction describes the topic, purpose, subject, object, and objectives of the master's thesis and provides some additional information about the material used.

The first chapter describes the analysis of the selected field, the use of machine learning methods in the gaming industry, the analysis of analogues of use, the relevance of using ML (machine learning) methods, and the setting of tasks to be performed.

The second chapter is devoted to the process of modelling artificial intelligence agents with a description of the tools used, existing methods for machine learning and methods for analysing and training agents, and the formulation of requirements specifications.

The third chapter describes the design of the enemy behaviour system in a game application using machine learning methods, including usage scenarios, UML (Unified Modelling Language) diagrams, mockups, and descriptions of technologies, components, and programming languages.

The fourth section describes the development, training, testing of artificial intelligence and analysis of its effectiveness in the environment.

The conclusions describe the analysis of the results obtained from the research and the completion of the task.

The qualification work is presented on 80 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 30 titles and 5 appendices. The work contains 5 tables and 47 figures.

Keywords: machine learning, data analysis, gaming application, artificial intelligence, C#, Unity2D.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ОС	–	операційна система
ПК	–	персональний комп’ютер
ПЗ	–	програмне забезпечення
ШІ	–	штучний інтелект

AC	–	armor class
AI	–	artificial intelligence
API	–	Application Programming Interface
BMU	–	best matching unit
CS	–	character sheet
CSS	–	Cascading Style Sheets
ES	–	embedded systems
HP	–	hit points
HTML	–	HyperText Markup Language
IDE	–	interactive development environment
LINQ	–	Language Integrated Query
LVL	–	level
ML	–	machine learning
NPC	–	non-playable character
ONNX	–	open neural network exchange
PC	–	personal computer
PS	–	play station
TV	–	television
UI	–	user interface
UML	–	Unified Modeling Language
VR	–	virtual reality

ВСТУП

Актуальність обраної теми полягає в тому, що на теперішній час ігрова індустрія у більшості жанрів розвиває ШІ (штучний інтелект), як для створення більш реалістичного середовища, де NPC (non-playable character) не буде виконувати роль декорації, буде займатись певними діями самостійно та не прописано, так і для створення більш розумного противника з не прописаними діями та реакцією на поточне становище, так і розумних NPC, що будуть слугувати помічниками гравця або досить розумним співрозмовником. Є багато аспектів, де ШІ використовується чи планується використовуватись, і це не тільки для створення NPC, так і для генерації світу, подій, завдань, середовища.

Саме дослідження буде у тому, щоб перевірити чи достатньо та обґрунтовано витрачати стільки сил, ресурсів та часу для створення агентів ШІ за допомогою методів машинного навчання, замість створення звичайного та прописаного інтелекту.

Для виконання цього дослідження та отримання потрібних даних для аналізу, буде використано створений проєкт з КРБ, це ігровий застосунок у жанрі RogueLike з картковою системою бою та системою PathFinder 2e, де в проєкті буде створено агентів ШІ поведінки противників. Для збору даних буде використано декілька методів створення, навчання штучного інтелекту, що підходять для створення поведінки противника у середовищі.

Об’єктом кваліфікаційної роботи є процес створення агентів штучного інтелекту в ігровому застосунку.

Предметом кваліфікаційної роботи є методи машинного навчання для формування агентів штучного інтелекту в ігровому застосунку.

Метою кваліфікаційної роботи є дослідження ефективності використання методів машинного навчання до створення агентів штучного інтелекту в ігрових застосунках, для визначення потрібності отриманих витрат у часі та ресурсах при створенні штучного інтелекту.

Для досягнення мети треба виконати наступні завдання:

- виконати аналіз методів машинного навчання;
- виконати аналіз ігрового застосунку;
- виконати порівняльний аналіз методів машинного навчання;
- сформулювати вимоги до агентів ШІ, що розробляється;
- виконати проєктування та моделювання агентів ШІ;
- реалізувати агентів ШІ для ігрового застосунку;
- протестувати розроблений ШІ агентів для ігрового застосунку;
- виконати аналіз витрачених ресурсів та ефективність роботи агентів ШІ.

Апробація результатів КМР відбулась під час XXVIII Всеукраїнської науково-практичної конференції «Могилянські читання – 2025», Миколаїв, 12 листопада, 2025 р. (Додаток Д).

1 АНАЛІЗ СУЧАСНОГО ВИКОРИСТАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ В ЗАСТОСУВАННІ ДО ІГРОВИХ ЗАСТОСУНКІВ

1.1 Аналіз предметної області

Методи машинного навчання, ML(machine learning) – це методи, що надають можливість створити та навчати штучний інтелект за певним типом надання інформації [1].

Є такі методи машинного навчання:

- навчання з учителем;
- навчання без учителя;
- навчання з підкріпленням.

Окрім цих трьох також існує метод комбінацій, це по суті комбінація наведених методів вище. Кожен з методів представляє схему навчання штучного інтелекту, наприклад підхід надання інформації. Розберемо суть цих методів та їхню роботу в навчанні штучного інтелекту [2].

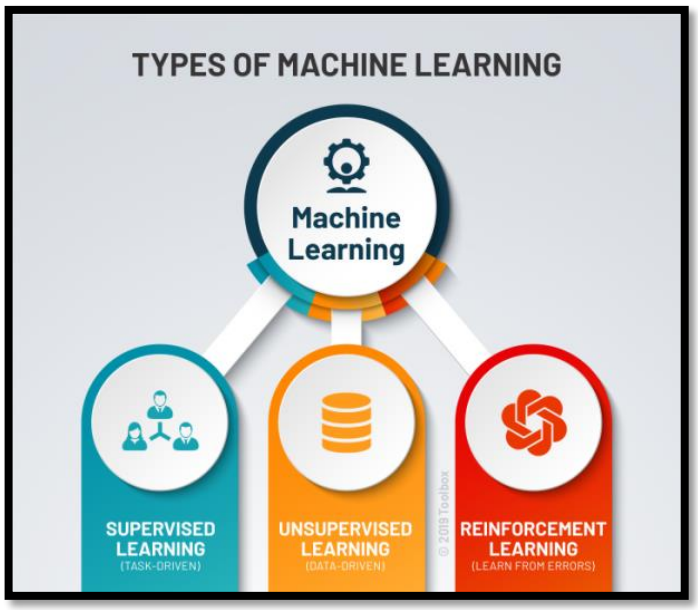

Рисунок 1.1 – Машинне навчання

Метод навчання з учителем, supervised learning – це метод, що навчає штучний інтелект на прикладах з правильними відповідями. Завдання штучного

інтелекту є відтворювати залежність між даними (X) та відповідями (Y). Наприклад, у вигляді даних подається фото предмета і є відповіді «кіт» чи «пес». Методами, за якими працює таке навчання є лінійна або логістична регресія, дерева рішень, випадковий ліс, метод опорних векторів, штучні нейронні мережі [3].

Метод, навчання без учителя, unsupervised learning – це метод, що навчає штучний інтелект шукати закономірності серед наданих даних (X). Робить це метод за допомогою групування та зменшення кількості даних. Наприклад, є певні дані сервера, штучний інтелект групує їх за закономірностями та таким чином отримує групи розробників, тестерів, бізнес-аналітиків. Методами, за якими працює таке навчання, є кластеризація, метод головних компонент, карти Кохонена [4].

Метод навчання з підкріпленням, reinforcement learning – це метод, що навчає штучний інтелект приймати певне рішення за допомогою нагород та штрафів. Завданням цього штучного інтелекту є навчитись приймати рішення за найбільшою нагородою в довгий строк перспективи. Наприклад, правильно вирішує математичне рівняння, отримує нагороду і навпаки. Методами, за якими працює таке навчання, є звичайне та глибоке Q-навчання, метод градієнта політики (Policy gradient) [5].

Кожен з цих методів підходить по своєму до певних задач, тому не кожен метод машинного навчання може підійти під одне й те саме завдання. Тому слід обирати за певною ціллю або тестувати та комбінувати.

Саме машинне навчання не існує просто так саме по собі, бо окрім звичних методів машинного навчання існують також методики глибинного навчання агентів штучного інтелекту або створення агентів штучного інтелекту за допомогою нейронних мереж.

За цими методиками є можливість створити агента штучного інтелекту, що є більш складний, розумний, багатофункціональний, схожий на людину за можливостями мислення, але не розумом або пам’яттю, бо в машини це набагато розвинуте. Однак ці методики не буде розглянуто в цій роботі.

1.2 Використання методів машинного навчання в сучасності

Штучний інтелект використовують для різних задач, найбільш відома задача у світі це пошук інформації за запитом. І за такими задачами є дуже відомі системи штучного інтелекту, що не раз вже переростали та навчались, формуючи нові версії самої себе, наприклад ChatGPT від OpenAI або Google AI Studio від Google [6].

Рисунок 1.2 – Система ChatGPT від OpenAI

Окрім відомого застосування, пошуку інформації, також є й інші види застосування, такі як:

- сфера медицини – діагностика захворювань, персоналізована медицина, роботизовані хірурги;
- сфера фінансів – торгівля на біржі, аналіз фінансової спроможності, запобігання шахрайству;
- сфера транспорту – автономні автомобілі, оптимізація маршрутів, інтелектуальні транспортні системи;
- сфера освіти – адаптивне навчання, віртуальні репетитори, аналіз успішності;
- сфера промисловості – роботи-маніпулятори, прогнозування технічного обслуговування, оптимізація ланцюга постачань;

– сфера науки – обробка великих обсягів даних, прогнозування, математичні та інженерні симуляції;

– сфера медіа – генерація контенту, відеоігри, deepfake, рекомендації, цифрові персонажі;

– сфера безпеки – відеоспостереження, виявлення небезпечної поведінки, виявлення атак у кіберпросторі, махінацій, фішингу, розпізнавання облич, відбитків пальців, прогнозування злочинності;

– сфера військова – дрони з автонаведенням або комп’ютерним зором із виявленням об’єктів, розумна зброя з донаведенням, симулятор боїв, кіберзахист.

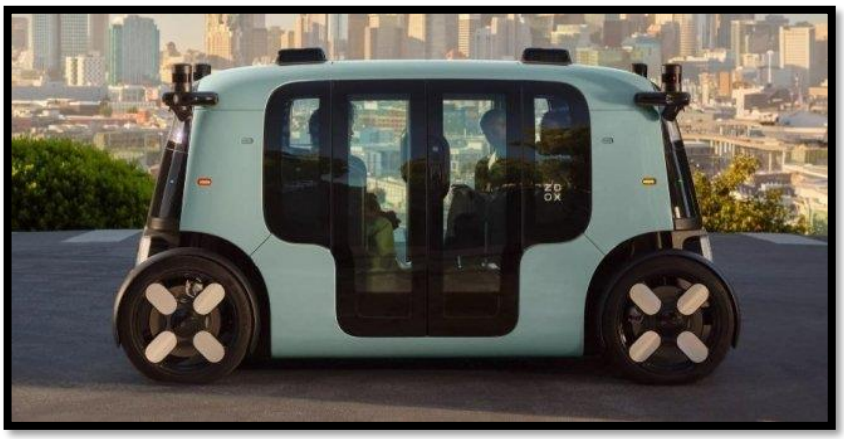

Рисунок 1.3 – Система автономного таксі

Дивлячись на застосування у різних сферах життя людини штучного інтелекту його основні задачі залишаються звичними, такі як аналіз, розпізнавання, обробка даних, вибір найкращого варіанту [7].

Найбільш цікава сфера застосування штучного інтелекту для цієї роботи є саме сфера медіа, а якщо ще точніше, то розробка ігрових застосунків. Штучний інтелект використовується у різних аспектах ігрового застосунку, такі як генерація рівнів, аналіз подій, керування поведінкою NPC, AI-тестування, навігація та знаходження шляхів (PathFinding), створення анімацій, створення інтерактивностей в середовищі гри, створення квестів, оптимізація та баланс гри [8].

Для реалізації цих функцій обирається певний метод машинного навчання враховуючи його функціональну сторону, отримувану інформацію або комбінуючи методи. Комбінувати буде складно та затратно по часу, тому розберемо основні методи машинного навчання, а саме їх застосування у розробці ігрового застосунку.

Метод машинного навчання supervised learning застосовується у розпізнаванні поведінки користувача, передбаченні або прогнозуванні дій гравця, створення рекомендацій залежно від ситуації, розпізнавання голосу та зображення.

Метод машинного навчання unsupervised learning застосовується у кластеризації гравців розподіляючи їх на групи, аналіз поведінки гравців та створення поведінкових патернів, автоматична генерація контенту, збалансування контенту.

Метод машинного навчання reinforcement learning застосовується у створенні AI ворогів та NPC з реалістичною поведінкою, пошук оптимального маршруту або PathFinding, тестування застосунку, адаптивна складність.

1.3 Аналіз аналогів використання методів ML в ігровому застосунку

Для аналізу ефективності обрано саме створення поведінки штучного інтелекту противника, та щоб зрозуміти які є аналоги за описаними методами проведено аналіз аналогів.

За цим аналізом буде продемонстровано ігрові застосунки, де було використано один з трьох методів машинного навчання для створення поведінки штучного інтелекту противника.

До кожного застосунку описано назву застосунку, архітектура або архітектури застосунку, система, тобто агент або агенти в застосунку, метод машинного навчання за котрим було створено систему, опис системи, її переваги та недоліки роботи в ігровому застосунку [9].

Рисунок 1.4 – Forza Motorsport

Ігровий застосунок: Forza Motorsport.

Архітектура: Xbox/PC (personal computer).

Система: Drivatar.

Метод машинного навчання: Supervised learning.

Опис: Ця система надає можливість постійно навчати штучний інтелект поведінки суперників за допомогою перегляду та аналізу стилю їзди гравців та після використовувати отримувані навички вже в дії, у грі в офлайн або онлайн режимі.

Переваги та недоліки: Перевагою є те, що агент має змогу постійно навчатись та мімікрувати під стиль гри гравців. По суті копіює маршрути та логіку їзди гравців на обраних рівнях гри.

Недоліком є те, що штучний інтелект не створює власний патерн поведінки, а копіює її та якщо в грі буде малий онлайн гравців, штучний інтелект може бути погано навченим, через що не бути на рівні гравців.

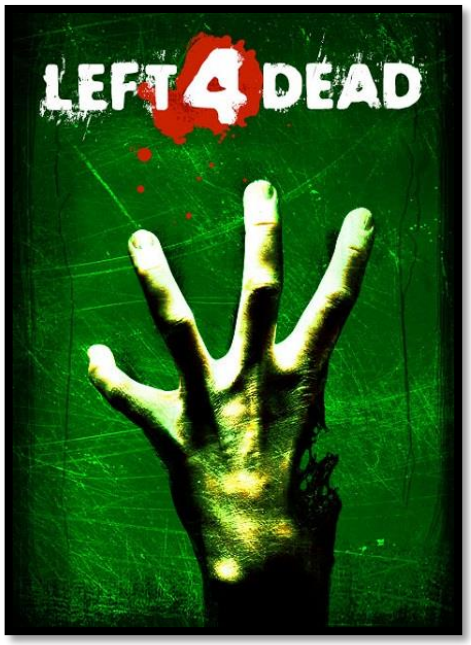

Рисунок 1.5 – Left 4 Dead

Ігровий застосунок: Left 4 Dead.

Архітектура: Xbox/PC.

Система: AI Director.

Метод машинного навчання: Unsupervised learning.

Опис: Ця система підлаштовує складність рівня підчас гри або підчас переходу на іншу складність, рівень, також змінює вірогідне розташування ворогів та ресурсів для гравців залежно від патерну поведінки гравців.

Переваги та недоліки: Перевагою є те, що агент сам знаходить патерни гравців та реагує відповідно цим патернам, наприклад агресивним гравцям більше супротивників, а пасивним навпаки. Відповідно патерни гравців неодноразово змінюються підчас гри через, що складно передбачити поведінку агенту.

Недоліком є те, що агент може бути іноді дивним та непередбачуваним, навіть для розробника, у виконанні своїх задач, а це може іноді пошкодити ігровий процес гравця.

Рисунок 1.6 – Dota 2

Ігровий застосунок: Dota 2.

Архітектура: PC.

Система: OpenAI Five.

Метод машинного навчання: Reinforcement learning.

Опис: Ця система навчає агента грати, розробляючи власні фішки, способи перемоги реагуючи на середовище, також агент грав багато симуляцій проти професійних гравців та переглядав матчі професійних гравців, як поза турнірами так і на турнірах.

Переваги та недоліки: Перевагою є те, що штучний інтелект сам знаходить дієві способи перемоги та знає більшість речей, що стосуються гри, також система здатна грати мільйони симуляцій аби розробити певні дії проти патернів гравців.

Недоліком є те, що штучний інтелект не готовий до незвичної поведінки та через це може вести себе дивно або виконувати дії, що не мав би виконувати у поточній ситуації.

Рисунок 1.7 – FIFA 2025

Ігровий застосунок: FIFA 2025.

Архітектура: PC/Xbox/PS (play station).

Система: Imitation AI.

Метод машинного навчання: Supervised learning.

Опис: Ця система навчається на спортивних матчах, телеметрії гравців, розмічених відео із анотаціями. Система намагається імітувати гравців, використовуючи тактики, стилі та навички гри гравців.

Переваги та недоліки: Перевагою є те, що штучний інтелект діє за новими або зарекомендованими тактиками в спорті та таким способом матч може виявитись складним для гравця, також оновлення метрик спортивних гравців є певною перевагою в системі.

Недоліком є те, що штучний інтелект може бути передбачуваний та не готовий до нестандартних тактик користувачів, що більш знайомі з комп’ютерними іграми, а не зі спортом та їх правилами або традиціями.

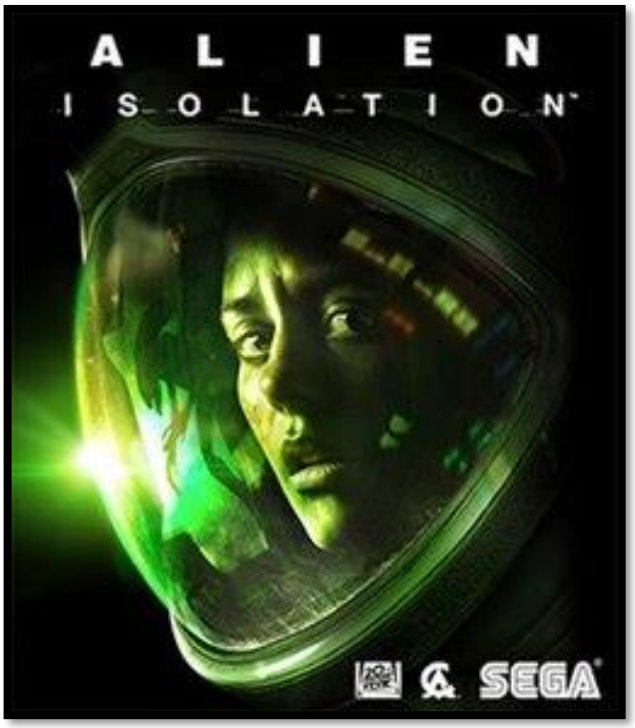

Рисунок 1.8 – Alien Isolation

Ігровий застосунок: Alien Isolation.

Архітектура: PC/Xbox/PS.

Система: Director AI та Reactive AI.

Метод машинного навчання: Supervised learning та reinforcement learning.

Опис: Ця система є парною, диспетчер слідкує за гравцем та надає не точні підказки щодо гравця або точні залежачи від поведінки гравця, а директор отримує дані з диспетчера та навчається на своєму досвіді, як протистояти гравцю.

Переваги та недоліки: Перевагою є те, що штучний інтелект діє на противагу гравцю, він отримує дані вже підчас ігрової сесії та вчиться майже у реальному часі та таким чином систему важко прогнозувати гравцем, система використовує різні поведінкові фактори гравця та використовує їх проти гравця. Система легка для процесору та не застосовує нейронні мережі.

Недоліком є незбереження досвіду після вимкнення застосунку, імітує навчання у реальному часі за допомогою правил, ШІ директора знає лише про ігрові події, а не глобальні.

1.4 Актуальність використання методів ML у створенні поведінки NPC в ігровому застосунку

Актуальність використання методів машинного навчання у створенні поведінки неігрових персонажів в ігровому застосунку полягає в тому, що ця технологія дозволяє полегшити роботу в створені логіки для певних систем або елементів у ігровому застосунку. Таким чином дає можливість збалансувати розумність неігрових персонажів.

Методи машинного навчання роблять неігрових персонажів більш живими та чесними у відношенні до гравців, бо раніше поведінка неігрового персонажів була високо передбачувана та механічна.

До використання методів машинного навчання, неігрові персонажі були занадто розумні або занадто тупі і на це впливало скільки та яку інформацію розробник надає неігровому персонажу. Також для взаємодії з середовищем, у неігрових персонажів були інструкції зі станами прописані шляхи від точки А до точки В в ігровому середовищі [10].

Рисунок 1.9 – Counter Strike 1.6

Наприклад, в ігровому застосунку Counter Strike версії 1.6, неігрові персонажі, так звані боти, дуже часто бачили своїх противників через стіни та знали

де противник знаходиться на мапі, однак вони були збалансованими складністю ботів, бо кожна складність означала якість та кількість інформації передано ботам [11].

Боти мали певні стани та сценарії (if-else). Через сценарій боти переходили до певного стану, а замість знаходження свого шляху до певного місця на карті, вони рухались за прописаними точками на мапі.

Із появою методів машинного навчання поведінка неігрових персонажів перестали бути скриптовими та стали агентами. Їх стало складно передбачити, бо тепер вони могли прийняти рішення на своєму досвіді, мали пам’ять про минулі події, відслідковувати стратегії та формувати свої.

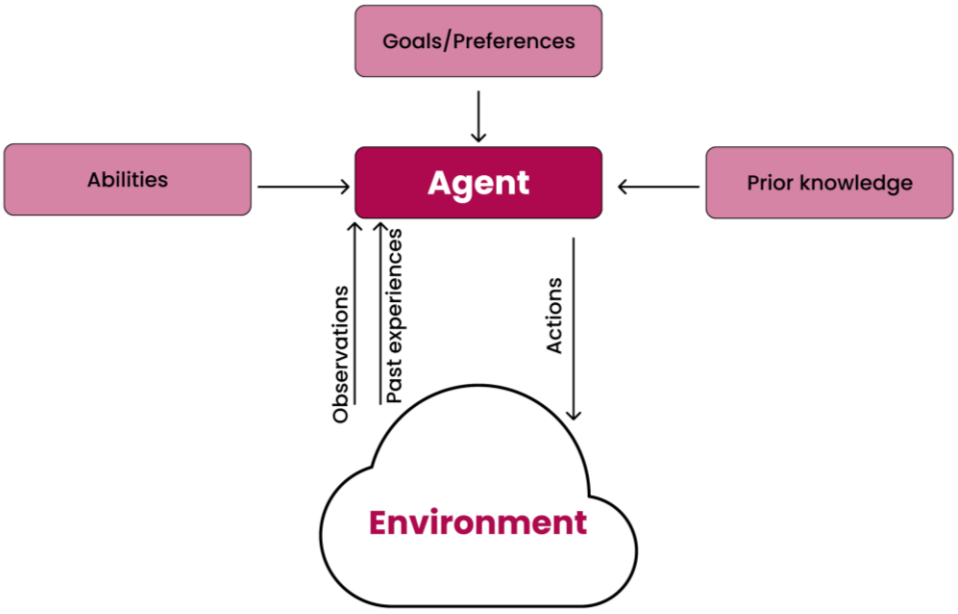

Рисунок 1.10 – Робота агента

На зараз створення штучного інтелекту поведінки є два типи, слабкий агент та сильний агент. Різниця між ними полягає у кількості пам’яті, що ті використовують, та близькості до інтелекту людини. Тобто слабких агентів створюють в основному для виконання маленьких однотипних задач, а сильні агенти можуть виконувати неоднотипні, складні та легкі задачі [12].

Для створення певних систем є практика створювати декілька слабких агентів для виконання своїх задач та взаємодії з іншими агентами, бо створення

сильного агента займає багато часу та місця для пам’яті, однак не виключають і використання одного сильного агенту разом із декількома слабкими агентами.

Прикладом використання декількох слабких агентів є ігровий застосунок Alien Isolation, про котрий з аналогів було наведено до цього, бо він має два агента котрі взаємодіють між собою та керують неігровим персонажем, чужим або біоморфом, де один агент на базі отриманих даних та своїй пам’яті намагається віднайти гравця, а другий агент підказує йому надаючи йому інформацію та підлаштовує стратегію для чужого аналізуючи стратегію гравця [13].

1.5 Постановка задач

Розібравшись із темою, предметом, об’єктом, метою залишається провести дослідження, проаналізувавши ефективність використання методів машинного навчання у ігровому застосунку. Для проведення аналізу буде використано ігровий застосунок створений окремо раніше.

Для проведення аналізу треба створити три агента, по методу машинного навчання на кожного, в ігровому застосунку та зібрати дані їх роботи окремо для кожного.

Ланкою, де буде проведено аналіз у ігровому застосунку, обрано саме поведінку NPC. Створення поведінки NPC за допомогою методів машинного аналізу обрано тому, що кожен метод можна застосувати у створенні поведінки NPC.

Роль агентів, що будуть створені в ігровому застосунку, це перемогти гравця. Їх основними функціями буде аналізувати стан NPC та гравця, обирати найкращий варіант підчас свого ходу аби перемогти гравця.

Для проведення дослідження існують такі задачі:

- ознайомитись із методами та способами створення та навчання агентів у ігровому середовищі;
- змоделювати та спроєктувати агентів, їхні функції, отримувані дані;
- підготувати ігрове середовище до реалізації агентів в ньому;

- створити систему перемикання між агентами;
- створити трьох агентів, кожен за окремим методом;
- навчити агентів за даними;
- протестувати систему;
- зібрати дані для аналізу кожного агенту;
- проаналізувати дані та зробити висновок.

Головними критеріями для дослідження є час навчання, результати матчів, складність реалізації, об’єм використаної пам’яті після навчання. За цими критеріями буде проведено аналіз та визначено ефективність кожного методу окремо, також загальний висновок про використання методів машинного аналізу в ігровому застосунку.

Висновки до розділу 1

За аналізом методів машинного навчання зроблено висновки щодо самих методів, їх сфер використання, їх значення та основних задач котрі штучний інтелект виконує. Також розглянуто, задля яких цілей та задач використовують штучний інтелект, які є методи машинного навчання, яка є різниця між ними, які їх основні, базові, функції та як вони працюють. Описано, як методи машинного навчання навчають штучний інтелект, який паттерн дій надають як отримують інформацію та в якій кількості.

Для більшого аналізу обрано медіа сферу, а саме сферу створення ігрових застосунків та за цією сферою проведено аналіз аналогів застосування штучного інтелекту поведінки в різних застосунках, де були порівняння методів машинного навчання, недоліки та переваги використання.

У висновку, штучний інтелект можна використати в майже кожній частині ігрового застосунку, а щодо створення поведінки, то можна створити різну поведінку, як непередбачувану так і передбачувану, однак є й свої мінуси в цьому застосуванні, бо іноді штучний інтелект може бути непередбачуваний навіть для самого розробника.

2 МОДЕЛЮВАННЯ СИСТЕМИ ПОВЕДІНКИ ПРОТИВНИКА

2.1 Інструментарій

Для реалізації поставлених задач треба скласти інструментарій, котрий допоможе в реалізації задумів. В інструментарій входить ігровий рушій, бібліотеки використані в рушію, середовище розробки, додаткові елементи для інтеграції.

Як для ігрового рушію буде використано рушій Unity за моделлю 2D.

Рисунок 2.1 – Ігровий рушій

Ігровий рушій Unity доволі популярний серед інших рушіїв, враховуючи те, що він є доступний для більшості користувачів, компаній, підприємців. Його доступність та відкритість і дає таку популярність. Рушій має документацію та безліч матеріалу для навчання з рушієм. На базі цього рушію були створені такі, популярні у свої часи, ігрові застосунки як: Pathfinder: Wrath of the Righteous; Subnautica; Valheim; Cuphead [14].

Проведемо характеристику ігрового рушію Unity, які він має особливості.

Таблиця 2.1 – Характеристика рушія Unity

№	**Мова програмування**
№1	C#
№2	C++
№	**Платформа**
№1	iOS
№2	Android, Android TV (television)
№3	Tizen
№4	Windows, Universal Windows Platform, Windows Mixed Reality

Продовження таблиці 2.1

№5	Mac
№6	Linux
№7	WebGL
№8	PlayStation 4, PlayStation 5, PlayStation Vita, PlayStation VR (virtual reality)
№9	Xbox One, Xbox Series X, Xbox Series S
№10	3DS
№11	Oculus Rift
№12	Google Cardboard, Google ARCore
№13	Steam VR
№14	Gear VR
№15	Daydream
№16	Samsung Smart TV
№17	tvOS
№18	Nintendo Switch
№19	Facebook Gameroom
№20	Apple ARKit
№21	Vuforia
№22	Magic Leap
№	**Особливості**
№1	Рушій має інтерфейс котрий користувач може підлаштувати під себе, усі вікна налаштовані та їх можна зберегти для наступного користування, інтерфейс має в собі оглядач ресурсів проєкту, інспектор поточного об’єкту, вікно попереднього перегляду, оглядач сцени та оглядач ієрархії ресурсів.
№2	Рушій поділяє проєкт на сцени, сцени зберігають в собі набори ефектів, сценаріїв, налаштувань, елементів довкілля. У сцені дозволено масштабувати, редагувати, переміщувати, обертати будь-які елементи.

Кінець таблиці 2.1

№3	Рушій підтримує фізику тіл котра може бути налаштована за бажаннями користувача. Дає можливість при створенні проєкту обрати тип світу, 2D або 3D. Відповідно у двовимірному світі використовуються спрайти, а у тривимірному світі меші на які накладають текстури, матеріали та шейдери.
№4	Рушій підтримує стиснення текстур, міпмапінг, різні роздільності екрана, бамп-мапінг, мапінг відображень, паралакс-мапінг, затінення навколишнього світла у екранному просторі, динамічні тіні за картами тіней, рендеринг у текстуру, обробка зображення.
№5	Рендеринг відбувається за допомогою камери на сцені, в рушію сцену можна споглядати окремо від камери, а підчас самого рендерингу все відбувається від лиця камери.
№6	Графічний рушій використовує DirectX, OpenGL, OpenGL ES (embedded systems), власне API (Application Programming Interface) для Wii. Підтримує файли 3dmax, maya, Softimage, blender, modo, zbrush, cinema 4D, cheetah3D, adobe photoshop, adobe fireworks. Для шейдерів використовує ShaderLab, що підтримує GLSL та Cg. Є вбудована підтримка Nvidia PhysX.

Для реалізації методу машинного навчання з підкріпленням та інших можна використати бібліотеку для навчання під назвою ML-Agents. Ця бібліотека встановлюється онлайн за допомогою Package Manager встановлюючи декілька пакетів разом із встановленням Python. Також потребується клонувати репозиторій бібліотеки з github [15].

Ось такі пакети слід встановити для проєкту:

- com.unity.ml-agents;
- com.unity.ml-agents.extensions(Optional).

Ось такі бібліотеки слід встановити для Python:

- mlagents-envs;
- mlagents.

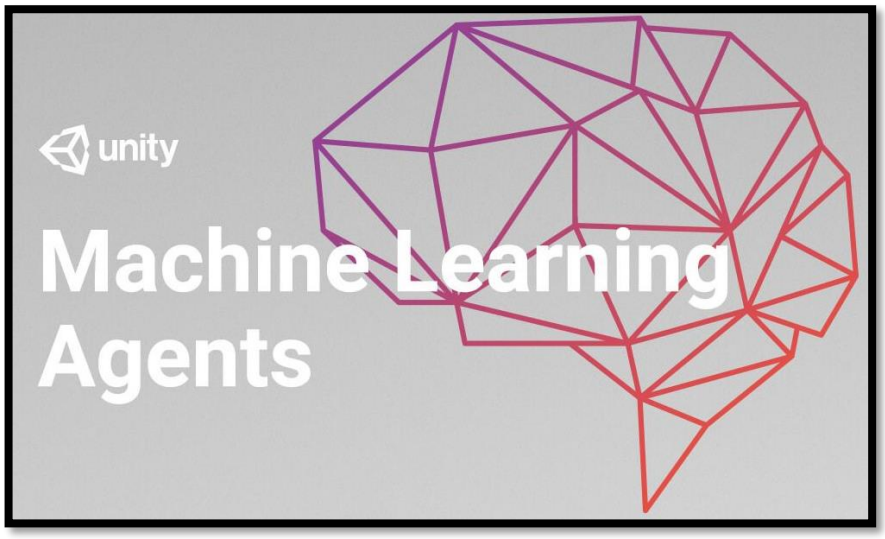

Рисунок 2.2 – Бібліотека ML-Agents

Ця бібліотека дозволяє використовувати ігри та симуляції як середовища для навчання ML агентів. Навчання може бути за методами навчання з підкріпленням, імітаційного навчання, нейроеволюції та інших методів. Ці агенти можуть виконувати різні завдання, як поведінка NPC, тестування, аналіз, прогнозування і так далі.

В самому рушію Unity, бібліотеки дозволяє перетворити будь-яку сцену в тренувальне, навчальне середовище, де можна навчати поведінку персонажів за допомогою методів навчання.

Для написання скриптів у рушію Unity слід обрати IDE (interactive development environment). Рушій підтримує декілька середовищ та має для них свої рекомендації щодо налаштувань, наприклад одне з них вказує тримати середовище оновлене до останньої версії. Рушій Unity пропонує такі середовища IDE:

- Visual Studio Code;
- Visual Studio;
- JetBrains Rider.

Visual Studio – це інтегроване середовище розробки від компанії Microsoft, що надає можливості у створені консольних застосунків, вебзастосунків, графічних застосунків [16]. Платформами, на яких розповсюджується середовище, є Windows 10 та пізні версії, Windows Server 2016 та пізні версії, MacOS.

Рисунок 2.3 – Visual Studio

Переваги середовища:

- ідеальне для великих корпоративних застосунків;
- глибокі інструменти для налагодження, управління проєктами та підтримку багатьох мов програмування;
- багата бібліотека розширень, які додають підтримку для інших мов та технологій, таких як C++, C#, Java, Python, PHP, Go, .NET і Unity;
- вбудований Git та інструменти для налагодження.

Недоліки середовища:

- професійні та корпоративні версії вимагають придбання ліцензії;
- складний інтерфейс та круту криву навчання, що робить його менш зручним для новачків у порівнянні з легшими редакторами;
- вимагає значних ресурсів, особливо при роботі з великими проєктами або численними розширеннями.

Visual Studio Code – це інтегроване середовище розробки схоже із Visual Studio, розроблене від компаній Microsoft та Electron Framework для платформ Windows, Linux, MacOS [17]. Середовище більше всього застосоване для розробки вебзастосунків.

Рисунок 2.4 – Visual Studio Code

Переваги середовища:

– доступний для всіх безкоштовно;
– кросплатформний;
– порівняно з іншими IDE, він запускається швидше та споживає менше ресурсів, хоча це може змінюватися залежно від встановлених розширень;
– має велику бібліотеку розширень, які дозволяють налаштувати його під будь-які потреби;
– вбудована підтримка Git та інструменти для налагодження.

Недоліки середовища:

– може працювати повільніше при відкритті дуже великих проєктів;
– порівняно з повноцінними IDE, як-от Visual Studio, VS Code може мати менш потужні інструменти для деяких завдань, наприклад, для налагодження або рефакторингу коду;
– деякі розширення можуть споживати багато ресурсів або призводити до нестабільної роботи програми;
– незважаючи на те, що він відносно легкий, при роботі з великими файлами або багатьма вкладками може споживати значну кількість оперативної пам’яті.

JetBrains Rider – це інтегроване середовище розробки для .NET, яке відоме своєю продуктивністю та широкими можливостями в аналізі коду та функціоналом ReSharper [18]. Розробником середовища є компанія JetBrains, яка поставляє

середовище як комерційний проєкт з можливостю використання безкоштовно, якщо ви навчаєтесь.

Рисунок 2.5 – JetBrains Rider

Переваги середовища:

– є кросплатформним, працює на Windows, macOS та Linux;

– підтримує .NET Framework, .NET Core, ASP.NET, Xamarin, Unity, JavaScript, TypeScript, HTML (HyperText Markup Language) і CSS (Cascading Style Sheets);

– є вбудована підтримка для управління базами даних, системами контролю версій, засобів для модульного тестування і профілювання продуктивності;

– часто відзначається вищою швидкістю роботи та стабільністю;

– ReSharper, який забезпечує глибокий аналіз коду, автоматичні виправлення, рефакторинг і запобігання помилкам.

Недоліки середовища:

– попри наявність безкоштовної версії для некомерційного використання, комерційні ліцензії можуть бути значно дорожчими;

– хоч Rider оптимізований для продуктивності, він може споживати значну кількість оперативної пам'яті та ресурсів процесора, особливо під час роботи з великими проєктами та інтенсивним аналізом коду;

− бібліотека плагінів для Rider менша. Це може обмежувати вибір для деяких вузькоспеціалізованих завдань.

2.2 Огляд існуючих методів

Огляд існуючих методів машинного навчання допоможе зрозуміти за якими правилами отримувані дані проходять аналіз, аналітику, прогнозування та які математичні формули використовуються для обчислення результату.

Регресія – це метод аналізу для визначення залежності однієї змінної X від іншої змінної Y або декількох змінних. Виражається залежність у числовому значенні. Алгоритми, що використовують цей метод, є лінійна регресія, поліноміальна регресія, Support Vector Regression, Random Forest Regression, Neural Network Regression [2, 3, 4, 5].

Класифікація – це метод віднесення змінної X або змінних до певної категорії або класу {X}. Алгоритми, що використовують цей метод, є логістична регресія, Decision Trees, Random Forest, Support Vector Machines, Neural Network, K-Nearest Neighbors [2, 3, 4, 5].

Кластеризація – це метод який групує змінні X, Y у кластери або класи схожі за ознаками без відомих міток класу або кластеру. Алгоритми, що використовують цей метод, є K-Means, Gaussian Mixture Models, Self-Organizing Maps, Hierarchical Clustering, DBSCAN [2, 3, 4, 5].

Розглянемо методи навчання з учителем, навчання без учителя та навчання з підкріпленням.

Метод навчання з учителем навчає агента на прикладах з правильними відповідями та має відтворювати залежність між даними та відповідями. Популярні алгоритми для навчання є лінійна регресія, логістична регресія, дерева рішень, випадковий ліс.

Лінійна регресія – це алгоритм, що використовується для передбачування, прогнозування, числових значень [2, 3, 4, 5]. Алгоритм моделює лінійну залежність між вхідними змінними $x_1, x_2, \dots, x_n$ і вихідним значенням y.

Формула алгоритму:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n, \quad (1)$$

де:

– $\hat{y}$ – передбачуване значення;

– w_i – коефіцієнти;

– x_i – ознаки;

– w_0 – константа.

Логістична регресія – це алгоритм класифікації, що використовується для прогнозування ймовірності події, за допомогою даних коли залежна змінна є бінарною, а саме коли залежність можна прослідкувати між значеннями 0 та 1 [2, 3, 4, 5].

Формула алгоритму обчислює спочатку лінійну комбінацію ознак за формулою(1), а після застосовує сигмоїдну функцію активації:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad (2)$$

де:

– z – результат лінійної комбінації ознак;

– $\hat{y}$ – це ймовірність, що клас = 1.

Дерева рішень – це ієрархічний алгоритм який послідовно ділить дані за певними умовами, кожен вузол дерева є перевіркою, а лист дерева результатом. Таким чином рішення не буде досягнене поки умова не буде виконана.

Випадковий ліс – це алгоритм за котрим існує ціла купа дерев рішень і вони працюють разом. Головна ідея цього алгоритму об’єднати результати багатьох слабких, щоб віднайти сильний класифікатор.

Є декілька формул для алгоритму, для регресії:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} \hat{y}_i, \quad (3)$$

де $N -$ кількість дерев у лісі.

Формула для класифікації:

$$\hat{y} = mode(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N), \quad (4)$$

де $N -$ кількість дерев у лісі.

Метод навчання без учителя навчає агента шукати закономірності серед наданих даних. Робить це метод за допомогою групування та зменшення кількості даних. Популярні алгоритми для навчання є кластеризація (K-Means), зменшення розмірності(PCA), карти Кохонена [19].

K-Means – це алгоритм, який шукає центр кластерів так, щоб мінімізувати суму квадратів відстаней усіх точок до найближчого центру [2, 3, 4, 5]. Наприклад алгоритм має набір точок $\{x_1, \dots, x_n\}$, то йому треба знайти k центрів у кластерах $\{c_1, \dots, c_k\}$, що мінімізує відстань до центру, таким чином формула буде така:

$$J = \sum_{i=1}^{n} \min_{j \in \{1,\dots,k\}} \left\| x_i - c_j \right\|^2, \quad (5)$$

де:

– $c -$ кластер;

– $k -$ центр кластеру;

– $n -$ кількість точок.

PCA – метод головних компонент, зменшення розмірності, за яким треба знайти нову осі координат вздовж яких розкид даних, тобто дисперсія, є максимальна [2,3,4,5].

Для отримання нової осі слід центрувати дані:

$$\acute{x}_i = x_i - \bar{x}, \quad (6)$$

де:

– $x_i -$ стовпець;

– $\bar{x} -$ середнє по стовпцю.

Обчислити коваріаційну матрицю та знайти власні вектори та значення:

$$\sum \vartheta_i = \lambda_i \vartheta_i, \quad (7)$$

де:

– λ_i – власне значення;

– ϑ_i – власний вектор.

Обрати k векторів ϑ_i з найбільшими λ_i та проєктувати дані на нову вісь.

Карти Кохонена – це нейронна мережа без учителя, що проєктує багатомірні дані на 2D мапу, щоб схожі елементи були поряд [2, 3, 4, 5]. Для отримання результату спочатку проходить ініціація ваг де випадково ініційовані вектори w_i. Після знаходять найкращий нейрон x за методом BMU [2, 3, 4, 5]:

$$\mathrm{c} = \arg\min_j \left\| x_i - w_j \right\|, \tag{8}$$

Оновити ваги BMU (best matching unit) та сусідів та виконати зменшення параметрів з часом.

Метод навчання з підкріпленням навчає агента приймати певне рішення за допомогою нагород та штрафів. Завданням цього штучного інтелекту є навчитись приймати рішення за найбільшою нагородою в довгий строк перспективи. Популярні алгоритми для навчання є звичайне Q-Learning, SARSA, Actor-Critic [20].

Q-Learning – це алгоритм, який навчає агента оцінювати функцію цінності дій $Q(s, a)$, тобто до якої дії вдаватись та за яких обставин [19]. Алгоритм намагається знайти найоптимальніший шлях максимізуючи очікуване значення повної нагороди в усіх послідовних кроках. Також цей алгоритм називають ще алгоритмом off-policy.

Формула оновлення:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{\acute{a}} Q(s_{t+1}, \acute{a}) - Q(s_t, a_t) \right], \tag{9}$$

де:

– $Q(s_t, a_t)$ – поточна оцінка дії;

– r_{t+1} – винагорода після виконання дії;

– γ – коефіцієнт дисконтування ($0 \leq \gamma \leq 1$);

– α – швидкість навчання ($0 \leq \alpha \leq 1$);

– $\max_{\acute{a}} Q(s_{t+1}, \acute{a})$ – найкраща можлива дія в наступному стані.

SARSA – це алгоритм пошуку стратегії за марковським процесом вирішування, а розшифровується назва, як State-Action-Reward-State-Action [2, 3, 4, 5]. Таким чином функція Q залежить від поточного State(стану) агенту, Action(дії), яку обрав агент, Reward(винагороди), яку агент отримає за вибір та State(стану) в який перейде після та Action(дії) котру агент обере після. Також цей алгоритм називають алгоритмом on-policy.

Формула оновлення:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \qquad (10)$$

де:

– $Q(s_t, a_t)$ – поточна оцінка дії;

– r_{t+1} – винагорода після виконання дії;

– γ – коефіцієнт дисконтування ($0 \leq \gamma \leq 1$);

– α – швидкість навчання ($0 \leq \alpha \leq 1$);

– $Q(s_{t+1}, a_{t+1})$ – наступна можлива дія в наступному стані.

Actor-Critic – це алгоритм, що комбінує в собі два підходи, актора та критика. Алгоритм є гібридом між policy-based і value-based RL, який працює з дискретними діями та безперервними [2, 3, 4, 5]. Актор діє за політикою, обирає дії та навчається на оцінці критика. Критик оцінює результати актора, дії, що той виконує та складає політику дій.

Для алгоритму існує дві формули оновлення, для актора та критика. Та актор і критик мають свої параметри, наприклад для актора є параметр θ, що описує таку політику $\pi_\theta(a|s)$. Критик має параметр w, що оцінює цінність стану $V_w(s)$.

Є формула за котрою визначається помилка тимчасової різниці, Temporal Difference Error:

$$\delta_t = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t), \qquad (11)$$

де:

– r_{t+1} – винагорода після виконання дії;

– γ – коефіцієнт дисконтування ($0 \leq \gamma \leq 1$);

– δ_t – сигнал зворотного зв'язку, TD error.

Формула оновлення для актора:

$$\theta \leftarrow \theta + \alpha_a \delta_t \nabla_\theta \ln \pi_\theta (a_t|s_t), \quad (12)$$

де:

– α_a – швидкість навчання актора;

– δ_t – сигнал зворотного зв'язку, TD error.

Формула оновлення для критика:

$$w \leftarrow w + \alpha_c \delta_t \nabla_w V_w(s_t), \quad (13)$$

де:

– α_c – швидкість навчання критика;

– δ_t – сигнал зворотного зв'язку, TD error.

2.3 Специфікація вимог до системи поведінки

1) ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

1.1) Призначення системи, для якої розробляється програмне забезпечення

Призначення системи є покращення ігрового процесу в ігровому застосунку для підтримки актуальності проєкту. Також для проведення аналізу ефективності використання подібних систем в ігрових застосунках даного типу, жанру, геймплею.

1.2) Погодження, що ухвалені в програмній документації

Було погоджено, що для створення системи поведінки противника в ігровому застосунку для карткового бою буде використано готовий проєкт та допоміжні ассети, бібліотеки Unity, наприклад Unity ML-agents.

1.3) Межі проєкту ПЗ

Розробка обмежена сферою розваг та наук, що є аналізом ефективності використання методів ML в ігровому застосунку.

2) ЗАГАЛЬНИЙ ОПИС

2.1) Сфера застосування

Система може бути використана задля розваг, тестування, кіберспорту щодо проходження ігрового застосунку на швидкість, аналізу ефективності систем.

2.2) Характеристики користувачів

Основна характеристика користувачів є: наявність ноутбуку або ПК.

Кінцеві користувачі є: гравці у комп’ютерні ігри.

2.3) Загальна структура і склад системи

Основні частини програмного забезпечення: штучний інтелект, тобто агенти, за методами машинного навчання, середовище рушію Unity.

2.4) Загальні обмеження

Програмне забезпечення обмежене доступом до даних з інтернету.

3) ФУНКЦІЇ СИСТЕМИ ПОВЕДІНКИ ПРОТИВНИКА В ІГРОВОМУ ЗАСТОСУНКУ

3.1) Функція взаємодії з картами

3.1.1) Опис функції

Агент ML повинен мати можливість обирати дії, як картки, та використати їх за своєю поведінкою, що був навчений.

3.1.2) Вхідна і вихідна інформація

Вхід: дані для навчання агенту поведінці.

Вихід: сформована поведінка агента.

3.1.3) Функціональні вимоги

Противник повинен мати власну колоду.

3.2) Функція взаємодії з другорядними діями

3.2.1) Опис функції

Агент ML повинен мати можливість обирати дії, що є другорядними, наприклад закінчити хід в раунді, та використати їх за своєю поведінкою, що був навчений.

3.2.2) Вхідна і вихідна інформація

Вхід: дані для навчання агенту поведінці.

Вихід: сформована поведінка агента.

3.2.3) Функціональні вимоги

Система раундів повинна мати можливість відслідковувати чий хід відбувається в поточний час.

3.3) Функція перемикання агентів ML в ігровому застосунку

3.3.1) Опис функції

Система повинна мати можливість перемикати агентів ML в ігровому застосунку через налаштування його.

3.3.2) Вхідна і вихідна інформація

Відсутні.

3.3.3) Функціональні вимоги

Система повинна мати зрозумілий інтерфейс налаштувань.

4) ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

4.1) Джерела і зміст вхідної інформації (даних)

Основним джерелом вхідної інформації системи є користувач. Змістом інформації є вибір дій, статус персонажа гравця.

4.2) Нормативно-довідкова інформація (класифікатори, довідники тощо)

Вимоги відсутні.

4.3) Вимоги до способів організації, збереження та ведення інформації

Збереження даних відбувається через файл, інформація збережена у вигляді JSON. Ведення інформації відбувається під час виконання застосунку.

5) ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Вимоги до технічного забезпечення не є великими, бо в ігровому застосунку де буде використана система не використовуються великого формату моделі або текстури, тому технічні дані близькі до мінімальних.

Процесор: Intel Core i5 або еквівалентний.

Оперативна пам'ять: не менше 8 ГБ.

Вільне місце на жорсткому диску: не менше 20 ГБ.

6) ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

6.1) Архітектура програмної системи

Архітектура застосунку де використана система, складається з клієнтської частини.

6.2) Системне програмне забезпечення

Ігровий застосунок де використана система, розроблена на ігровому двигуні Unity 2D. На операційній системи Windows 10.

6.3) Мережне програмне забезпечення

Підключення до інтернету не потребується.

6.4) Програмне забезпечення ведення інформаційної бази

Вимоги відсутні.

6.5) Мова і технологія розробки ПЗ

Ігровий застосунок де використана система, має рушій Unity з використанням мови C# та Python.

7) ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

7.1) Інтерфейс користувача

Інтерфейс повинен бути приємним на око, зручним, інтуїтивно зрозумілим, не заважати ігровому процесу гравця.

7.2) Апаратний інтерфейс

Апаратним інтерфейсом є ОС Windows 10.

7.3) Програмний інтерфейс

У ході розробки було використано дві категорії Unity Scripting API: UnityEditor (Animations, Events тощо) та UnityEngine (Analytics, Audio тощо).

7.4) Комунікаційний протокол

Вимоги відсутні.

8) ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

8.1) Доступність

Система доступна в ігровому застосунку для всіх, за умови якщо у користувача є апаратний інтерфейс.

8.2) Супроводжуваність

Не потребує.

8.3) Переносимість

Ігровий застосунок де використана система, сумісний з ОС Windows 10.

8.4) Продуктивність

Продуктивність залежить від характеристик ПК чи ноутбуку.

8.5) Надійність

Підтримка ISO/IEC 42001:2023[21].

8.6) Безпека

Підтримка ISO/IEC TR 24028:2020[22].

9) ІНШІ ВИМОГИ

Усі вимоги сформовано.

Висновки до розділу 2

У цьому розділі описано інструментарій, що застосується підчас останнього розділу роботи, описано відомі методи для навчання за трьома методами машинного навчання, з учителем, без учителя, з підкріпленням, сформовано вимоги до системи, що буде застосована в ігровому застосунку.

В інструментарії наведено ігровий рушій, що застосовується, та його можливості, бібліотека з менеджера пакетів рушію Unity та її можливості в реалізації системи, три варіанта для інтегрованого середовища розробки де описано їх недоліки та переваги.

Для кожного обраного методу машинного навчання розібрано методи з навчання агентів машинного навчання, їх формули, значення, застосування методів.

Сформовано вимоги до системи ігрового застосунку за форматом та наведено потрібну інформацію для документування вимог.

3 АРХІТЕКТУРА, МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Сценарії використання системи

Змоделюємо, для чого потрібна система поведінки противника створена за допомогою методів машинного навчання та які цілі використання системи. Для цього створимо сценарії використання системи, короткий сценарій, поверхневий сценарій та повноцінний сценарій.

Короткий usecase сценарій:

Мета сценарію довести рівень здоров’я героя до певної межі.

Противник обирає картку атаки та атакує героя, зменшуючи здоров’я героя до мінімальної межі.

Поверхневий usecase сценарій:

Мета сценарію – прожити певну кількість раундів бою.

1) Противник зіграв картку з затримкою до наступного раунду.

2) Залишилась одна дія, противник обирає картку з ціною в одну дію, де пріоритетом є захист.

3) Настав хід героя.

4) Герой витрачає усі дії, аби завершити бій до настання ефекту противника, але противник виживає.

5) Настав новий раунд, хід противника, ефект зіграно.

Таблиця 3.1 – Usecase повноцінний

Primary Actor	Гравець
Scope	Ігровий застосунок
Level	Мета користувача
Preconditions	Гравець розпочав бій
Stakeholders and interests	Гравець: виграти бій
Success guarantee	Гравець перемагає всіх ворогів під час бою.

Продовження таблиці 3.1

Main Success Scenario	1. Починається раунд, перший хід починає гравець. 2. Гравець використовує на усі свої дії атакуючі картки. Гравець закінчує хід. 3. Наступає хід противника, противник аналізує стратегію гравця та обирає паттерн, що зупинить агресивного гравця. 4. Противник йде в захист та трохи атакує, завершує свій хід, наступає новий раунд. 5. Гравець продовжує свою тактику, але противника складно швидко перемогти, гравець втрачає багато ресурсів, але перемагає противника.
Extensions	1. Противник перемагає. Гравець не зміг нанести достатньо шкоди для того, щоб перемогти противника і противник встиг нанести смертельної шкоди першим. 2. Противник переможений, але він наніс певну шкоду. Противник протягом бою наносить достатню кількість шкоди, через що герою прийдеться витратити більше ресурсів на відновлення або зупинитися на відпочинок. 3. Противник переможений, але було використано багато ресурсів. Гравець витратив усі особливі карти в своїй колоді і йому потрібно перепочити після бою. 4. Битва була на час і гравець не встиг завершити його. Гравець не встигає завершити бій і через це він переможений.

Кінець таблиці 3.1

Special Requirements	Початок бою.
Technology and Data Variations List	Гравець має CS (character sheet), де збережена вся інформація про персонажа, а саме важливі частини це HP (hit points), Temp HP, LVL (level), Proficiency, AC (armor class).
Frequency of Occurrence	Підчас переходу на новий рівень.

Сценарій вище показує роботу поведінки штучного інтелекту, якби штучний інтелект міг відрізнити тип гри гравця на патерни поведінки та відпрацьовувати протидію. Таким чином, у цьому прикладі використання є гравець, що діє агресивно, й противник, що є штучним інтелектом, побачив патерн гравця та обрав поведінку грати в оборону зрідка атакуючи героя, щоби той витратив більше ресурсів і битва затягнулась.

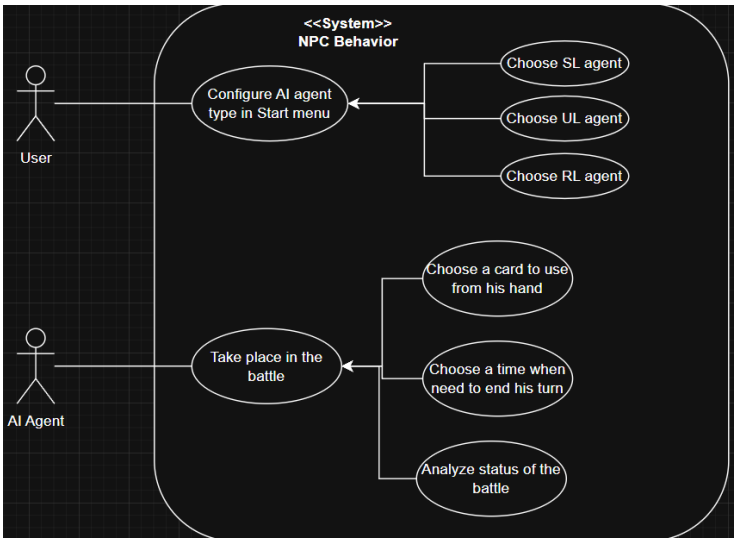

Рисунок 3.1 – Діаграма варіантів використання системи

Система буде передбачати можливість користувача змінити тип агента в початку гри, а сам агент отримує можливості мати власну колоду та обирати карти, що більше підходять для ситуації на полі бою. Щодо цілей штучного інтелекту, це

додати складності, аби гравець не міг передбачити усі дії противника та міг їх тільки вгадати або завчити комбінації.

3.2 Діаграми станів системи

Діаграма станів допоможе показати, як система буде працювати підчас роботи ігрового застосунку та які будуть її дії всередині застосунку, реагуючи на певні цикли, дії, флаги. Діаграма покаже поведінку системи в формі його станів, та призначення діаграми є описати можливу послідовність цих станів та переходів між собою до останнього його життєвого циклу.

Так як система поведінки противника буде реалізована трьома способами, а точніше трьома агентами окремо за кожним методом машинного навчання, то буде розроблено три діаграми станів, щоб показати цикл станів та переходів для кожного агенту поведінки за своїм методом машинного навчання.

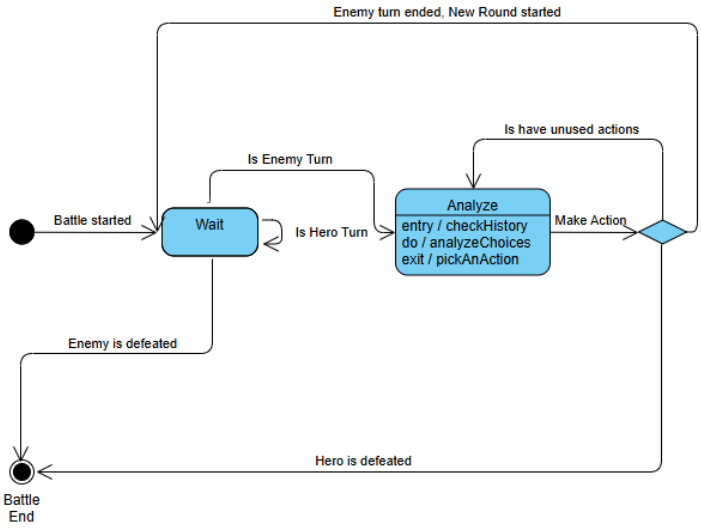

Рисунок 3.2 – Діаграма станів №1 Supervised Learning Machine Learning

На діаграмі станів зображено стани та послідовності системи поведінки противника агента навченого за методом Supervised Learning. На рисунку зображено початок його роботи, а саме початок битви та його два стани, очікування та аналіз із вибором кращого варіанту.

Коли агент знаходиться в стані очікування, він очікує поки герой закінчить свій хід і тоді вже агент перейде до стану аналізу.

Стан аналізу має вхідну дію, як перегляд історії ходів та дій героя, далі йде дія аналізу кращого варіанту, а на виході отримується вже проаналізований найкращий варіант. Система виконує запропоновану дію агентом та перевіряє себе на кількість залишених дій, аби не закінчувати свій хід із запасом дій та використати їх усі можливі варіанти.

В діаграмі є певна послідовність, якщо герой або противник помирає, то бій завершується. Якщо дії підчас ходу закінчились, то починається новий раунд. Якщо противник виконав дію, він перевірить чи використав усі можливі дії та за можливістю виконає додаткову дію.

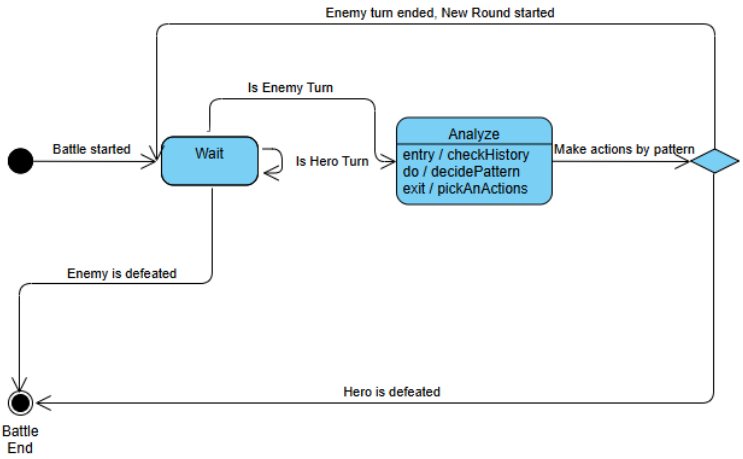

Рисунок 3.3 – Діаграма станів №2 Unsupervised Learning Machine Learning

На цій діаграмі станів, як і на попередньому малюнку зображено два стани та послідовності системи поведінки противника агента, навченого за методом Unsupervised Learning. На рисунку зображено початок його роботи, а саме початок битви та його два стани, очікування та аналіз із вибором кращого патерну, що підходить під стиль гравця або битви.

Коли агент знаходиться в стані очікування, він очікує, поки герой закінчить свій хід і тоді вже агент перейде до стану аналізу.

Стан аналізу має вхідну дію, як перегляд історії ходів та дій героя, далі йде дія аналізу стилю гравця або битви де буде знайдено підходящий паттерн, а на виході отримується послідовність дій за паттерном. Система виконує запропоновану послідовність дій агентом та закінчує хід.

В діаграмі є певна послідовність, якщо герой або противник помирає, то бій завершується. Якщо дії підчас ходу закінчились, то починається новий раунд.

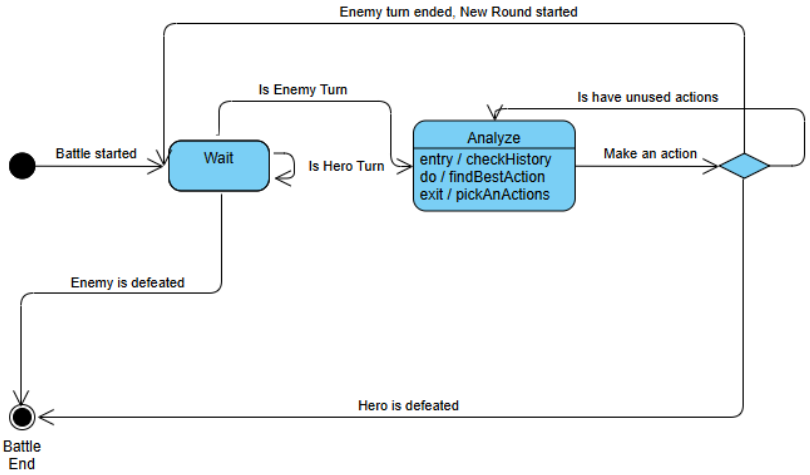

Рисунок 3.4 – Діаграма станів №3 Reinforcement Learning Machine Learning

На цій діаграмі станів, так само зображено два стани та послідовності, але для системи поведінки противника агента, навченого за методом Reinforcement Learning. На рисунку зображено початок його роботи, а саме початок битви та його два стани, очікування та аналіз із вибором кращої дії за балами в поточній ситуації.

Коли агент знаходиться в стані очікування, він очікує поки герой закінчить свій хід і тоді вже агент переходить до стану аналізу.

Стан аналізу має вхідну дію, як перегляд історії ходів та дій героя, далі йде дія аналізу найкращої дії за балами, щоб набрати максимальний бал у цій битві, а на виході отримується дія за найкращим балом. Система виконує запропоновану дію агентом та перевіряє чи є ще можливість діяти, якщо є то проходить цикл знову, а навпаки закінчує хід та переходить до режиму очікування.

В діаграмі є певна послідовність, якщо герой або противник помирає, то бій завершується. Якщо дії підчас ходу закінчились, то починається новий раунд.

3.3 Діаграма класів

Розглянемо діаграму класів, в цій діаграмі буде зображено ресурси, дані, що будуть аналізувати агенти підчас битви. Це буде діаграма на якій зображено класи для листу персонажа та класи, що відображають дані карт та їх застосування.

Щодо листу персонажа, то це загальні та додаткові характеристики персонажа, що визначають його можливі дії та де можна побачити в чому персонаж найкращий або найгірший.

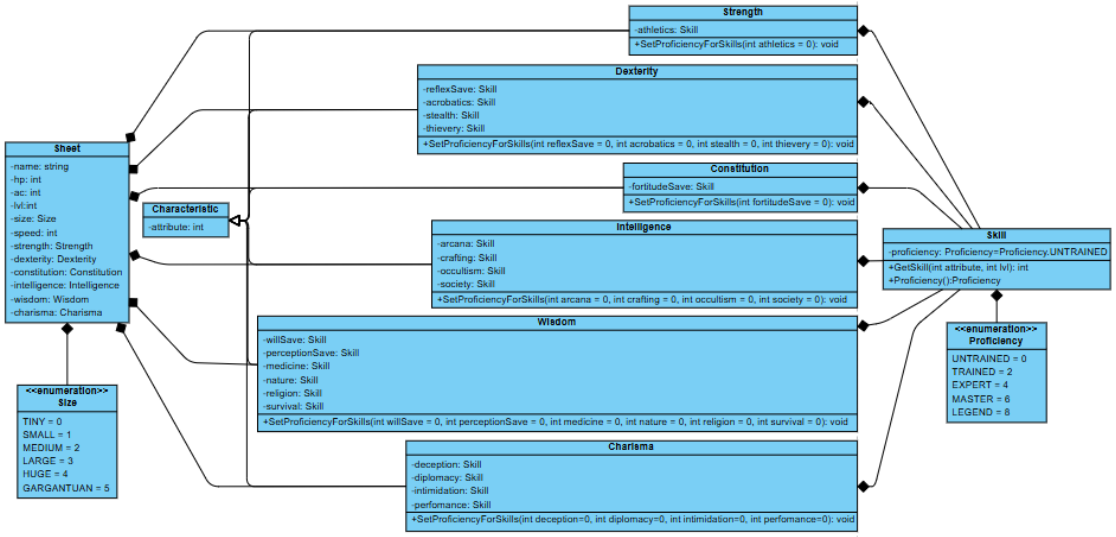

Рисунок 3.5 – Діаграма класів №1 лист персонажа

На рисунку діаграми класів листу персонажа зображено два класи Enum та дев’ять звичних класів. Головним класом, що комбінує усі наведені тут характеристики є клас Sheet. Цей клас зберігає в собі дані, що є name, hp, ac, lvl, size, speed, strength, dexterity, constitution, intelligence, wisdom, charisma. Для певних характеристик є окремі класи, що зберігають в собі професійність у певних скілах, навичках. Ці характеристики є сила, спритність, витримка, інтелект, мудрість, харизма, вони усі наслідують батьківський клас Characteristic, що має атрибут. Ці характеристики мають такі навички, атрибути:

- характеристика сили має навичку атлетики;

– характеристика спритності має навички акробатики, крадіжки, скритності та навичку захисту рефлексів;

– характеристика витримки має навичку захисту стійкості;

– характеристика інтелекту має навички магії, ремесла, окультизму, соціуму;

– характеристика мудрості має навички медицини, природи, релігії, виживання та навички захисту волі і відчуття;

– характеристика харизми має навички обману, дипломатії, залякування, виступу.

Навички мають атрибут, котрий визначається за ступенем класу Enum Proficiency, ці ступені мають Untrained 0, Trained 2, Expert 4, Legend 6, Master 8.

Так само, як із класом Enum Proficiency та класом Skill, є атрибут розміру, що визначається за класом Enum Size, ці ступені мають Tiny 0, Small 1, Medium 2, Large 3, Huge 4, Gargantuan 5.

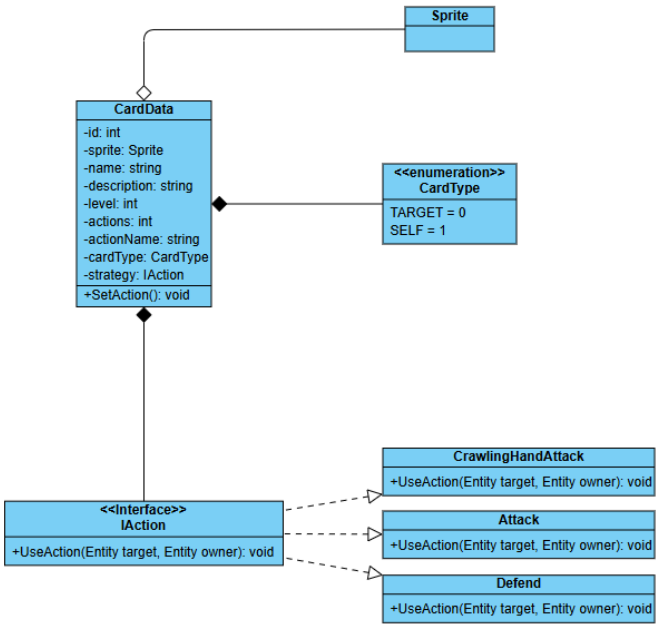

Рисунок 3.6 – Діаграма класів №2 ігрова картка

На цьому рисунку діаграми класів ігрової картки зображено один клас Enum, один інтерфейс IAction та чотири класи, один з яких є базовим від рушія Unity.

Головним класом у цій діаграмі є клас CardData, він зберігає в собі атрибути id, sprite, name, description, level, actions, actionName, cardType, strategy. Атрибут sprite зберігає в собі картинку від базового класу Sprite. Атрибут cardType отримує ступінь від класу CardType Enum, що є Target або Self. Є атрибут strategy, він зберігає в собі дію, що виконує сама ігрова картка, тому тут використовується інтерфейс для реалізації методу різними способами.

Інтерфейс IAction має на реалізацію один метод, котрий буде реалізовано у класах, що наслідують цей інтерфейс. Наразі є такі класи, що реалізують інтерфейс, а саме CrawlingHandAttack, Attack, Defend.

Ось такі дані, як на двох діаграмах класів, будуть аналізувати агенти, окрім ще історії битв та ходів.

3.4 Діаграма розгортання

Діаграма розгортання продемонструє необхідне обладнання для роботи агентів у середовищі ігрового застосунку на рушію Unity, неважливо від того яким методом вони були створені.

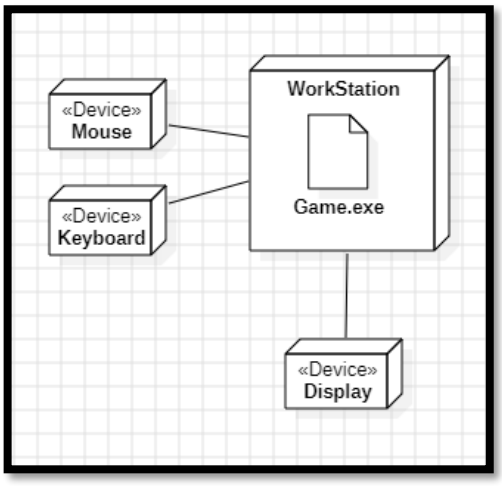

Рисунок 3.7 – Діаграма розгортання ігрового застосунку

На діаграмі зображено, що ігровий застосунок потребує таких девайсів, як дисплей, миша, клавіатура. Тож агенти поведінки штучного інтелекту не будуть

застосовані за межами локального середовища, а точніше не будуть використовувати онлайн сесії або мережу інтернет для навчання.

Хоча й обмеження доступу агентів до мережі може вплинути негативно на їх навчання, але аналіз потребує лабораторного середовища із сухими даними. Бо доступу до мережі інтернету або до онлайнових сесій застосунку може певним чином вплинути на навчання агентів, як позитивно так і негативно.

3.5 Інтерфейс налаштувань

Однією з функціональних вимог було створити зрозумілий інтерфейс для обирання типу агенту ML в ігровому застосунку, щоб змінювати агентів в налаштуванні ігрового застосунку та змінювати складність проходження. Для реалізації інтерфейсу в ігровому застосунку треба створити Mockup налаштувань з перемиканням агентів ML.

Рисунок 3.8 – Mockup інтерфейсу налаштування агентів

На цьому Mockup зображено інтерфейс налаштувань агентів ML, де є меню вибору агенту ML, з самого першого запуску ігрового застосунку в налаштуваннях першим агентом ML буде перший по списку агент.

Для того, щоб обрати агента ML, треба обрати в Selective меню варіант агента ML та після натиснути кнопку Choose. Після цього агент ML буде обрано як поточний та меню буде закрите.

Саме меню налаштувань агентів ML буде вбудоване в інтерфейсі налаштування ігрового застосунку, як ще одне меню. Це меню буде доступне на запуску ігрового застосунку, але меню не буде доступне в загальному меню під час гри.

3.6 Вибір мов програмування, технологій, компонентів програмування

Розробка системи поведінки вимагає вибору мов програмування для роботи з рушієм Unity та для роботи з навчанням агентів ML за допомогою алгоритмів навчання. З мов програмування, що доступні з інструментарію обрано мову програмування C# для роботи з ігровим рушієм Unity.

Мова C# є гарним вибором для роботи над ігровим застосунком в ігровому рушію Unity, однак є альтернатива у вигляді C++, тому слід описати переваги та недоліки обох мов у поставленій задачі.

Рисунок 3.9 – Платформа .NET, мова програмування C#

C# або C Sharp – об’єктно-орієнтована мова програмування розроблена компанією Microsoft у рамках платформи .NET. Ця мова програмування поєднує частку простоти мови C з потужністю мови C++, є зручною та типобезпечною. Платформа .NET, що C# є частиною цього, поділяється на дві частини, кросплатформений .NET Core та Windows-орієнтований .NET Framework. Також розробники Microsoft постійно підтримують мову програмування C# [23].

Переваги мови програмування C#:

– простота синтаксису;
– автоматичне керування пам’яттю, Garbage Collector;
– висока безпека типів;
– розвинена стандартна бібліотека .Net;
– кросплатформеність, підтримка Windows, Linux, macOS;
– підтримка сучасних концепцій, асинхронність, LINQ, делегати, лямбда, тощо;
– швидка розробка, інтеграція з Visual Studio, потужний IntelliSense, дебагінг, автоформатування;
– основна мова програмування в Unity.

Недоліки мови програмування C#:

– менший контроль над пам’яттю, не можливо керувати розміщенням і видаленням об’єктів;
– потреба в середовищі .NET, бо виконується код в віртуальній машині CLR;
– менша швидкодія через проміжну компіляцію;
– менше застосування у вбудованих системах, наприклад контролери.

Рисунок 3.10 – Мова програмування C++

C++ – це високорівнева, компільована, об’єктно-орієнтована мова програмування, створена Б’ярном Страуструпом, як розширення для мови C. Вона поєднує швидкість і гнучкість низькорівневого програмування, як в C, тільки з можливостями структурного та об’єктно-орієнтованого підходу [24]. Мова програмування C++ використовується для системного програмування, розробки програмного забезпечення, створення драйверів, потужних серверних та

клієнтських програм та відеоігор. Також мова програмування C++ підтримує декілька парадигм, процедурне програмування, ООП, узагальнене та функціональне програмування.

Переваги мови програмування C++:

- висока продуктивність через роботу напряму;
- повний контроль над пам’яттю;
- можливість низькорівневого доступу;
- велика кількість бібліотек і стандартів;
- кросплатформеність;
- підтримка ООП і шаблонів;
- підходить для системного програмування.

Недоліки мови програмування C++:

- складність синтаксису і навчання мови;
- відсутність автоматичного прибирання пам'яті;
- вища ймовірність помилок через проблеми з пам’яттю;
- повільний цикл розробки, немає єдиної платформи як з .NET;
- менше готових інструментів.

Для проведення аналізу та алгоритмів навчання для агентів ML обрано мову Python, для конфігурації тренувань моделей агентів є мова YAML, а для збереження даних є мова JSON. Наведемо опис цих двох мов та чому було обрано саме них.

Рисунок 3.11 – Мова програмування Python

Python – інтерпретована, об’єктно-орієнтована, високорівнева мова програмування створена розробником Ґвідо ван Россумон. Її ідея полягає в простоті

та читабельності коду, тому синтаксис читабельний та зрозумілий, величезна стандартна бібліотека та підтримка процедурної, об’єктно-орієнтованої та функціональної парадигми програмування [25]. Через простоту синтаксису вивчення цієї мови програмування є простим, бо окрім простоти є й купа відео-гайдів, документація, тощо. Мова є кросплатформеною та працює на Windows, Linux, macOS, Android, Raspberry. Має потужні бібліотеки для роботи з навчання агентів ML, такі як pandas, NumPy, scikit-learn, Matplotlib, Seaborn.

YAML – це формат зберігання та обміну даних, що є зручний для читання людини форматом, створений Кларком Евансом, дуже схожий на JSON, XML, але більше використовується для конфігураційних файлів у багатьох системах, наприклад Github, Docker. Розшифровується як YAML Ain’t Markup Language, тобто це не мова розмітки, а структурованих даних [26].

JSON – це формат зберігання та обміну даних, який є простим, легким і зрозумілим для людини та машини, створений Дугласом Крокфордом. Формат дозволяє описувати структури даних та об’єкти сам формат описується, як JavaScript Object Notation, тобто він базується на синтаксисі JavaScript [27]. Спосіб представлення даних є у вигляді пар ключ-значення, списків і вкладених об’єктів. JSON дуже часто використовують для обміну даних між сервером та клієнтом.

Компоненти, що потрібні для роботи з навчання, це бібліотеки для мови програмування Python [28]. Серед потрібних бібліотек є:

- scikit-learn – бібліотека машинного навчання, що дає можливість використовувати функціональність різних алгоритмів для навчання, таких як лінійна регресія, випадковий ліс, тощо;
- skl2onnx – бібліотека машинного навчання, що за допомогою методів бібліотеки scikit-learn дає можливість перетворити модель в формат ONNX. Це потрібно для рушію Unity.
- pandas – бібліотека надає можливості маніпуляції даними та аналізу даних;

– NumPy – бібліотека, що вважається розширенням мови Python, дає розширені можливості для багатовимірних масивів і матриць, високорівневі математичні функції та операції над масивами;

– Matplotlib – бібліотека для візуалізації даних у 2D вимірі та також у 3D вимірі;

– PyTorch – бібліотека машинного навчання на основі іншої з назвою Torch, що застосовується для задач із комп'ютерним зором та обробки природної мови.

Для роботи з навчанням агентів потребується віртуальне середовище з Python версії 3.10.12, тому для створення такого буде застосовано дистрибутив Anaconda. За його допомогою можна створити віртуальне середовище з потрібними налаштуваннями, конфігурацією, бібліотеками.

Рисунок 3.12 – Дистрибутив Anaconda

Anaconda – це дистрибутив, що поєднує в собі велику кількість модулів та різних програмних продуктів у більшості для мов програмування R та Python, спеціалізується на застосуванні методів машинного навчання, обробці даних, аналізу даних та має на меті спрощування розгортання та управління пакетами даних [29].

Для роботи з Unity є пакети Unity ML-agents, ML-agents Toolkit. Про роботу пакету Unity ML-agents було роз'яснено в першому розділі про інструментарій, пакет ML-agents Toolkit є інструментарієм для роботи з основним пакетом.

Для роботи з написанням коду є можливості використання інтегрованих середовищ розробки Visual Studio та Visual Studio Code. Visual Studio буде використаний задля написання коду для скриптів ігрового застосунку. Для написання YAML, JSON конфігураційних файлів та файлів для збереження даних.

Опис інтегрованих середовищ розробки було наведено в другому розділі, підрозділу інструментарію та окрім опису там було наведено переваги та недоліки середовищ розробки.

Таке застосування середовищ розробки пояснюється тим, що написання файлів конфігурації та файлів для збереження даних більш зручне за допомогою середовища розробки Visual Studio Code. Для написання коду в скриптах ігрового застосунку використовується Visual Studio, бо він більш зручний у поєднанні з середовищем Unity.

Для користування буде корисним встановити бібліотеку TensorFlow, що є бібліотекою для машинного навчання та має інструменти для візуалізації даних навчання агента штучного інтелекту. Інструмент для візуалізації даних називається TensorBoard.

Висновки до розділу 3

В результаті розділу було створено декілька UML діаграм, це є діаграми станів, діаграми класів, діаграма розгортання, також створено декілька сценаріїв використання з котрих є короткий, повноцінний та поверхневий, окрім сценаріїв, діаграм також створено mockup інтерфейсу налаштувань агентів ML та описано мови програмування, технології, компоненти, що будуть використані в цій роботі.

В діаграмах станів описано стани трьох агентів ML коли вони активно працюють в ігровому застосунку, в діаграмах класів описано основні класи, що будуть використані як дані для тренування агентів ML. Діаграма розгортання коротко описала компоненти потрібні для взаємодії з ігровим застосунком та системою поведінки.

Сценарії використання показали роботу агентів ML, як систему поведінки противника, що буде діяти та виконувати задачі в ігровому застосунку.

В останньому підрозділі описано мови програмування, технології та компоненти, що будуть використані в роботі.

4 РОЗРОБКА ТА АНАЛІЗ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Встановлення та налаштування інструментів

Для встановлення Unity пакету ML-Agents та ML-Agents Toolkit потребується спочатку встановити віртуальне середовище для Python версії 3.10.12 [30].

Для цієї задачі потребується встановити дистрибутив Anaconda, цю задачу виконати легко, бо для встановлення потребується на сайті завантажити встановлювач та обрати його на виконання.

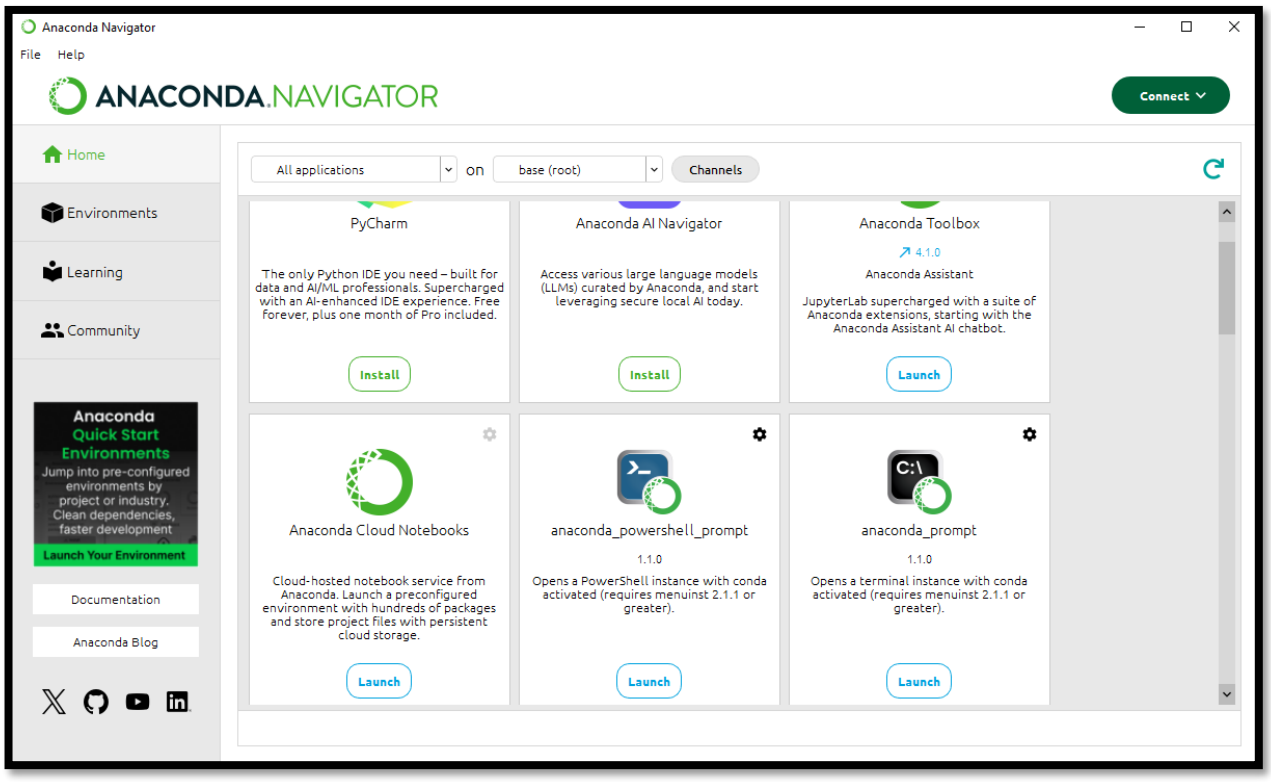

Рисунок 4.1 – Інтерфейс Anaconda

Після встановлення в інтерфейсі дистрибутива треба запустити оболонку PowerShell Anaconda та створити віртуальне середовище вказавши потрібну версію Python 3.10.12 та погодившись із командою на виконання.

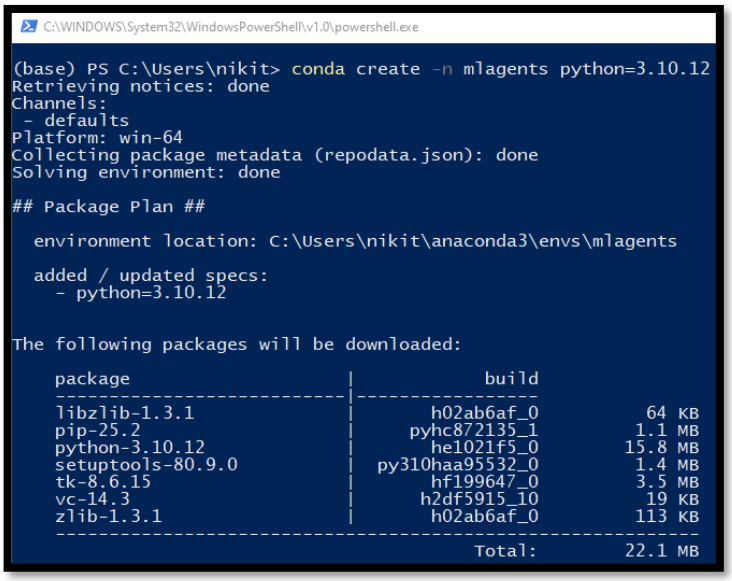

Рисунок 4.2 – Створення віртуального середовища

Після створення віртуального середовища потребується завантажити додаткові бібліотеки, що будуть використані та можуть знадобитись в роботі. Для цього треба запустити віртуальне середовище та за допомогою команди «conda install» завантажити потрібні бібліотеки, для PyTorch треба використати команду «pip3 install». Усі потрібні бібліотеки були наведені в останньому підрозділі третього розділу роботи.

Після встановлення потрібних бібліотек знадобиться встановити бібліотеку ML-Agents для python, тому знаходимо бібліотеку в github, знаходимо підходящу версію бібліотеки та завантажуємо. Після треба зайти в бібліотеку та завантажити в віртуальне середовище середовище імен та саму бібліотеку за допомогою команди «python -m pip install».

Коли всі бібліотеки були налаштовані та залежності між їхніми версіями налаштовано, слід пересвідчитись у роботі головної бібліотеки mlagents. Для перевірки достатньо вивести в консолі команду «mlagents-learn --help» і якщо не було жодної помилки то середовище налаштовано.

Для взаємодії між проєктом, де будуть створюватись та навчатись агенти, потрібно також встановити пакет ml-agents за допомогою менеджера пакетів Unity.

4.2 Налаштування середовища для навчання агентів

Для навчання агентів у середовищі треба створити окрему сцену без зайвого функціоналу, інтерфейсу, графіки, речей, що не будуть впливати на навчання агента, його логіку.

Таким чином створено окремий скриптовий файл, що дублює головний функціонал та прибирає зайвий, два актора на основному екрані, простий скрипт для імітування дій гравця розписаним за трьома патернами, атака, захист, приголомшлива атака.

```
public void AIAct(Entity target)
{
    int choice = UnityEngine.Random.Range(0, 3);
    patterns[choice].Invoke(self, target);
    self.isTurnEnded = true;
}

private void AttackPattern(Entity self, Entity target)
{
    Debug.Log("Attack");
    self.CharacterSheet.Cards[0].UseCard(self, target);
    self.CharacterSheet.Cards[0].UseCard(self, target);
}

private void DefendPattern(Entity self, Entity target)
{
    Debug.Log("Defend");
    self.CharacterSheet.Cards[0].UseCard(self, target);
    self.CharacterSheet.Cards[1].UseCard(self, target);
}

private void AccurateAttackPattern(Entity self, Entity target)
{
    Debug.Log("AccurateAttack");
    self.CharacterSheet.Cards[2].UseCard(self, target);
    self.CharacterSheet.Cards[1].UseCard(self, target);
}
```

Рисунок 4.3 – Блок коду імітатора гравця

За першим патерном гравець двічі атакує суперника. За другим патерном гравець атакує суперника та встає в захист. За третім патерном гравець хапає противника та встає в захист. Ця поведінка зроблена спеціально трохи слабкою аби агенти змогли навчитись перемагати.

Для того щоб слідкувати за прогресом агентів було створено інтерфейс, що відображає навчання агента.

Рисунок 4.4 – Інтерфейс сцени з агентом навчання з підкріпленням

Наприклад для агента ШІ за методом навчання з підкріпленням створено інтерфейс із слідкуванням за епізодом навчання та накопиченої нагороди протягом цього епізоду.

4.3 Створення агентів штучного інтелекту за методами машинного навчання, їх навчання та аналіз навчання

Для створення агентів було налаштовано сцену ігрового застосунку та налаштовано віртуальне середовище Python для навчання агентів. Тож уся підготовка була виконана і тепер на черзі створення агентів.

Першим агентом на створення обрано агента штучного інтелекту за методом навчання з підкріпленням. Створення першим саме цього агента обумовлено тим, що його створення буде більш простішим, бо усі потрібні інструменти вже існують в середовищі Unity та взагалі це найбільш застосований метод навчання в спільноті Unity.

Для створення агенту клас агенту має наслідувати батьківський клас Agent, що представлений ресурсним пакетом ML-agents. При наслідування цього класу додаються методи, за котрими агенту дається можливість обирати дії основані на даних, що збирає агент підчас епізодів.

Наслідування класу Agent дає велику кількість методів для навчання агента, але з основних методів це Initialize, OnEpisodeBegin, CollectObservations, OnActionReceived, EndEpisode, Heuristic.

Після наслідування класу Agent в компонентах до агенту додається компонент Behavior Parameters, в цьому компоненті дається можливість налаштувати поведінку навчання агента.

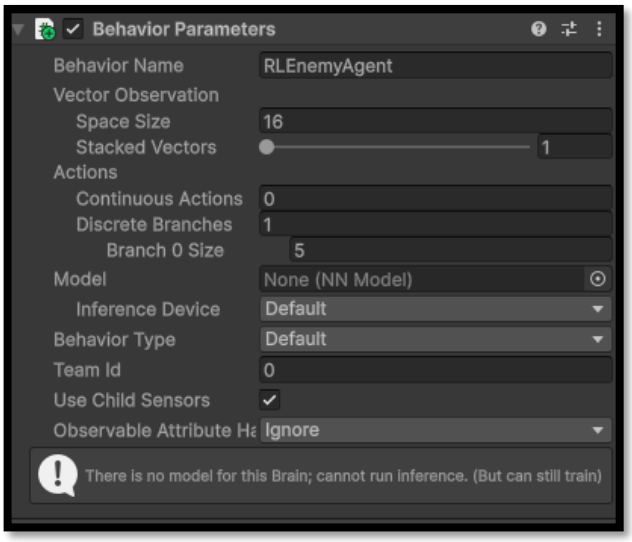

Рисунок 4.5 – Компонент визначення поведінки навчання

Методи Initialize, OnEpisodeBegin, OnActionReceived відпрацьовують у певний момент праці застосунку або навчання, наприклад Initialize відпрацьовує в момент роботи між Awake та Start, OnEpisodeBegin відпрацьовує на початку епізоду навчання, OnActionReceived відпрацьовує коли для агента запрошується дія, працює цей метод разом із методом CollectObservations коли той збере дані для опрацьовування їх та визначення дії. Метод EndEpisode закінчує епізод навчання, а метод Heuristic створений для певного навчання де агент копіює дії, рішення гравця. Таким чином виходить така стандартна послідовність Initialize => [OnEpisodeBegin => [CollectObservations => OnActionReceived] => EndEpisode].

Для запиту на дію агента існує декілька варіантів, наприклад створення компоненту Decision Requester або вручну в коді виконувати метод RequestDecision.

Агент намагається навчитись правильно обирати карти аби обіграти противника та знизити його рівень HP до 0. Агент має колоду карт у кількості дев’яти, серед колоди є чотири типи карт, атакуюча карта, карта з ефектом захисту, карта з ефектом пониження захисту противника, карта з зняттям ефекту пониження захисту з себе.

Агенту надаються для дослідження дані поточного HP власного та противника, захист власний AC та противника, кількість карт на руках, та самі карти разом із їх типом. Таким чином агент споглядає за 16 значеннями.

Агенту для дії слід обирати одну карту з п’яти, тобто агент матиме дискретну гілку розміром в п’ять можливих дій.

```
public override void CollectObservations(VectorSensor sensor)
{
    Debug.Log("CollectObservations");
    sensor.AddObservation(self.CharacterSheet.HPCurrent);
    sensor.AddObservation(self.CharacterSheet.AC.Value);
    sensor.AddObservation(cardHand.Count);
    sensor.AddObservation(turnActions);

    sensor.AddObservation(target.CharacterSheet.HPCurrent);
    sensor.AddObservation(target.CharacterSheet.AC.Value);

    for (int i = 0; i < handSize; i++)
    {
        if (i < cardHand.Count)
        {
            sensor.AddObservation((int)cardHand[i].ActionType);
            sensor.AddObservation(cardHand[i].Actions);
        }
        else
        {
            sensor.AddObservation(-1);
            sensor.AddObservation(-1);
        }
    }
}
```

Рисунок 4.6 – Блок коду споглядання за набором даних агентом для навчання

В методі OnActionReceived, де подається параметром ActionBuffers actions, що слугує визначеною дією, визначаються дії котрі агент буде виконувати та окрім самих дій в цьому блоці також визначається нагорода за певні дії, щоб видати нагороду агенту слід використовувати метод AddReward.

Окрім визначення нагороди також слід закінчувати ланцюг задач методом EndEpisode, і в цьому скрипті цей метод прописаний в середині методу IsGameOver, що знаходиться в скрипті тренувальної сцени, де перевіряються умови перемоги мішені та агента та визначається фінальна нагорода для агента. Також нагорода визначається в методі EndTurn, бо ігровий застосунок має покрокові битви де є раунди та ходи акторів. В цьому методі усі дані записуються, наприклад збір усієї нагороди для відображення нагороди в UI (user interface) підчас навчання, та хід агента завершується і передається даль по черзі.

Тобто агент окрім навчання обирати найкращий варіант серед можливих, вчиться вигравати в суперника коли він не має переваги і ходить після противника, тобто останнім, але й коли він ходить першим в черзі ініціативи. Та слідкує за своїм захистом та захистом противника аби знати коли треба підвищити захист, а коли треба зняти з себе зниження захисту.

```
public override void OnActionReceived(ActionBuffers actions)
{
    Debug.Log("OnActionReceived");

    reward = GetCumulativeReward();

    int cardIndex = actions.DiscreteActions[0];

    if (cardIndex ≥ cardHand.Count)
    {
        AddReward(-0.1f);
        EndTurn();
        return;
    }

    CardData selectedCard = cardHand[cardIndex];

    if (selectedCard.Actions > turnActions)
    {
        AddReward(-0.2f);
        EndTurn();
        return;
    }

    int lastTargetHP = target.CharacterSheet.HPCurrent;

    selectedCard.UseCard(self, target);
    turnActions -= selectedCard.Actions;
    DropCardToDiscard(selectedCard);

    if (target.CharacterSheet.HPCurrent < lastTargetHP)
    {
        AddReward(0.5f);
    }
    if (turnActions ≤ 0 || cardHand.Count == 0)
    {
        EndTurn();
        return;
    }
    RequestDecision();
}
```

Рисунок 4.7 – Блок коду вибору дії агентом та надання нагород за дії

Так як агенту має запрошуватись три дії, бо кожен актор має під час свого ходу три дії, для запиту на дію вручну прописується метод RequestDecision замість додавання компоненту Decision Requester, бо тоді агенту задавалось виконувати дії коли його хід ще не настав.

```
public void OnAgentDamage()
{
    if (self.CharacterSheet.HPCurrent < lastknownHP)
    {
        AddReward(-0.5f);
        return;
    }
    AddReward(0.3f);
}

public void OnAgentDeath()
{
    AddReward(-2.0f);
    reward = GetCumulativeReward();
    EndEpisode();
}

public void OnPlayerDeath()
{
    AddReward(2.0f);
    reward = GetCumulativeReward();
    EndEpisode();
}
```

Рисунок 4.8 – Блок коду нагород за дії

Агент отримує нагороду за:

1) -0.1 – якщо агент обирає карту, якої нема в нього на руці, задля того споглядачу надається п’ять карт в будь якому випадку, але неіснуючі карти мають значення -1, тож агент вчиться не обирати негативні значення у якості карт;

2) -0.2 – якщо агент обирає карту в котрої кошт дій більше ніж в наявності в агента;

3) 0.5 – якщо агент наносить шкоду противнику, то отримує позитивну нагороду;

4) -0.5 – якщо агенту наносять шкоду;

5) 0.3 – якщо агенту не наносить шкоду;

6) -2.0 – якщо агент програє битву;

7) 2.0 – якщо агент виграє битву.

```
[INFO] Resuming training from step 206987.
[INFO] RLEnemyAgent. Step: 250000. Time Elapsed: 401.580 s. Mean Reward: -0.712. Std of Reward: 2.034. Training.
[INFO] RLEnemyAgent. Step: 300000. Time Elapsed: 843.327 s. Mean Reward: -0.683. Std of Reward: 2.043. Training.
[INFO] RLEnemyAgent. Step: 350000. Time Elapsed: 1281.406 s. Mean Reward: -0.649. Std of Reward: 2.093. Training.
[INFO] RLEnemyAgent. Step: 400000. Time Elapsed: 1706.168 s. Mean Reward: -0.631. Std of Reward: 2.114. Training.
[INFO] RLEnemyAgent. Step: 450000. Time Elapsed: 2157.437 s. Mean Reward: -0.565. Std of Reward: 2.165. Training.
[INFO] RLEnemyAgent. Step: 500000. Time Elapsed: 2609.150 s. Mean Reward: -0.657. Std of Reward: 2.071. Training.
[INFO] Exported results\rl_agent_ch\RLEnemyAgent\RLEnemyAgent-499986.onnx
[INFO] Exported results\rl_agent_ch\RLEnemyAgent\RLEnemyAgent-500007.onnx
[INFO] Copied results\rl_agent_ch\RLEnemyAgent\RLEnemyAgent-500007.onnx to results\rl_agent_ch\RLEnemyAgent.onnx.
```

Рисунок 4.9 – Навчання агента за методом навчання з підкріпленням

На рисунку зображено результати тренувань агента за описаним методом.

Агент пройшов 500000 кроків навчання та отримав у результаті середній результат нагород, як -0.657. Враховуючи, що характеристики персонажа, якого агент використовує для тренування, є слабшими за характеристики мішені, то результат є непоганим, бо для тренування у якості мішені використовується персонаж другого рівня з трохи слабким вибором дій, а для агента використовується персонаж з негативним першим рівнем. Такий вибір обумовлений, бо персонаж, за яким навчався агент є монстром, що атакує в зграях, тому він заслабкий у бою один на один.

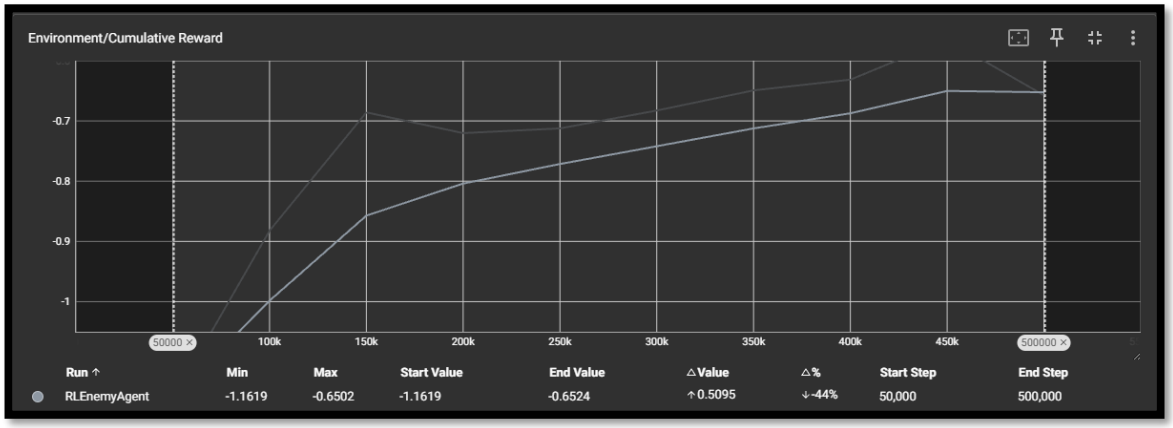

Рисунок 4.10 – Графік середньої нагороди агента RLEnemyAgent

На графіку можна споглядати зміну середнього значення нагороди за кінець епізоду, котрий отримував агент підчас навчання. На графіку записано усі середні значення нагород починаючи з відмітки 50000 кроків до 500000 кроків.

Протягом навчання агент навчався обирати більш вигідні дії з запропонованих та навчався на своїх помилках та успіхах, де в результаті з середнього значення нагороди в -1.1619 агент зрівняв свій успіх до -0.6524, що є доволі відмінно від перших кроків агента.

Однак на графіку навчання видно, що починаючи з 450000 кроків результат навчання став більш сталим.

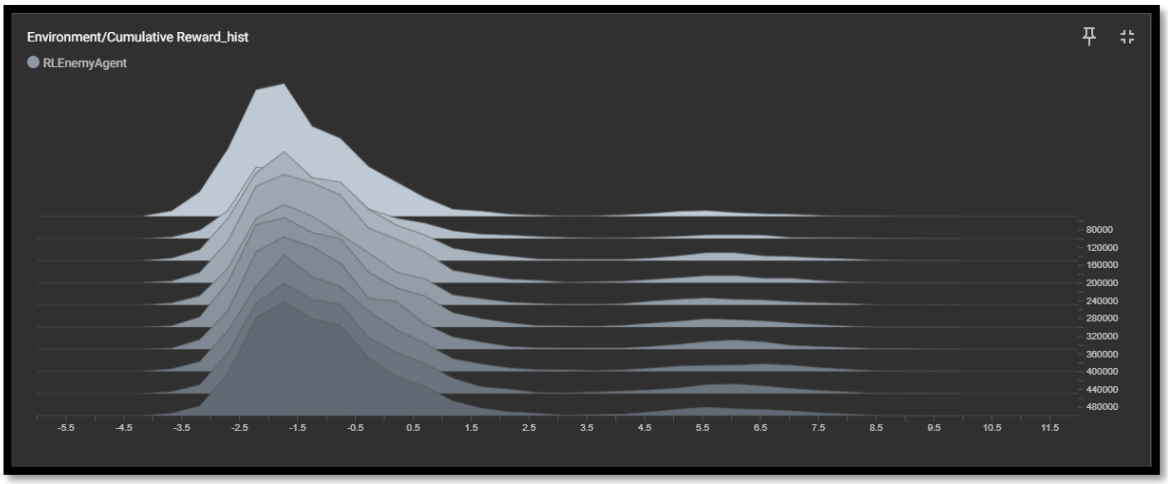

Рисунок 4.11 – Гістограма середньої нагороди агента RLEnemyAgent

На цій гістограмі зображено наскільки часто агент отримував нагороду за епізод та за цим можна побачити, що агент частіше програвав та отримував негативну нагороду, однак є позитивні моменти. Агент мав тенденцію вигравати, що можна побачити на збільшенні значення між оцінкою 4.5-8.5, що є доказом того, що незважаючи на переважаючу силу агент все ще міг виграти.

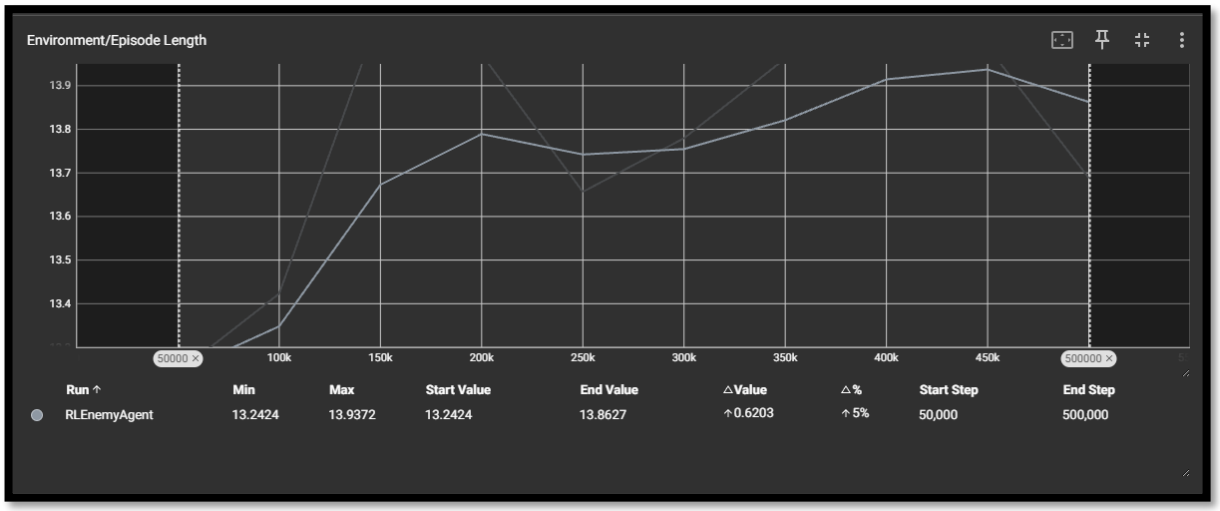

Рисунок 4.12 – Графік довжини епізодів агента RLEnemyAgent

На цьому графіку показано довжину епізодів, що залежить від частоти програшів та виграшів агента, якщо графік має найвищу позначку, то це означає, що бій був довгий та затягнутий. З початку навчання битви завершались швидко, але з навчанням бої стали затягнуті та не завершувались так швидко.

Для створення агенту за методом машинного навчання, навчання з учителем, supervised learning, треба самостійно створити модель в python. Так як пакет ресурсів Barracuda не підтримує всі типи шарів ONNX (open neural network exchange), а саме TreeEnsembleClassifier/ZipMap, які створюються для класичних алгоритмів навчання, таких як Random Forest, Decision Tree, Gradient Boosting, то для створення агента було обрано створити модель нейронної мережі, бо Barracuda приймає такий тип ONNX.

Нейронна мережа навчається за методом MLP. На вхід до моделі було надано дані, що були записані підчас роботи першого створеного агента в роботі за методом навчання з підкріпленням. За його роботою було зроблено близько 6800 записів.

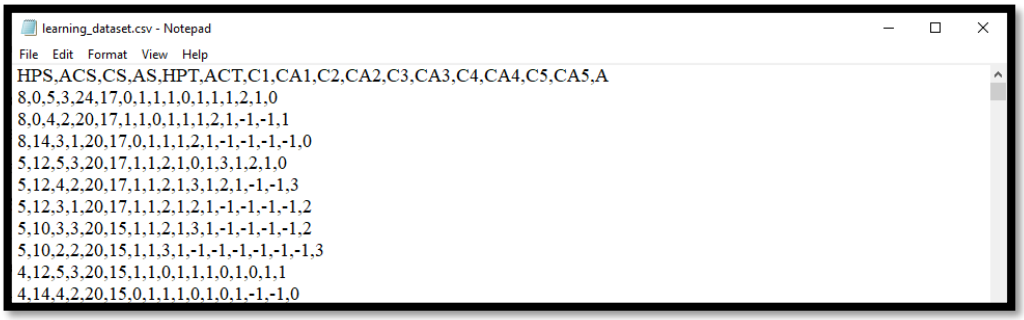

Рисунок 4.13 – Записані дані для навчання моделі агента з учителем

Дані було записано в CSV файл та використано як дані для навчання моделі. Для створення моделі дані було розбито на вибірку станів та вибірку дій, де після створено модель, критерій та оптимізатор.

Після створення потрібних параметрів проведено тренування з першої епохи до двохсотої епохи. В нейронній мережі існувало три рівня нейронів, де перший та третій вписувались командно, а другий шар мав 128x128 нейронів.

```python
data = pd.read_csv("Quinn/learning_dataset.csv")
X = data.drop(['A'], axis=1).values.astype(float)
y = data['A'].values.astype(int)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
class Net(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.fc1 = nn.Linear(input_size, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, output_size)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
model = Net(X.shape[1], len(set(y)))
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
writer = SummaryWriter(log_dir="runs/supervised_training")
for epoch in range(200):
    inputs = torch.tensor(X_train, dtype=torch.float32)
    labels = torch.tensor(y_train, dtype=torch.long)
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    writer.add_scalar("Loss/train", loss.item(), epoch)
    with torch.no_grad():
        val_outputs = model(torch.tensor(X_val, dtype=torch.float32))
        val_pred = val_outputs.argmax(dim=1)
        val_acc = (val_pred == torch.tensor(y_val)).float().mean()
        writer.add_scalar("Accuracy/val", val_acc.item(), epoch)
writer.close()
dummy_input = torch.randn(1, X.shape[1])
torch.onnx.export(model, dummy_input, "SLEnemyAgent.onnx", input_names=['input'], output_names=['output'])
```

Рисунок 4.14 – Блок коду Python із тренування моделі за навчанням з учителем

Після навчання моделі та запису її результатів до дошки tensor були отримані результати навчання моделі, які переглянемо далі. Однак окрім цього модель була експортована до типу ONNX, тепер цю модель можна використати в редакторі Unity та використати її можливості.

Для роботи із навченою моделлю створено новий скрипт з ім’ям SupervisorAgent.

```
public class SupervisedAgent : MonoBehaviour, IBaseAI
{
    public NNModel modelAsset;
    private Model model;
    private IWorker worker;

    params

    private List<float> observations;

    void Start()
    {
        model = ModelLoader.Load(modelAsset);
        worker = WorkerFactory.CreateWorker(WorkerFactory.Type.Auto, model);
    }

    public void OnEpisodeStart()
    {
        SetDeck(self.CharacterSheet.Cards);
        FillHandArea();
    }
```

Рисунок 4.15 – Блок коду скрипту SupervisorAgent де призначення моделі

В цьому відрізку показано, як модель призначається та ініціалізується агент.

Також показано метод, за яким ініціалізується колода агента, та його рука карт для проведення битви. Цікавого з того, що модель легко завантажити з самого редактора Unity і це не займає багато часу. В параметрах є список спостерігачів, туди записуємо підчас роботи агента стани гри та передаємо їх агенту для вибору дій.

Як в інших агентах при виклику методу AIAct, агент збирає стан гри та в методі обирає дію, карти, і після виконання дій завершує свій хід. На відміну від минулого агента, цей агент обирає тип карти котру треба зіграти, а минулий агент обирав карту за індексом.

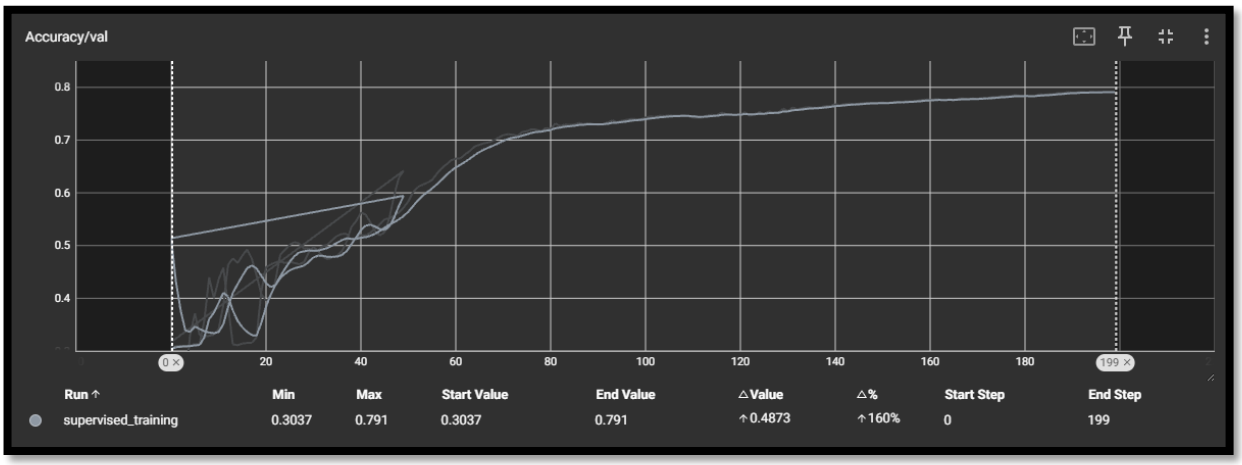

Рисунок 4.16 – Графік точності щодо значень агента SLEnemyAgent

На графіку зображено точність прогнозування значень за станами битви з періоду від першої епохи до двохсотої епохи. За графіком можна побачити, що за навчанням моделі до двадцятої епохи модель вчилась бездоганно, однак в наступних епохах точність погіршилась, але з новими епохами точність стала краще і ближче до останньої епохи точність стала близька до 0.8, 0.791, а спочатку точність була 0.3037. Середнім значенням є значення 0.4873. Дуже добре є те, що значення передбачення близько до одиниці, це є гарним результатом тренування.

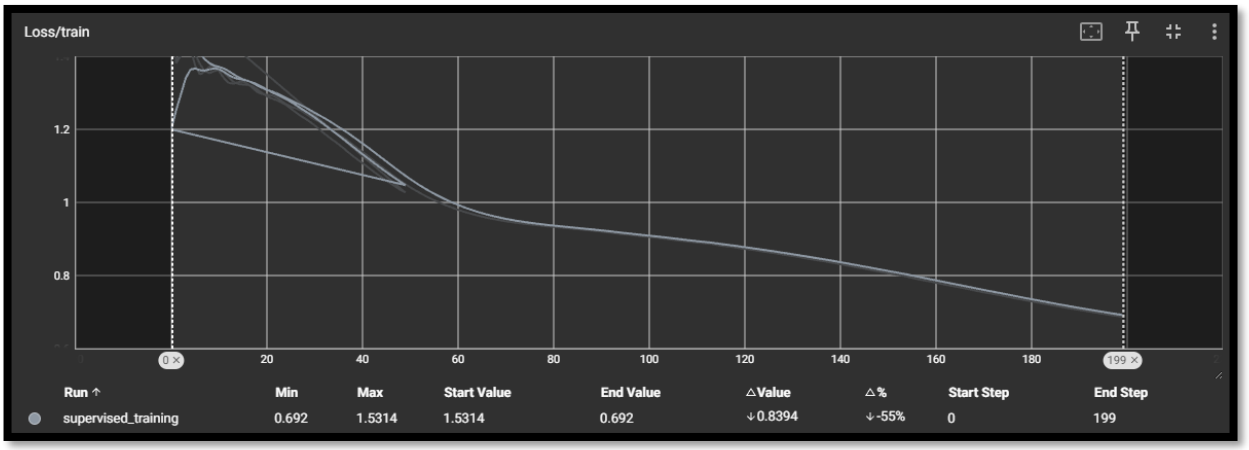

Рисунок 4.17 – Графік втрат при тренуванні агента SLEnemyAgent

За цим графіком можна споглядати кількість втрат при навчанні агента з першої до двохсотої епохи. З першої епохи значенням втрат було 1.5314, а в кінці самого тренування, навчання, стало близько 0.692. Це є гарним результатом, але не ідеальним.

Останній агент на створення є агентом за методом машинного навчання навчання без вчителя, unsupervised learning, для цього агента використаємо кластеризацію станів, що були зібрані в грі та під кожен кластер, а їх буде чотири, буде використаний той тип карти за котрим є кластер стану гри в ігровому застосунку підчас битви.

Для кластеризації використаємо метод K-Means та збережемо результати кластеризації в файл JSON для подальшого його зчитування та використання вже підчас гри отримуючи стан гри та шукаючи його кластер для отримання типу карти, що треба зіграти агенту. Створення моделі буде виконуватись у Python.

Для кластеризації будуть надані дані CSV файлу, що був попередньо використаний для навчання моделі із застосування нейронної мережі.

```python
data = pd.read_csv("Quinn/learning_dataset.csv")
X = data.drop(['A'], axis=1).values
kmeans = KMeans(n_clusters=4, random_state=22)
labels = kmeans.fit_predict(X)
centers = kmeans.cluster_centers_
writer = SummaryWriter("runs/unsupervised_training")
for dim in range(centers.shape[1]):
    for cluster_id in range(4):
        writer.add_scalar(f"Centers/Dim{dim}", centers[cluster_id][dim], cluster_id)
cluster_sizes = np.bincount(labels)
for cid, size in enumerate(cluster_sizes):
    writer.add_scalar("ClusterSize/Cluster", size, cid)
avg_distances = []
for cid in range(4):
    cluster_points = X[labels == cid]
    distances = np.linalg.norm(cluster_points - centers[cid], axis=1)
    avg_distance = np.mean(distances)
    avg_distances.append(avg_distance)
    writer.add_scalar("ClusterCompactness/AverageDistance", avg_distance, cid)

sil_score = silhouette_score(X, labels)
writer.add_scalar("Quality/SilhouetteScore", sil_score, 0)
writer.close()
with open("cluster_centers.json", "w") as f:
    json.dump(centers.tolist(), f)
```

Рисунок 4.18 – Блок коду кластеризації станів ULEnemyAgent

За таким кодом відбувається створення чотирьох кластерів та аналіз кластеризація за допомогою TensorBoard. За отриманим результатом було також отримано аналіз кластеризації станів битви. Наприклад для чотирьох кластерів станів гри, з котрих 6800, розподілились по 1607, 2501, 696, 1996 відповідно.

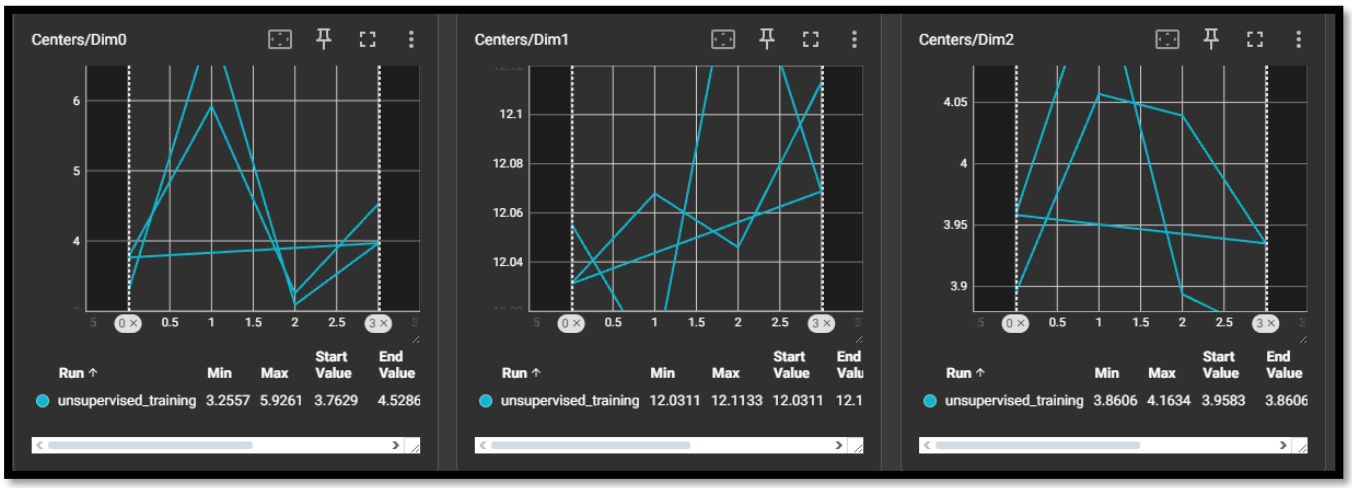

Рисунок 4.19 – Графік розташування цетроїдів ULEnemyAgent

За цим графіком показано приклад розташування трьох близьких центроїдів та за ним відображено, що центроїди розташовані хаотично, але на близькому рівні до ближнього. Також є дані середньої відстані до центроїда, що є різним для кожного кластеру 4.477, 4.152, 4.784, 4.352 та якість кластеризації, що є 0.1849.

Якість кластеризації має поганий результат через велику кількість шуму вибірки, а саме через те, що до вибірки вписують усі карти в руці.

В редакторі Unity створено скрипт з ім’ям UnsupervisedAgent, де і прописано логіку агента. За логікою, агент отримує кластери JSON файлом та перетворює їх у список масива типу Float. Після, агент отримує стан битви та шукає найкращий кластер за станом, коли агент знаходить підходящий кластер, то за його номером виконується тип карти в битві, якщо такої нема на руці, знаходиться інша.

```
float[] GetState(Entity self, Entity target)
{
    List<float> state = new List<float>();
    state.Add(self.CharacterSheet.HPCurrent);
    state.Add(self.CharacterSheet.AC.Value);
    state.Add(cardHand.Count);
    state.Add(turnActions);
    state.Add(target.CharacterSheet.HPCurrent);
    state.Add(target.CharacterSheet.AC.Value);
    for (int i = 0; i < handSize; i++)
    {
        if (i < cardHand.Count)
        {
            state.Add((int)cardHand[i].ActionType);
            state.Add(cardHand[i].Actions);
        }
        else
        {
            state.Add(-1);
            state.Add(-1);
        }
    }
    return state.ToArray();
}
int PredictCluster(float[] state)
{
    int bestCluster = 0;
    float minDist = float.MaxValue;
    foreach (var cluster in clusterCenters.Select((value, index) => new { value, index }))
    {
        float dist = 0;
        for (int i = 0; i < state.Length; i++) dist += (state[i] - cluster.value[i]) * (state[i] - cluster.value[i]);
        dist = Mathf.Sqrt(dist);

        if (dist < minDist)
        {
            minDist = dist;
            bestCluster = cluster.index;
        }
    }
    return bestCluster;
}
```

Рисунок 4.20 – Блок коду UnsupervisedAgent

Інші методи класу схожі з попереднім класом SupervisedAgent заради функціоналу в битві.

У результаті було створено три агента штучного інтелекту за трьома методами, навчання з учителем, навчання без учителя, навчання з підкріпленням. Тепер буде зібрано дані для аналізу їх ефективності в ігровому застосунку для створення системи поведінки неігрового персонажа.

4.4 Тестування застосунку із новоствореною системою поведінки NPC

Тестування новоствореної системи полягає в тому аби зібрати ігрові дані щодо агентів ШІ та протестувати їх роботу в ігровому застосунку, а саме підчас битви із гравцем.

Для збору ігрової інформації буде створено скрипт LearningLogger, де ігрові дані матчу буде збирати скрипт та формувати з даних CSV файл з даними котрий після буде оброблено в віртуальному середовищі Python за допомогою бібліотеки pandas.

Для тестування ігрового застосунку буде створено декілька тест-кейсів де головні чинники для тестування буде інтерфейс вибору агентів ШІ для ігрового застосунку, використання дій в битві проти агентів, завершення ходу без здійснення дій в битві, завершення битви проти агента отримавши перемогу над агентом або отримавши поразку від агента.

Таблиця 4.1 – Тест-кейси системи поведінки NPC

Опис тест-кейсу	**Послідовність виконаних дій**	**Очікуваний результат**	**Результат тестування**
Перевірка вибору типу агента в налаштуваннях	1.В стартовому меню обрати пункт налаштувань; 2.Обрати тип агента; 3.Закрити меню налаштувань.	Тип агента обрано та відображається в редакторі Unity	Виконано
Перевірка використання дій проти агента	1.Почати гру в стартовому меню; 2.Обрати дію проти агента.	Агент сприйняв дію	Виконано

Кінець таблиці 4.1

Перевірка дії агента після завершення ходу гравцем	1.Почати гру в стартовому меню; 2.Обрати дію закінчення ходу.	Агент почав свій хід, виконав дії та завершив хід	Виконано
Перевірка роботи агент після кінця битви	1.Почати гру в стартовому меню; 2.Зіграти серію раундів; 3.Закінчити гру перемогою над агентом або поразкою.	Агент закінчує свою роботу	Виконано

Зі збором інформації для кожного агента було зібрано ігрових даних на тисячу матчів, тобто з кожним агентом окремо було зіграно тисячу матчів та дані цих матчів зібрано в окремі CSV файли.

Зі тестування системи поведінки NPC все вийшло добре, усі тест-кейси були пройдені та робота агентів не зазнала критичних помилок.

4.5 Проведення аналізу ефективності за отриманою статистикою

В цьому підрозділі буде проведено аналіз ефективності застосування агентів штучного інтелекту в застосуванні системи поведінки неігрових персонажів та для початку аналізу треба провести аналіз даних.

Для аналізу даних скористуємось мовою Python та бібліотекою pandas.

З перших критеріїв перевіримо статистику перемог та поразок, для цього збираємо дані з ігрового застосунку за допомогою написаного скрипту та

переводимо статистику в файл розширення CSV, де після за допомогою мови програмування Python обробляємо дані статистики.

Для формування вибірки статистики проведено тисячу битв для кожного агента та до вибірки записано епізоди, скільки раундів пройшло в епізоді, хто переміг. За вибіркою було отримано такі результати.

Таблиця 4.2 – Критерій відсоток перемог, аналіз перемог та поразок агентів

Агент	**Відсоток перемог**	**Середнє прожитих раундів**	**Середнє раундів для перемоги**	**Середнє раундів для поразки**
SL	6,6%(0.066)	5,473	8,924	5,229
UL	5,2%(0.052)	5,348	8,807	5,158
RL	6,7%(0.067)	5,526	9,044	5,273

За такими результатами можна скласти висновок, що агент за методом навчання з підкріпленням, RL, є кращий серед інших у своїх результатах. Однак, найближчий за результатами до нього є агент за методом навчання з учителем.

Низький відсоток перемог не значить, що агенти погано виконують свою задачу, бо вони керують низькорівневим персонажем, який б’ється в команді з іншими, тому на одинці тобто 1vs1 він показує слабкий відсоток перемог, але навіть так, він перемагає.

Також на цій статистиці можна побачити, що якість кластеризації для агента за методом навчання без учителя зіграла свою негативну роль, і хоча він відстає на один відсоток, враховуючи вище описане, це погано.

Для визначення критерію часу навчання агентів є лише точна мітка часу навчання агента за методом RL, а для інших навчання проходило дуже швидко, тому замірити це буде складно, однак до навчання також входить збір даних для навчання, 629 секунд.

Тому за висновком вийде, що агент за методом SL навчався 629~ секунд, так само навчався агент і за методом UL, але агент за методом RL навчався 2609 секунд.

За критерієм використаної пам’яті для навчання агента є дві моделі агентів та набір кластерів у форматі JSON.

Розмір моделі агента SL є 76816 байтів, тобто 76,8 кілобайта.

Розмір набору кластерів агента UL 1130 байтів, тобто 1,13 кілобайта.

Розмір набору кластерів агента RL 80315 байтів, тобто 80,3 кілобайта.

За критерієм складність реалізації треба визначити оцінку не ординарно. Кожен метод для створення агента було легко реалізувати, але агента за методом навчання з підкріпленням було легше, в плані було достатньо документацій, гайдів для налаштування, створення та тренування, легко інтегрується в пакеті ресурсів ML-Agents та налаштовується в редакторі Unity, не має потреби виконувати методи в віртуальному середовищі Python, бо інструмент вже все зробив.

З агентом за методом навчання з учителем було складніше, так як не всі методи шарування ONNX моделі підтримується в Unity Barracuda. Через це було вимушено використати саме методи, що підтримуються Barracuda.

З агентом за методом навчання без учителя було легше ніж з попереднім, для створення кластерів станів битви в ігровому застосунку не було проблем, лише написання коду та використання потрібних бібліотек для реалізації.

За таким описом я поставив оцінки від 5 до 1 за складністю реалізації, де 5 є дуже складно та 1 є дуже легко. Таким чином фінальна таблиця виглядатиме так.

Таблиця 4.3 – Статистика за всіма критеріями

Агент	**Відсоток перемог(%)**	**Час навчання(s)**	**Пам’яті використано(Kb)**	**Складність реалізації(5…1)**
SL	6,6%	629~ sec	76,8 Kb	4
UL	5,2%	629~ sec	1,13 Kb	2
RL	6,7%	2609 sec	80,3 Kb	1

За цією статистикою можна відокремити ефективність методу навчання з підкріпленням, цей метод має переваги в реалізації методу в рушію Unity, має

гарний відсоток перемог зважаючи на нерівний бій, але має довгий процес навчання та використовує більше пам’яті ніж інші методи.

Метод навчання з учителем є ближче до методу навчання з підкріпленням, як по відсотках перемог так і по використанню пам’яті, але має швидший процес навчання і цей метод складно реалізувати через обмеженість шарування моделей ONNX для Unity Barracuda. Однак в пакеті ресурсів Unity ML-Agents бажають додавати нові методи навчання агентів, тому з часом складність реалізації може бути облегшеною.

Метод навчання без учителя має низький відсоток перемог, схожий за часом навчання з методом навчання з учителем і займає дуже мало пам’яті, бо це не створення самої моделі, а створення набору кластерів. За складністю реалізації метод не є складно реалізувати в рушію Unity.

Методи машинного навчання є ефективними у створенні ігрових застосунків. В цьому застосунку їх використання було у створенні системи поведінки неігрових персонажів і ефективність застосування саме такої методики створення системи є кращим за попередню методику створення системи поведінки персонажів. Минула методика мала на меті створенні великої кількості коду із відображенням станів та кожної реакції на той стан. Це могло бути декілька файлів скрипту для різних персонажів, але за новою методикою модель навченого агента можна передати іншому персонажу.

Для інших систем ігрового застосунку методи машинного навчання можуть бути навіть дуже ефективними у застосуванні, наприклад створення агенту для імітації води в середовищі для урахування фізики середовища та його взаємодії з іншими об’єктами та станами.

Висновки до розділу 4

В цьому розділі описано конфігурацію середовища та інструментів потрібних для роботи із створенням та навчання агентів штучного інтелекту за трьома

методами машинного навчання, окрім опису конфігурації тут описано створення агентів та їх навчання.

Підчас навчання було зібрано аналітичні дані навчання самих агентів та ці дані було описано в графіках після навчання агентів.

Після створення та навчання агентів проведено аналіз ефективності усіх трьох агентів, тобто зіграно понад тисячу епізодів для кожного агента, за зібраними результатами визначено відсоток перемог із зіграних епізодів, середню кількість раундів потрібних для перемоги, середню кількість для поразки. Також зібрано інформацію про час навчання кожного з агентів, їх кількість використаної пам’яті для моделі та їх складність реалізації у рушію Unity.

В кінці підведено висновки щодо ефективності використання методів машинного навчання у застосуванні їх для систем ігрового застосунку.

ВИСНОВКИ

Кваліфікаційна робота присвячена дослідженню ефективності використання методів машинного навчання у створенні штучного інтелекту для ігрового застосунку.

І дослідження полягало в ефективності методів машинного навчання у тій чи інший системі ігрового застосунку та загальна ефективність методів машинного навчання у всьому ігровому застосунку.

За проведеним дослідженням в ігровому застосунку, агент за методом навчання з підкріпленням є найкращим варіантом для типу поведінки неігрового персонажа в обраному ігровому застосунку, його результати були найкращі серед інших, хоча й має певні недоліки типу великого часу навчання та кількості використаної пам’яті для навчання.

Ефективність методів машинного навчання у застосуванні в системах ігрового застосунку є великою, бо за допомогою них можна створювати, як складні системи, котрі було б складно реалізувати без агентів ШІ так і легкі системи для облегшення роботи або покращення роботи системи.

Також в результаті дослідження було модифіковано ігровий застосунок новою системою поведінки неігрового персонажа, що підвищило ефективність противників в битві проти гравця та підвищило технічну складову і технічну новизну ігрового застосунку. І тепер є можливість створення нових агентів штучного інтелекту в ігровому застосунку для реалізації нових противників та можливості перенавчання вже існуючих під новий функціонал, правила, стани гри.

Окрім застосування найкращого типу агента для поведінки неігрових персонажів, інші агенти можна використати в системах, де вони більше підходять за логікою, наприклад генерація контенту для агента за методом навчання без учителя, або система прокладання шляху, Pathfinding за допомогою навчання агента з учителем.

В процесі виконання кваліфікаційної роботи було вирішено наступні завдання:

- виконано аналіз методів машинного навчання;
- виконано аналіз ігрового застосунку;
- виконано порівняльний аналіз методів машинного навчання;
- сформулювано вимоги до агентів ШІ, що розробляється;
- виконано проєктування та моделювання агентів ШІ;
- реалізовано три види агентів ШІ для ігрового застосунку;
- протестовано розроблений ШІ агентів для ігрового застосунку;
- виконано аналіз витрачених ресурсів та ефективність роботи ШІ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Oxford A Dictionary of Computing : Abbreviations URL: https://www.oxfordreference.com/display/10.1093/acref/9780199234004.001.0001/acref-9780199234004 (Last accessed: 19.05.2025).

2. Buckland M. Programming Game AI by Example. Jones & Bartlett, 2004, 495 p.

3. Millington I., Funge J. Artificial Intelligence for Games. CRC Press, 2009, 869 p.

4. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Pearson, 2020, 1136 p.

5. Yannakakis G. N., Togelius J. Artificial Intelligence and Games. Springer, 2018, 360 p.

6. Tkachenko O., Mamaiev A. Artificial Intelligence Usage in Forming Computer Games Space. Штучний інтелект; генеративний штучний інтелект; нейронні мережі; комп'ютерні ігри; жанри; віртуальна реальність; game-індустрія; простір комп'ютерних ігор. 2024. 7(1):97-109 DOI:10.31866/2617-796X.7.1.2024.307012.

7. Які сфери змінюватиме штучний інтелект у найближчому майбутньому. KyivstarHub: вебсайт. URL: https://hub.kyivstar.ua/articles/yaki-sfery-zminyuvatyme-shtuchnyj-intelekt-u-najblyzhchomu-majbutnomu (Дата звернення: 19.05.2025).

8. How AI is revolutionizing game development. AllStarsIt: вебсайт. URL: https://www.allstarsit.com/blog/how-ai-is-revolutionizing-game-development (Дата звернення: 19.05.2025).

9. What Type of AI is Used in Games? OpenGrowth: вебсайт. URL: https://www.blogs.opengrowth.com/what-type-of-ai-is-used-in-games (Дата звернення: 19.05.2025).

10. Make Your Convai-Powered AI Characters Act on Your Command in Unity. Convai: вебсайт. URL: https://convai.com/blog/adding-actions-for-ai-characters-in-unity (Дата звернення: 19.05.2025).

11. Counter-Strike. Valve Developer Community: вебсайт. URL: https://developer.valvesoftware.com/wiki/Counter-Strike (Дата звернення: 19.05.2025).

12. Hutson J., Ratican J. Adaptive Worlds: Generative AI in Game Design and Future of Gaming, and Interactive Media. Generative AI, Game design, Real-time content generation, Interactive media, Personalized gaming experiences. 2024. DOI:10.5281/zenodo.13894497.

13. Revisiting the AI of Alien: Isolation. Game developer: вебсайт. URL: https://www.gamedeveloper.com/design/revisiting-the-ai-of-alien-isolation (Дата звернення: 19.05.2025).

14. Unity Documentation: вебсайт. URL: https://docs.unity3d.com/Manual/index.html (Дата звернення: 19.05.2025).

15. Levkivskyi V., Marchuk G., Moskalyk O. Game artificial intelligence model for non-playable characters. Штучний інтелект; гра; некерований гравцем персонаж; машинне навчання; модель; навчання з підкріпленням; NPC. 2024. DOI:10.25140/2411-5363-2024-4(38)-163-171.

16. Microsoft Visual Studio. Microsoft: вебсайт. URL: https://visualstudio.microsoft.com (Дата звернення: 19.05.2025).

17. Visual Studio Code: вебсайт. URL: https://code.visualstudio.com (Дата звернення: 19.05.2025).

18. JetBrains Rider. JetBrains: вебсайт. URL: https://www.jetbrains.com/rider (Дата звернення: 19.05.2025).

19. An Overview of Reinforcement Learning: Teaching Machines to Play Games. GameDevAcademy: вебсайт. URL: https://gamedevacademy.org/an-overview-of-reinforcement-learning-teaching-machines-to-play-games/ (Дата звернення: 19.05.2025).

20. Background: Machine Learning. Unity: вебсайт. URL: https://docs.unity3d.com/Packages/com.unity.ml-agents@4.0//manual/Background-Machine-Learning.html (Дата звернення: 19.05.2025).

21. ISO/IEC 42001:2023. Artificial Intelligence Management System (AIMS). 1th Ed. 2023. 51 p.

22. ISO/IEC TR 24028:2020. Information technology. Artificial intelligence. Overview of trustworthiness in artificial intelligence. 1th Ed. 2020. 43 p.

23. Troelsen A. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 2022, 1705 p.

24. Develop C and C++ applications. Microsoft: вебсайт. URL: https://visualstudio.microsoft.com/vs/features/cplusplus (Дата звернення: 19.05.2025).

25. Python: вебсайт. URL: https://www.python.org (Дата звернення: 19.05.2025).

26. YAML: вебсайт. URL: https://yaml.org (Дата звернення: 19.05.2025).

27. Introducing JSON. JSON: вебсайт. URL: https://www.json.org/json-en.html (Дата звернення: 19.05.2025).

28. Unity-Technologies/ml-agents/docs/Installation. Github: вебсайт. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_22_docs/docs/Installation.md (Дата звернення: 19.05.2025).

29. Anaconda.org. Anaconda: вебсайт. URL: https://anaconda.org (Дата звернення: 19.05.2025).

30. Unity: A General Platform for Intelligent Agents. 2018 / Juliani A. et al. DOI: 10.48550/arXiv.1809.02627.

ДОДАТОК А

Class LearningLoggerData

```
public class LearningLoggerData : MonoBehaviour
{
    private string path = "analyze_dataset.csv";

    public void Start()
    {
        string line = "Episode,Round,Winner";
        File.WriteAllText(path, line + "\n");
    }

    public void Log(int episode, int round, string winner)
    {
        string line = $"{episode},{round},{winner}";
        File.AppendAllText(path, line + "\n");
    }
}
```

ДОДАТОК Б

Class ReinforcementAgent

```
public class ReinforcementAgent : Agent, IBaseAI
{
    public Entity self;
    public Entity target;
    public LearningSceneManager learningSceneManager;
    private List<CardData> cardDataDeck = new List<CardData>();
    private List<CardData> cardDataDiscard = new List<CardData>();
    public List<CardData> cardHand = new List<CardData>();
    public int handSize = 5;
    public int turnActions = 3;
    private int lastknownHP;
    public override void Initialize()
    {
        learningSceneManager.episode = 0;
        learningSceneManager.reward = 0f;
    }
    public override void OnEpisodeBegin()
    {
        learningSceneManager.episode++;
        learningSceneManager.reward = 0f;
        turnActions = 3;
        lastknownHP = self.CharacterSheet.HPCurrent;
        SetDeck(self.CharacterSheet.Cards);
        FillHandArea();
    }
    public override void CollectObservations(VectorSensor sensor)
```

```
{
    sensor.AddObservation(self.CharacterSheet.HPCurrent);
    sensor.AddObservation(self.CharacterSheet.AC.Value);
    sensor.AddObservation(cardHand.Count);
    sensor.AddObservation(turnActions);
    sensor.AddObservation(target.CharacterSheet.HPCurrent);
    sensor.AddObservation(target.CharacterSheet.AC.Value);
    for (int i = 0; i < handSize; i++)
    {
        if (i < cardHand.Count)
        {
            sensor.AddObservation((int)cardHand[i].ActionType);
            sensor.AddObservation(cardHand[i].Actions);
        }
        else
        {
            sensor.AddObservation(-1);
            sensor.AddObservation(-1);
        }
    }
}
public override void OnActionReceived(ActionBuffers actions)
{
    learningSceneManager.reward = GetCumulativeReward();
    int cardIndex = actions.DiscreteActions[0];
    if (cardIndex >= cardHand.Count)
    {
        AddReward(-0.1f);
        EndTurn();
```

```
            return;
        }
        CardData selectedCard = cardHand[cardIndex];
        if (selectedCard.Actions > turnActions)
        {
            AddReward(-0.2f);
            EndTurn();
            return;
        }
        int lastTargetHP = target.CharacterSheet.HPCurrent;
        selectedCard.UseCard(self, target);
        turnActions -= selectedCard.Actions;
        DropCardToDiscard(selectedCard);
        if (target.CharacterSheet.HPCurrent < lastTargetHP)
        {
            AddReward(0.5f);
        }
        if (turnActions <= 0 || cardHand.Count == 0)
        {
            EndTurn();
            return;
        }
        RequestDecision();
    }
    public void OnAgentDamage()
    {
        if (self.CharacterSheet.HPCurrent < lastknownHP)
        {
            AddReward(-0.5f);
```

```
            return;
        }
        AddReward(0.3f);
    }
    public void OnAgentDeath()
    {
        AddReward(-2.0f);
        learningSceneManager.reward = GetCumulativeReward();
        EndEpisode();
    }
    public void OnPlayerDeath()
    {
        AddReward(2.0f);
        learningSceneManager.reward = GetCumulativeReward();
        EndEpisode();
    }
    private void EndTurn()
    {
        lastknownHP = self.CharacterSheet.HPCurrent;
        learningSceneManager.reward = GetCumulativeReward();
        turnActions = 3;
        FillHandArea();
        self.isTurnEnded = true;
    }
    public void AIAct(Entity target)
    {
        RequestDecision();
    }
}
```

ДОДАТОК В

Class SupervisedAgent

```
public class SupervisedAgent : MonoBehaviour, IBaseAI
{
    public NNModel modelAsset;
    private Model model;
    private IWorker worker;
    public Entity self;
    public Entity target;
    public LearningSceneManager learningSceneManager;
    private List<CardData> cardDataDeck = new List<CardData>();
    private List<CardData> cardDataDiscard = new List<CardData>();
    public List<CardData> cardHand = new List<CardData>();
    public int handSize = 5;
    public int turnActions = 3;
    private List<float> observations;

    void Start()
    {
        model = ModelLoader.Load(modelAsset);
        worker = WorkerFactory.CreateWorker(WorkerFactory.Type.Auto, model);
    }
    public void OnEpisodeStart()
    {
        SetDeck(self.CharacterSheet.Cards);
        FillHandArea();
        learningSceneManager.episode++;
    }
```

```
public void CollectObservations()
{
    observations = new List<float>
    {
        self.CharacterSheet.HPCurrent,
        self.CharacterSheet.AC.Value,
        cardHand.Count,
        turnActions,
        target.CharacterSheet.HPCurrent,
        target.CharacterSheet.AC.Value
    };
    for (int i = 0; i < handSize; i++)
    {
        if (i < cardHand.Count)
        {
            observations.Add((int)cardHand[i].ActionType);
            observations.Add(cardHand[i].Actions);
        }
        else
        {
            observations.Add(-1);
            observations.Add(-1);
        }
    }
}
public int PredictAction()
{
    float[] observsArr = observations.ToArray();
    using (var tensor = new Tensor(1, observsArr.Length, observsArr))
```

```
            {
                worker.Execute(tensor);
                var output = worker.PeekOutput("output");
                int action = output.ArgMax()[0];
                return action;
            }
        }
        public void AIAct(Entity target)
        {
            CollectObservations();
            int actionType = PredictAction();
            CardData selectedCard = cardHand.Find(c => c.ActionType ==
(ActionType)actionType);
            if (selectedCard == null)
            {
                Debug.Log("No Card");
                selectedCard = cardHand.Find(c => c.Actions <= turnActions);
                if(selectedCard == null)
                {
                    Debug.Log("Error No Actions Decision");
                    EndTurn();
                }
            }
            selectedCard.UseCard(self, target);
            turnActions -= selectedCard.Actions;
            DropCardToDiscard(selectedCard);
            if (turnActions <= 0 || cardHand.Count == 0)
            {
                EndTurn();
```

```
            return;
        }
        AIAct(target);
    }
    private void EndTurn()
    {
        turnActions = 3;
        FillHandArea();
        self.isTurnEnded = true;
    }
}
```

ДОДАТОК Г

Class UnsupervisedAgent

```
public class UnsupervisedAgent : MonoBehaviour, IBaseAI
{
    public Entity self;
    public Entity target;
    public LearningSceneManager learningSceneManager;
    private List<CardData> cardDataDeck = new List<CardData>();
    private List<CardData> cardDataDiscard = new List<CardData>();
    public List<CardData> cardHand = new List<CardData>();
    public int handSize = 5;
    public int turnActions = 3;
    private List<float[]> clusterCenters = new List<float[]>();

    void Start()
    {
        LoadClusters("Assets/AI Models/cluster_centers_UL.json");
    }
    public void OnEpisodeStart()
    {
        SetDeck(self.CharacterSheet.Cards);
        FillHandArea();
        learningSceneManager.episode++;
    }
    void LoadClusters(string path)
    {
        string json = File.ReadAllText(path);
        clusterCenters = JsonConvert.DeserializeObject<List<float[]>>(json);
```

```
}
float[] GetState(Entity self, Entity target)
{
   List<float> state = new List<float>();
   state.Add(self.CharacterSheet.HPCurrent);
   state.Add(self.CharacterSheet.AC.Value);
   state.Add(cardHand.Count);
   state.Add(turnActions);
   state.Add(target.CharacterSheet.HPCurrent);
   state.Add(target.CharacterSheet.AC.Value);
   for (int i = 0; i < handSize; i++)
   {
      if (i < cardHand.Count)
      {
         state.Add((int)cardHand[i].ActionType);
         state.Add(cardHand[i].Actions);
      }
      else
      {
         state.Add(-1);
         state.Add(-1);
      }
   }
   return state.ToArray();
}
int PredictCluster(float[] state)
{
   int bestCluster = 0;
   float minDist = float.MaxValue;
```

```
            foreach (var cluster in clusterCenters.Select((value, index) => new { value,
index }))
            {
                float dist = 0;
                for (int i = 0; i < state.Length; i++) dist += (state[i] - cluster.value[i]) *
(state[i] - cluster.value[i]);
                dist = Mathf.Sqrt(dist);
                if (dist < minDist)
                {
                    minDist = dist;
                    bestCluster = cluster.index;
                }
            }
            return bestCluster;
        }
        public void AIAct(Entity target)
        {
            float[] state = GetState(self, target);
            int actionType = PredictCluster(state);
            CardData selectedCard = cardHand.Find(c => c.ActionType ==
(ActionType)actionType);
            Debug.Log(actionType);
            if (selectedCard == null)
            {
                Debug.Log("No Card");
                selectedCard = cardHand.Find(c => c.Actions <= turnActions);
                if (selectedCard == null)
                {
                    Debug.Log("Error No Actions Decision");
```

```
                EndTurn();
            }
        }
        selectedCard.UseCard(self, target);
        turnActions -= selectedCard.Actions;
        DropCardToDiscard(selectedCard);
        if (turnActions <= 0 || cardHand.Count == 0)
        {
            EndTurn();
            return;
        }
        AIAct(target);
    }
    private void EndTurn()
    {
        turnActions = 3;
        FillHandArea();
        self.isTurnEnded = true;
    }
}
```

ДОДАТОК Д

Апробація кваліфікаційної роботи

Результати досліджень були представлені на конференції:

– Могилянські читання – 2025, : Моделі, методи та засоби програмної інженерії : XXVIII Всеукр. наук.-практ. конф. : тези доповідей : Технічні науки, Миколаїв, 12 листоп. 2025 р. / ЧНУ ім. Петра Могили. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2025.

Могилянські читання 2025

Секція «Технічні науки»

Підсекція «Моделі, методи та засоби програмної інженерії»

12 листопада, 15:30, https://meet.google.com/efi-dfnu-cwk

Керівник: канд. техн. наук, доцент Давиденко Євген Олександрович

Секретар: канд. техн. наук, доцент Горбань Гліб Валентинович

Мета проведення: Створення сприятливих умов для об'єднання та реалізації творчих здібностей науковців різних поколінь, залучення їх до активної науково-дослідної, пошукової діяльності у вирішенні сучасних проблем в інформаційних технологіях та програмній інженерії.

1. ***Somriakov B. O.*** (applicant for the first level of higher education (bachelor's degree) of Petro Mohyla Black Sea National University, Mykolaiv, Ukraine), ***Borovlyova S. Yu.*** (senior lecturer at the Department of Software Engineering of Petro Mohyla Black Sea National University, Mykolaiv, Ukraine). **Preference function models for comparative analysis of decisions in areas of uncertainty.**
2. ***Somriakov B. O.*** (applicant for the first level of higher education (bachelor's degree), Petro Mohyla Black Sea National University, Mykolaiv, Ukraine), ***Ralenko V. S.*** (lecturer at the Department of Software Engineering of Petro Mohyla Black Sea National University, Mykolaiv, Ukraine). **Geometric perspectives on word embeddings and semantic similarity: implementation and analysis.**
3. ***Антипова К. О.*** (PhD, доцентка (б. в. з.) кафедри інженерії програмного забезпечення, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). **Розпізнавання контенту, згенерованого авторегресійними та мультимодальними моделями.**
4. ***Безрядіна С. Є.*** (здобувачка першого (бакалаврського) рівня вищої освіти, Чорноморський національний університет імені Петра Могили, м. Миколаїв, Україна), ***Боровльова С. Ю.*** (старша викладачка кафедри інженерії програмного забезпечення, Чорноморський національний

Рисунок 1 – Програма Могилянських читань 2025р.

13. ***Кубицький М. С.*** (здобувач другого (магістерського) рівня вищої освіти, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна)**,** ***Горбань Г. В.*** (канд. техн. наук, доцент, доцент кафедри інженерії програмного забезпечення, Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна). **Аналіз ефективності використання методів машинного навчання у створенні штучного інтелекту для ігрового застосунку.**

Рисунок 2 – Подана заявка на конференцію

ФОРМА ЗАЯВКИ НА УЧАСТЬ КОНФЕРЕНЦІЇ

Прізвище	Кубицький
Ім'я	Микита
По батькові	Сергійович
Науковий ступінь, вчене звання	-
Посада	-
Науковий керівник (якщо автор тез-студент)	Горбань Гліб Валентинович
Телефон	0662082130
E-mail	nikita.kubitskiy@gmail.com
Назва доповіді/тез	Аналіз ефективності використання методів машинного навчання у створенні штучного інтелекту для ігрового застосунку
Тематика форуму(секція)	Комп'ютерні технології

Рисунок 3 – Форма заявки

УДК

Кубицький М. С.,
здобувач другого (магістерського) рівня вищої освіти,
ЧНУ імені Петра Могили, Миколаїв, Україна

АНАЛІЗ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ МЕТОДІВ МАШИННОГО НАВЧАННЯ У СТВОРЕННІ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІГРОВОГО ЗАСТОСУНКУ

В сучасному світі штучний інтелект має багато способів використання та займає певне місце серед побуту людини. Іноді навіть складно уявити, як вижити без агента помічника котрий як енциклопедія готовий розказати тобі те, чого ти зажадав. І для створення подібного або більш схожого агента існують інструменти, методи, ці методи називаються методи машинного навчання(ML).

Існує три основні методи машинного навчання: навчання з учителем; навчання без учителя; навчання з підкріпленням. Кожний метод має свою логіку надавання інформації та безпосередньо логіку навчання за даними. Ці методи використовують під певні задачі іноді комбінуючи з іншими методами. З розвитком технологій актуальність використання ML агентів стає більш значною, бо агенти залучені в різних сферах життя, наприклад автономні автомобілі Tesla, автоматичне налаштування продуктивності в ноутбуках Legion.

В розробці ігрових застосунків є актуально використання штучного інтелекту в різних елементах застосунку та таким чином створюються різні системи за призначенням, існує система, що налаштовують анімації без ручних налаштувань імітуючи певні характеристики, система, що імітує фізику води в середовищі під різними ефектами та станами, система оптимізації ресурсів застосунку. Якщо більш загально, то агентів ML у розробці ігрового застосунку використовують у:

- генерації контенту – створення текстур, моделей, рівнів, середовищ, наприклад процедурна генерація;
- тестування та контроль якості – використання агентів для проходження рівнів або проходження тест сценаріїв, балансування ігрового процесу, оптимізація ресурсів застосунку;
- локалізація – автоматичний переклад, локалізація мов;
- аналіз гравців – персоналізація контенту, аналіз поведінки гравців, збір даних гравців, прогнозування тенденцій;
- розробка ігрових механік – фізика елементів ігрового середовища, поведінка NPC, процедурна анімацій;
- генеративний контент – генерація сцен, сюжету, діалогів, завдань.

Хоча й ці системи є по своєму важливі, однак провести аналіз кожного методу для цих систем не зовсім можливо, бо кожна система вимагає своєї логіки навчання та взаємодії з даними. Однак є ланка в котрій є можливість використати кожен з методів та проаналізувати кожен у цій ланці ігрового застосунку і це є поведінка NPC.

Застосування методів машинного навчання для створення агентів поведінки неігрового персонажа є універсальним, бо сама поведінка не має певних правил з середовища або функціоналу, ці правила створює сам розробник, тому логіка навчання та отримання даних буде підпорядкована самим розробником.[1]

Рисунок 4 – Тези подані на апробацію