

Національний університет "Львівська політехніка"

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Спеціальність 122 «Комп'ютерні науки»



ARTIFICIAL  
INTELLIGENCE

## ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської кваліфікаційної роботи на тему:

**«Аналіз методів машинного навчання для керування неігровими персонажами у грі виживання»**

Студента групи

КНСШ-23

Задерецького В. А.

Керівник роботи

\_\_\_\_\_

(підпис)

( Шиманський В. М. )

Консультанти

\_\_\_\_\_

(підпис)

( )

\_\_\_\_\_

(підпис)

( )

Завідувач  
кафедри СШ

\_\_\_\_\_

(підпис)

( Мельникова Н. І. )

Львів - 2025

Національний університет “Львівська політехніка”  
Інститут комп’ютерних наук та інформаційних технологій  
Кафедра систем штучного інтелекту  
Спеціальність 122 «Комп’ютерні науки»

**ЗАТВЕРДЖУЮ:**

Завідувач кафедри СШІ \_\_\_\_\_

“ \_\_\_\_ ” \_\_\_\_\_ 2025р.

**ЗАВДАННЯ**  
**НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ**

Задерецькому Володимирі Андрійовичу

1. Тема роботи: «Аналіз методів машинного навчання для керування неігровими персонажами у грі виживання» затверджена наказом університету № 11140-4-06 від 31.03.2025
2. Термін подання закінченої роботи: \_\_\_\_\_ 06.06.2025 \_\_\_\_\_.
3. Вихідні дані до (проекту) роботи: вхідними даними до проекту є набори даних записаний під час багатьох симуляційних ігор.
4. Зміст розрахунково-пояснювальної записки: (перелік питань, що належить розробити): аналітичний огляд літературних та інших джерел, системний аналіз та обґрунтування проблеми, методи та засоби вирішення проблеми, практична реалізація.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): схема PRISMA, діаграма архітектури нейронної мережі, діаграма класів, Confusion Matrix.

**6. Консультанти по роботі, із зазначенням розділів проекту, що стосується їх**

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Консультант з іноземної мови			

7. Дата видачі завдання 10.03.2023

Керівник роботи \_\_\_\_\_  
(підпис)

Завдання прийняв до виконання \_\_\_\_\_  
(підпис)

**КАЛЕНДАРНИЙ ПЛАН**

№ п/п	Назва етапів дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури	10.03.2023 – 03.04.2023	
2	Створення архітектури	03.04.2023 – 15.05.2023	
3	Розробка алгоритмічного та програмного забезпечення	15.05.2023 – 10.11.2023	
4	Експериментальні дослідження	11.09.2023 – 15.11.2023	
5	Оформлення записки	06.05.2025 – 06.06.2025	

Керівник роботи \_\_\_\_\_  
(підпис)

Студент \_\_\_\_\_  
(підпис)

## АНОТАЦІЯ

Магістерська кваліфікаційна робота виконана студентом групи КНСШ-23 Задерецьким Володимиром Андрійовичем. Тема роботи — “Аналіз методів машинного навчання для керування неігровими персонажами у грі виживання”. Дослідження проведено на здобуття ступеня магістра за спеціальністю 122 «Комп’ютерні науки».

У роботі досліджено методи машинного навчання, які можуть бути ефективно застосовані для створення адаптивної, гнучкої та реалістичної поведінки неігрових персонажів (NPC) у відеоіграх жанру виживання. Об’єктом дослідження є процеси керування NPC у віртуальному бойовому середовищі. Предмет дослідження — алгоритми машинного навчання, які дозволяють агентам аналізувати ігрову ситуацію, приймати оптимальні рішення та адаптувати свою поведінку у динамічних умовах.

Метою роботи є створення ефективних моделей керування NPC з використанням методів машинного навчання та порівняння їх продуктивності. Для досягнення мети було реалізовано чотири підходи: багат шаровий перцептрон (MLP), Q-learning, гібридну модель, яка поєднує MLP і Q-learning, а також Random Forest. Особливу увагу приділено гібридному підходу, який дозволяє поєднати точність моделі з навчання з учителем із гнучкістю моделі навчання з підкріпленням.

У процесі реалізації було сформовано власний ігровий датасет на основі сценаріїв боїв між двома командами агентів у середовищі Unity. Датасет включає як базові, так і агреговані ознаки, що дозволяють детальніше описати тактичну ситуацію навколо NPC. На основі цього датасету було натреновано моделі, проведено їхню валідацію, а також порівняння результатів за метриками точності та середньої винагороди.

Для реалізації моделей використовувалися сучасні інструменти машинного навчання, включаючи бібліотеки Python (PyTorch, sklearn, pandas) та середовища розробки (Google Colab, Jupyter Notebook). Проведено комплексне

тестування моделей з урахуванням різних сценаріїв ігрової взаємодії. Зокрема, гібридна модель показала найвищі результати серед усіх реалізованих підходів, продемонструвавши високу точність класифікації дій та покращену динаміку середньої винагороди агента в грі.

У результаті виконання дипломної роботи було реалізовано ефективні алгоритми для автоматизованого керування неігровими персонажами, які можуть бути використані в ігровій індустрії. Робота закладає підґрунтя для подальших досліджень у сфері адаптивних AI-агентів та інтеграції навчання з підкріпленням у реалістичні симуляції. Отримані результати підтверджують ефективність комбінованих підходів та їхню здатність підвищити реалізм ігрового процесу.

Загальний обсяг роботи: 74 сторінок, 21 рисунок, 22 посилання.

**Ключові слова:** машинне навчання, нейронна мережа, MLP, Q-learning, Random Forest, гібридна модель, неігрові персонажі, гра виживання.

## ABSTRACT

Master's degree work of the student of the group CSAI-23 Zaderevskyi Volodymyr Andriiovych, a student of group KNSH-23, on the topic: "Analysis of Machine Learning Methods for Controlling Non-Playable Characters in a Survival Game." The study was conducted in pursuit of a master's degree in the specialty 122 "Computer Science."

This work explores machine learning techniques that can be effectively applied to create adaptive, flexible, and realistic behavior of non-playable characters (NPCs) in survival game environments. The object of the study is the control process of NPCs in a virtual combat simulation. The subject of the study is machine learning algorithms that enable agents to analyze the game state, make optimal decisions, and adapt their behavior to dynamic in-game conditions.

The aim of the research is to develop efficient control models for NPCs using machine learning methods and compare their performance. Four approaches were implemented: multilayer perceptron (MLP), Q-learning, a hybrid model combining MLP and Q-learning, and Random Forest. Special emphasis was placed on the hybrid approach, which leverages the accuracy of supervised learning with the adaptability of reinforcement learning.

During implementation, a custom dataset was generated based on combat scenarios between two teams of agents in a Unity simulation environment. The dataset included both raw and aggregated features to better describe the tactical context around the NPC. Based on this dataset, the models were trained, validated, and evaluated in terms of classification accuracy and average reward metrics.

Modern tools and libraries for machine learning in Python (PyTorch, sklearn, pandas) were used for model development, and environments such as Google Colab, Jupyter Notebook, and PyCharm facilitated experimentation and visualization. Thorough testing was carried out under various simulated conditions. The hybrid model achieved the best results among all tested approaches, showing the highest classification accuracy and significantly improved agent performance in the game.

As a result of this work, efficient algorithms were developed for automated NPC

control, which can be applied in the game industry. This research lays the foundation for future studies in adaptive AI agents and the integration of reinforcement learning into realistic simulations. The results confirm the effectiveness of combined learning strategies and their ability to enhance the realism and quality of gameplay.

Total volume of the thesis: 74 pages, 21 figures, 22 references.

**Keywords:** machine learning, neural network, Q-learning, Random Forest, hybrid model, non-playable characters, survival game.

## ЗМІСТ

АНОТАЦІЯ.....	4
ABSTRACT.....	6
ВСТУП .....	9
1. АНАЛІТИЧНИЙ РОЗДІЛ .....	12
1.1. Опис предметного середовища.....	12
1.2. Огляд наявних аналогів.....	13
1.3. Постановка задачі .....	24
1.4. Висновки до розділу .....	25
2. ДОСЛІДНИЦЬКИЙ РОЗДІЛ.....	26
2.1. Вхідні дані .....	26
2.2. Вибір методів машинного навчання.....	36
2.3. Опис нейронних мереж для керування NPC.....	38
2.3.1 Архітектура MLP .....	38
2.3.2 Q-Learning .....	41
2.3.3 Random forest .....	43
2.4. Дослідження впливу різних факторів на роботу моделей.....	44
2.5. Висновки до розділу .....	46
3. РОЗДІЛ АПРОБАЦІЙ ТА РЕЗУЛЬТАТІВ .....	47
3.1. Засоби реалізації .....	47
3.2. RL-Agent .....	51
3.3. Опис процесу реалізації .....	53
3.4. Отримані результати .....	56
3.5. Вимоги до технічного та програмного забезпечення .....	64
3.6. Висновки до розділу .....	65
ВИСНОВКИ .....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	70
Додаток А. Результати .....	73



## ВСТУП

Штучний інтелект є однією з найбільш захоплюючих та інноваційних галузей сучасної технології. Його застосування в різних сферах життя постійно розширюється, а однією з таких сфер є галузь комп'ютерних ігор. За останні десятиліття штучний інтелект відіграв значну роль у розвитку інтерактивних та реалістичних ігрових досвідів.

Загалом, застосування штучного інтелекту в іграх дозволяє створити інтелектуальних ворогів, реалістичну поведінку персонажів та забезпечити досить цікавий геймплей для гравців. ШІ в іграх, безпосередньо, відповідає за прийняття рішень, планування стратегій, адаптацію до геймплейних ситуацій та взаємодію з гравцями. Він забезпечує інтелектуальну реакцію на дії гравця різного типу, надаючи відчуття реалістичності та викликів під час процесу проходження гри.

Застосування штучного інтелекту в ігровій розробці є однією з найцікавіших та інноваційних галузей, яка надає можливість створювати захоплюючі та реалістичні ігрові досвіди. ШІ дозволяє розробникам створювати інтелектуальних персонажів, розумні системи поведінки та складні геймплейні сценарії, що роблять ігри більш цікавими та викликають більш глибоке занурення гравців у віртуальний світ.

Одним з найпопулярніших застосувань ШІ в іграх є створення системи штучного інтелекту для управління неігровими персонажами (Non-Player Characters, NPC). Це дозволяє створювати різноманітних ворогів або союзників зі складною поведінкою, які можуть адаптуватися до стратегії гравця та забезпечувати цікавий та викликовий геймплей.

Unity, один з найпопулярніших інструментів для розробки ігор, надає можливості для використання штучного інтелекту. Завдяки широкому набору функцій та інструментів, Unity дозволяє розробникам легко впроваджувати інтелектуальні системи в свої ігри. Від визначення поведінки персонажів до створення складних логічних сценаріїв, Unity надає зручні інструменти для

реалізації штучного інтелекту в ігровому середовищі.

Для інтеграції нейронних мереж в Unity використовується плагін ML-Agents. ML-Agents (Machine Learning Agents) - це відкритий інструментарій від Unity Technologies, який дозволяє розробникам інтегрувати машинне навчання і штучний інтелект (AI) у свої відеоігри та симуляційні проекти, створюючи інтелектуальних агентів.

Основна ідея застосування ML-Agents - навчити агентів вчитися та вдосконалювати свої навички на основі взаємодії з ігровим середовищем. ML-Agents дозволяє створювати інтелектуальних агентів для різних ігор та симуляцій, робить їх більш адаптивними та навченими, що додає глибини та інтерактивності грі та дослідженню.

**Метою роботи** є аналіз нейромережових алгоритмів для створення неігрового персонажа, який самостійно прийматиме рішення для наступної дії.

Для досягнення мети необхідно вирішити ряд **задач**:

- дослідження предметної області;
- аналіз існуючих аналогічних алгоритмів;
- пошук та вибір оптимальних методів реалізації;
- розроблення та тестування алгоритму;
- демонстрація результатів виконання алгоритму на тестових даних.

**Об'єктом дослідження** є поведінка неігрового персонажа.

**Предметом дослідження** є методи та алгоритми машинного навчання для визначення поведінки неігрового персонажа.

**Методами дослідження** є моделі машинного навчання та нейронні мережі для розв'язування задач класифікації.

**Очікувана наукова новизна** це розробка алгоритму поведінки неігрового персонажу на основі алгоритмів машинного навчання та його багатопоточна оптимізація для отримання швидких результатів.

**Очікувана практична цінність роботи** це порівняння ефективності різних алгоритмів машинного навчання для ігрового застосунку.

**Апробація роботи.** Основні наукові, теоретичні положення та практичні

результати кваліфікаційної роботи доповідалися і обговорювалися на: кафедрі «Системи штучного інтелекту», НУ ЛП (Львів, 2024)

***Структура і обсяг роботи.*** Кваліфікаційна робота має таку структуру: вступ, три розділи (з висновками до них), висновки для всієї зробленої роботи, список використаних джерел та додатків. Загальний обсяг роботи — 74 сторінок (обсяг без додатків — 72 сторінок).

# 1. АНАЛІТИЧНИЙ РОЗДІЛ

## 1.1. Опис предметного середовища

Створення ігор – один із найбільших сегментів індустрії розваг, масштаби якого можна порівняти з кіноіндустрією. А за швидкістю зростання за останні п'ять років індустрія ігор суттєво її випереджала. У 2020 році за оцінками Google світовий ринок ігор зріс на 23,1% – це був найвищий показник за десятиліття і був оцінений у \$177,8 млрд, очікувалося, що до 2024 року ринок досягне \$218,7 млрд, при цьому середньорічний темп зростання становитиме 9,64% [1]. За даними британського видання Games Industry у 2022-му ринок оцінювався у \$184,4 млрд [2].

За рік цей показник впав на 4,3%. Найбільше прибутку генерують мобільні ігри (50%). За ними йдуть консольні (28%), завантажувані чи фізичні ПК-ігри (21%) та браузерні ПК-ігри (1%). 94,2% продажів ігор були цифровими, лише 5,8% припало на фізичні копії. На ПК фізичні ігри майже не купують (2%), на консолях купують 28%. Найбільше продавалася FIFA 23 (Велика Британія), Call of Duty Modern Warfare 2 (США), у Японії – Splatoon 3.

В умовах пандемії ігрова індустрія, яка у 2020 році вийшла на першу позицію за рівнем затребуваності і залишається там і зараз, незважаючи на невелике падіння у 2022 році, дарує розробникам ігор шанс знайти свою нішу. Навіть в умовах війни європейські та інші компанії були не проти поширити свої робочі місця для українців [3], тому розробка алгоритмів з використанням штучного інтелекту для ігрової індустрії є актуальною. При розробці ігор два шляхи: скористатись повністю готовим ШІ або ж взяти готові концепції та на їх основі створювати власний ШІ. Більш популярним в середовищі розробників є другий підхід.

Існують різні підходи до створення ігрового штучного інтелекту, і одним із найпростіших з них є побудова поведінки персонажів на основі набору простих правил. Однак цю модель важко назвати повноцінним штучним інтелектом, оскільки персонажі, які її використовують, просто виконують певні дії відповідно до заданих програмних правил. Цей підхід підходить лише для дуже

простих ігор, де обмежений набір можливих дій персонажа.

Інший підхід полягає в створенні моделі ігрового штучного інтелекту на основі скінченних автоматів. У такої моделі керований персонаж має набір станів, кожен із яких визначає певну поведінку. Також задаються правила, за якими персонаж переходить між цими станами. Цей підхід широко використовується у сучасних відеоіграх, особливо в тих, де дія відбувається в реальному часі.

Ще одним підходом є використання дерев рішень, що часто застосовуються у комп'ютерних версіях класичних настільних ігор, таких як шахи, шашки і го, а також у іграх стратегії покрокового типу. Штучний інтелект аналізує всі можливі стани гри на певну кількість ходів вперед і обчислює оцінку для кожного можливого ходу на основі того, наскільки цей хід вигідний для нього.

Останнім часом нейронні мережі стали популярними моделями для ігрового штучного інтелекту. Цей підхід вимагає правильної конструкції архітектури мережі, вибору вхідних даних та навчання моделі. Для навчання таких моделей часто використовують підхід навчання з підкріпленням, оскільки його можна використовувати для симуляції безлічі ігрових сценаріїв з випадковими модифікаціями та визначити, яка модель є найкращою для певної гри.

## **1.2. Огляд наявних аналогів**

У [1] досліджено створення системи навігації та управління для керування групою персонажів та координації їх дій у відеогрі жанру стратегії в реальному часі (RTS, рис. 1.1).

Для визначення маршрутів, по яких персонажі повинні переміщатися, застосовано ідеї з теорії поля. Всі об'єкти у грі розглядаються як точкові заряди з різним потенціалом, і для кожної точки обчислюється заряд, що є сумою потенціалів всіх точкових зарядів об'єктів, враховуючи їх відстань. Персонажі стараються уникати точок з негативним зарядом і дістатися до точок з позитивним зарядом. Основним недоліком цього підходу є висока складність

обчислень, оскільки потрібно обчислити заряд для всіх точок у грі, яка може бути безмежною.



*Рис. 1.1. Приклад гри жанру RTS*

У статті [2] розглянуто модель поведінки ботів під час стрільби у відеогрі жанру від першої особи (FPS, рис. 1.2). Автори вказують на те, що розробити бота, який стрілятиме швидко і точно, є відносно просто, оскільки швидкодія комп'ютера та точність його рухів перевищують людські. Однак їх метою є створення моделі поведінки, яка б стріляла подібно до людини, щоб користувач не відрізняв ботів від реальних гравців.

Модель розробляється на основі гри Unreal Tournament 2004 з використанням алгоритму SARSA, враховуючи інформацію про наявну зброю та рух цілі. Навчання моделі проводиться з використанням Reinforcement Learning, де кількість нанесеної шкоди виступає в якості показника винагороди. Однак недоліком є залежність моделі від конкретної ігрової локації і списку доступної зброї, що потребує перенавчання для нових локацій та зброї.

Публікація [3] стосується навігації персонажів, зокрема їх ефективного переміщення у грі, де вони повинні досягти мети в будь-якій точці ігрового світу, будуючи оптимальний маршрут і уникаючи застрягання. Стаття розглядає гру Unreal Tournament 2004. Для навігації ботів використовується граф з численними вершинами, і боти планують свій маршрут за допомогою алгоритму A.





*Рис. 1.2. Приклад гри жанру FPS*

Однак іноді персонажі можуть виходити за межі графа і застрягати. Автори роблять спроби визначити, коли бот застрягає (наприклад, коли немає вершини графа навігації поруч або бот витрачає надто багато часу на коригування маршруту).

У публікації також розглянуто способи вирішення подібних ситуацій. Автори пропонують спостерігати за маршрутами, якими реальні гравці переміщуються по ігровій локації і зберігати ці маршрути у базі даних.

Якщо бот застрягає, то з бази даних виділяється маршрут, який найкраще підходить для поточної позиції бота, і бот рухається цим маршрутом, щоб вибратися з проблемної частини ігрової локації. Цей метод є досить простим і водночас ефективним, а також можливо адаптувати до ігор жанру RPG. Однак основним недоліком є необхідність заповнення бази даних маршрутами для кожної ігрової локації, що вимагає активної участі реальних гравців у грі для збору даних.

У публікації [4] розглядається модель поведінки ігрового штучного інтелекту EISBot у грі стратегії в реальному часі "StarCraft". Ця модель штучного інтелекту використовує "поведінкову мову" (ABL) для створення поведінкових дерев (BT). Вузли і листки цих дерев відповідають цілям і діям для досягнення

цих цілей. Кожному листку дерева присвоюється пріоритет, і модель вибирає дію з найвищим пріоритетом для досягнення певної цілі.

Однак ця модель не вчиться на власних помилках, оскільки пріоритети дій є статичними. Це означає, що, навіть якщо обрана дія виявиться неправильною, її пріоритет не зміниться. Автори пропонують модифікацію моделі, де пріоритети дій стають динамічними, знижуються після помилкової дії і збільшуються після успішної дії. Однак цей підхід потребує декількох помилок, перш ніж модель встановить правильні пріоритети.

У роботі [5] розглянуто підходи до навігації у іграх-платформерах (рис. 1.3) з використанням поведінкових дерев, еволюційних алгоритмів та алгоритму пошуку маршрутів A. Модель випробовується на грі "Mario AI Benchmark", яка є модифікацією класичної гри "Mario". Еволюційний алгоритм будує поведінкове дерево, яке містить різні моделі поведінки для різних ситуацій гри. Однак цей підхід важко адаптується до ігор жанру RPG через відмінності в алгоритмах поведінки у різних жанрах.



*Рис. 1.3. Приклад гри жанру платформер*

Автори роботи [6] розглядають підхід для навігації персонажів у грі стратегії в реальному часі "StarCraft" на основі концепції потенціальних полів та



точкових зарядів. Вони розробляють розширену версію моделі з зарядами, де кожен об'єкт може мати різний точковий заряд для різних персонажів в залежності від їх характеристик та цілей. Однак цей підхід виникає складнощі у вузьких коридорах ігрових локацій.

У роботі [7] розглядається поєднання еволюційного програмування та моделі поведінкових дерев у грі "Mario AI Benchmark". Модель поєднує алгоритм A для навігації з еволюційним програмуванням для створення конкретних екземплярів дерев поведінки.

Однак цей підхід може потребувати декількох спроб для встановлення правильних пріоритетів дій та може бути нестійким до змін в стратегії опонента. У статті розглянуто різні методи вирішення цієї проблеми. Один із них - еволюція поведінкового дерева на різних випадкових рівнях гри або на одному "усередненому" рівні, який має властивості для створення універсального поведінкового дерева. Це означає, що штучний інтелект намагається навчитися пристосовувати свою поведінку до різних ігрових ситуацій.

У статті [8] розглянуто підходи до імітації поведінки людини-гравця в грі "The Open Racing Car Simulator" (рис. 1.4) за допомогою рекурентних нейронних мереж (RNN). Модель отримує на вході різні показники, такі як швидкість автомобіля, відхилення від середини дороги, відстань до країв маршруту тощо, і намагається вести себе схоже до того, як гравець керував би автомобілем.

Оцінка роботи моделі проводиться за допомогою трьох метрик, одна з яких спрямована на якість та швидкість водіння, а дві інші оцінюють схожість з поведінкою людини-гравця. Однак цей підхід складно адаптувати до ігор жанру RPG, оскільки вхідні дані і параметри моделі не відповідають специфіці ігор цього жанру.

У статті [9] досліджено ігровий штучний інтелект для гри у жанрі RPG (рис. 1.5) під назвою "Desktop Dungeons". Головною особливістю цієї гри є те, що рівні завжди генеруються випадково, що ставить вимогу до штучного інтелекту проявляти узагальнюючі здібності.



Рис. 1.4. Приклад гри жанру симулятор

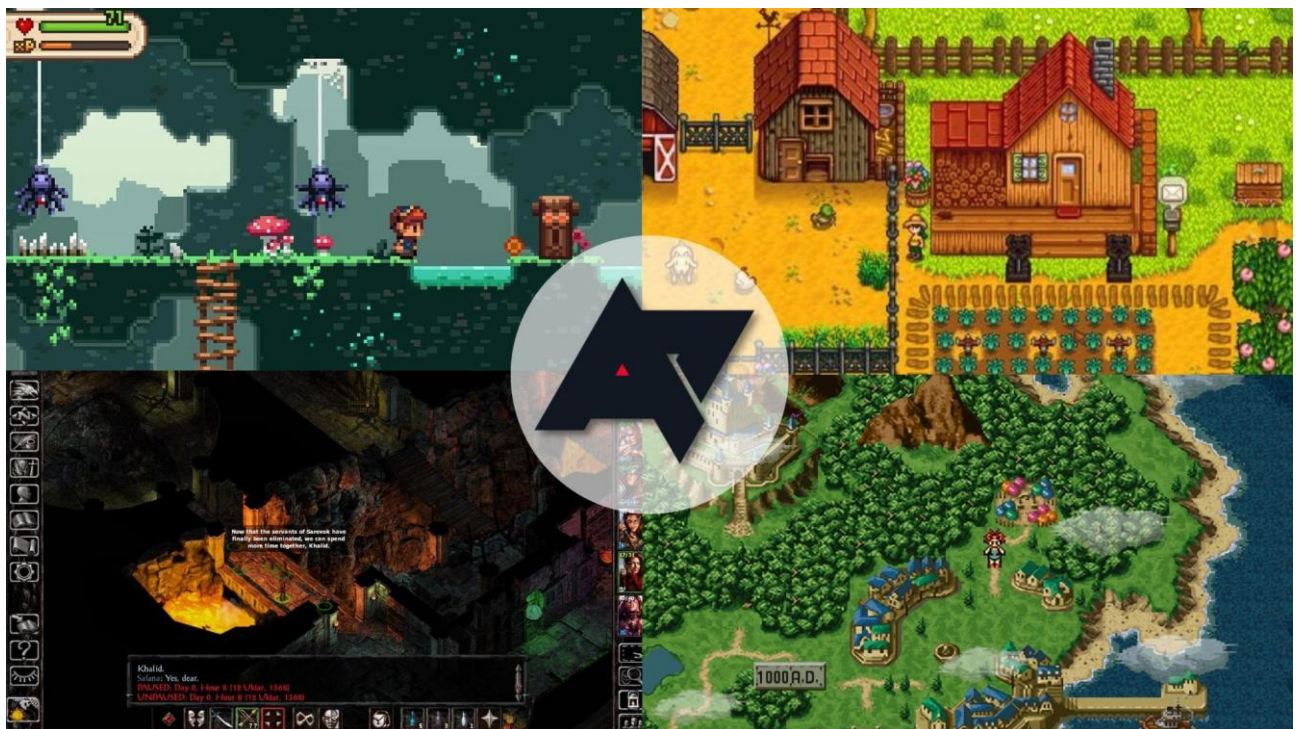


Рис. 1.5. Приклад гри жанру RPG

У роботі розроблено "жадібний" алгоритм, який керує гравцем та еволюційний алгоритм для налаштування параметрів цього алгоритму. "Жадібний" алгоритм має два списки стратегій - для проходження гри та для атаки ворожих персонажів. Еволюційний алгоритм налаштовує параметри стратегій для досягнення максимальної ефективності. Однак основним

недоліком цього підходу є простота "жадібного" алгоритму, який не завжди може забезпечити оптимальну стратегію.

У статті [10] досліджено використання згорткових нейронних мереж (CNN) для гри у жанрі шутера від першої особи (рис. 1.6) "ViZDoom", яка є модифікацією гри "Doom". Модель приймає на вхід зображення з гри і видає дії, які має виконати персонаж. Цей підхід застосовується для ігор з великою кількістю візуальних ігрових елементів.



*Рис. 1.6. Приклад гри жанру шутер*

Оцінка ефективності моделі проводиться за допомогою різних метрик. Однак, цей підхід також складно адаптувати до ігор жанру RPG, оскільки ігри цього жанру мають іншу специфіку і вимагають інших методів обробки інформації.

Для порівняння методів машинного навчання, що використовуються для керування неігровими персонажами у грі виживання, розглянемо кілька існуючих підходів та їхні характеристики. Нижче наведена таблиця з оглядом наявних аналогів, які використовуються у цій сфері.

**Таблиця 1.1** Огляд наявних аналогів

Метод/Підхід	Використані технології	Переваги	Недоліки	Приклади застосування
--------------	------------------------	----------	----------	-----------------------

Нейронні мережі	TensorFlow, Keras, PyTorch	Гнучкість, здатність навчатися складних залежностей, адаптивність до змінних умов	Високі обчислювальні витрати, потреба у великій кількості даних для навчання	Керування поведінкою NPC в іграх, прогнозування дій
Q-навчання (Q-learning)	OpenAI Gym, TensorFlow	Проста реалізація, ефективне навчання у середовищах з обмеженою кількістю дій	Складність у масштабуванні до складних середовищ, потреба у значній кількості епізодів	Ігрові алгоритми, роботи, автономні агенти
Random Forest	Scikit-learn	Висока точність на табличних даних, не потребує масштабування даних, інтерпретованість	Менша ефективність на великих наборах даних, менш ефективний для складних завдань	Аналіз даних, визначення дій NPC
Генетичні алгоритми	DEAP, PyGAD	Ефективність у пошуку глобальних оптимумів, адаптивність до різних типів задач	Високі обчислювальні витрати, тривалість процесу пошуку оптимального рішення	Еволюційні ігрові алгоритми, оптимізаційні задачі
Дерева рішень	Scikit-learn	Простота реалізації та інтерпретації, швидке навчання та прогнозування	Схильність до перенавчання, менша точність на складних задачах	Аналіз даних, прийняття рішень NPC
Лінійна регресія	Scikit-learn, Statsmodels	Простота, швидке навчання та прогнозування	Низька точність на складних нелінійних задачах	Економічне моделювання, прогнозування попиту



У таблиці наведено огляд різних методів машинного навчання, які використовуються для керування неігровими персонажами у грі виживання. Розглянуті методи включають нейронні мережі, Q-навчання, Random Forest, генетичні алгоритми, дерева рішень та лінійну регресію. Для кожного методу зазначено використовувані технології, переваги та недоліки, а також приклади застосування.

Нейронні мережі використовують такі бібліотеки, як TensorFlow, Keras та PyTorch, і забезпечують гнучкість та здатність навчатися складних залежностей. Проте вони потребують значних обчислювальних ресурсів та великої кількості даних для навчання.

Q-навчання (рис. 1.7) зазвичай реалізується з використанням OpenAI Gym та TensorFlow, відрізняється простотою реалізації та ефективним навчанням у середовищах з обмеженою кількістю дій. Проте, воно складне у масштабуванні до більш складних середовищ.

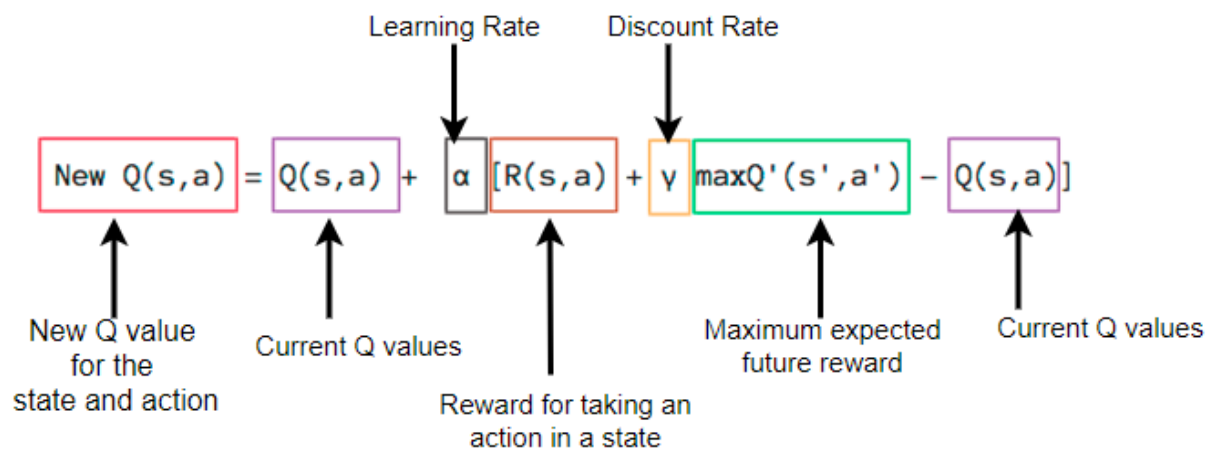


Рис. 1.7. Принцип Q-навчання

Random Forest (рис. 1.8) реалізується за допомогою бібліотеки Scikit-learn. Цей метод забезпечує високу точність на табличних даних та не потребує масштабування, проте менш ефективний для великих наборів даних та складних задач.

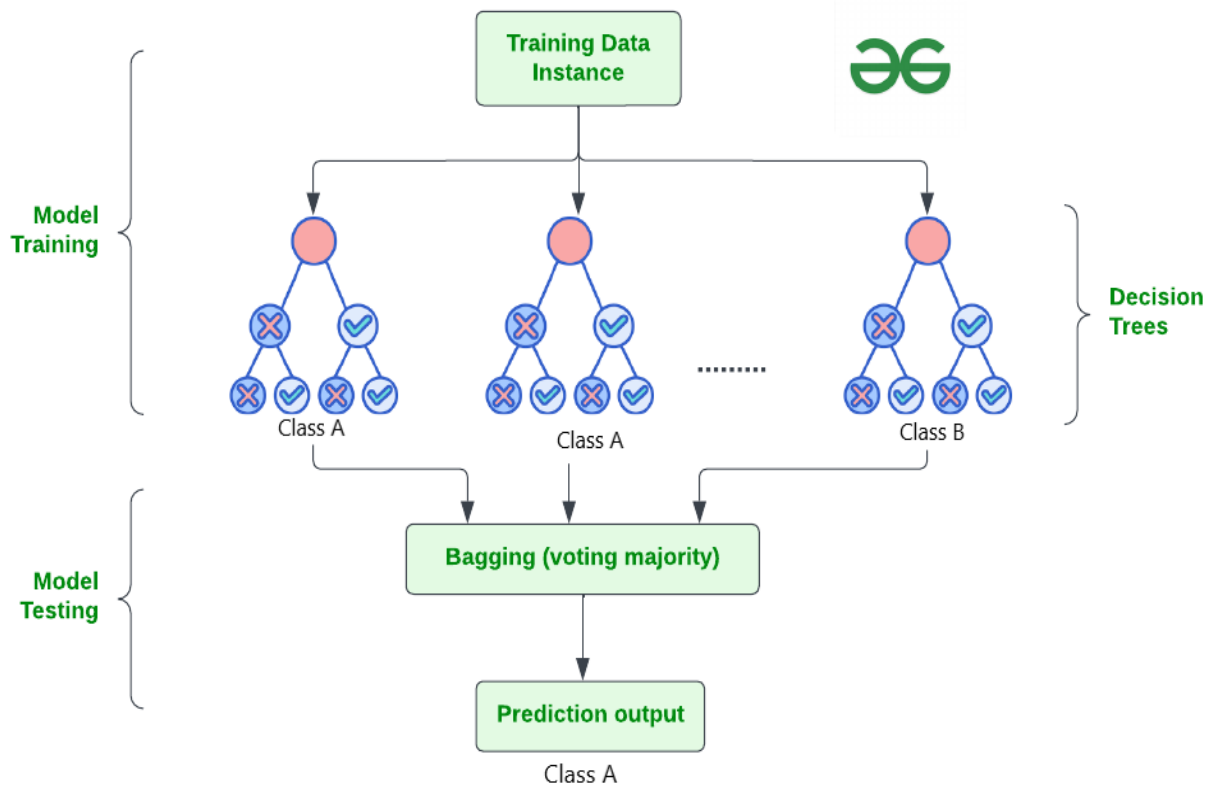


Рис. 1.8. Принцип дії Random Forest

Генетичні алгоритми реалізуються з використанням бібліотек DEAP та PyGAD, і є ефективними у пошуку глобальних оптимумів. Вони адаптивні до різних типів задач, але потребують високих обчислювальних витрат.

Дерева рішень (рис. 1.9) також реалізуються за допомогою Scikit-learn, забезпечують простоту реалізації та інтерпретації, але схильні до перенавчання і менш точні на складних задачах.

Лінійна регресія (рис. 1.10) є найпростішим методом, що реалізується з використанням Scikit-learn та Statsmodels. Вона забезпечує швидке навчання та прогнозування, але має низьку точність на складних нелінійних задачах.

Цей огляд дозволяє зрозуміти різні підходи до керування неігровими персонажами та вибрати найефективніший метод для конкретних умов гри.

У загальному, в ігровому штучному інтелекті найчастіше використовуються різні алгоритми і моделі, залежно від жанру гри та його специфіки.

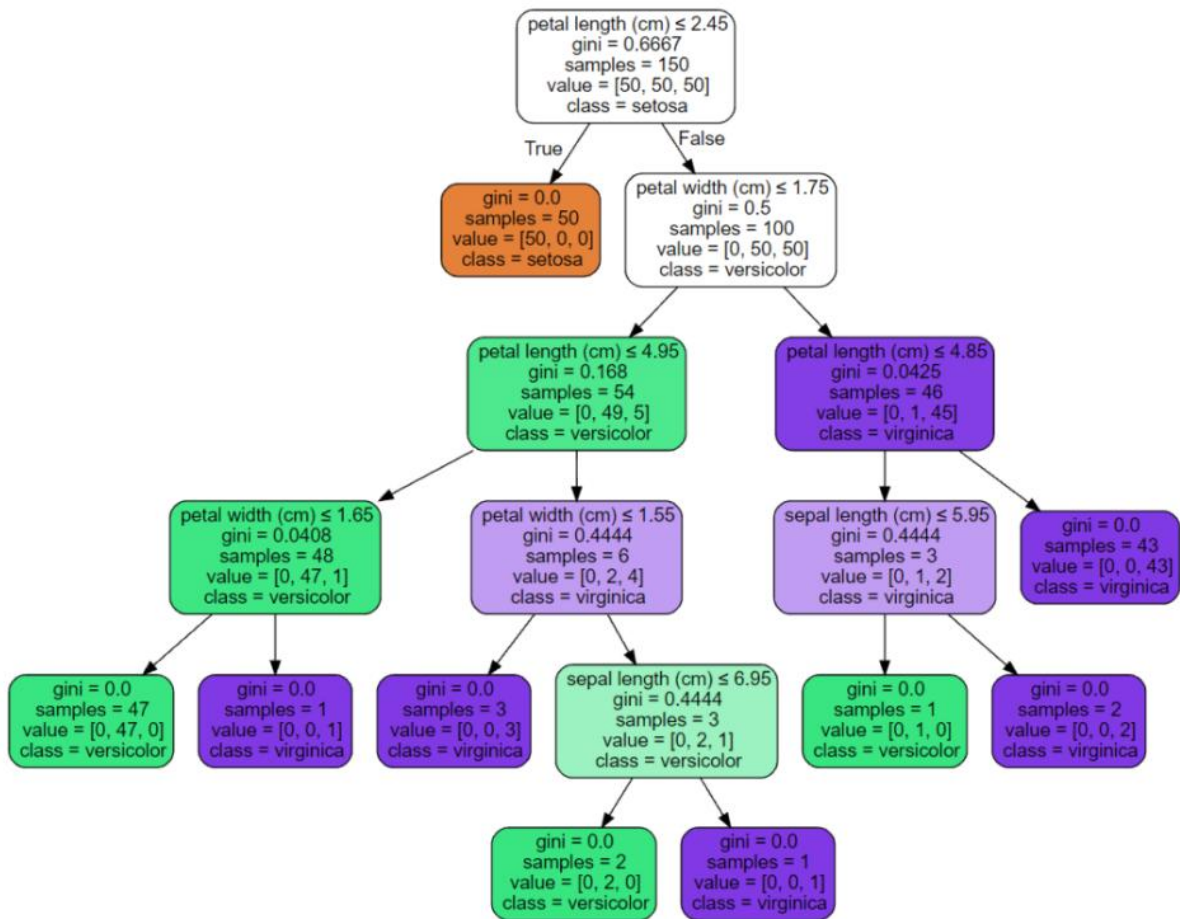


Рис. 1.9. Приклад дерева рішень

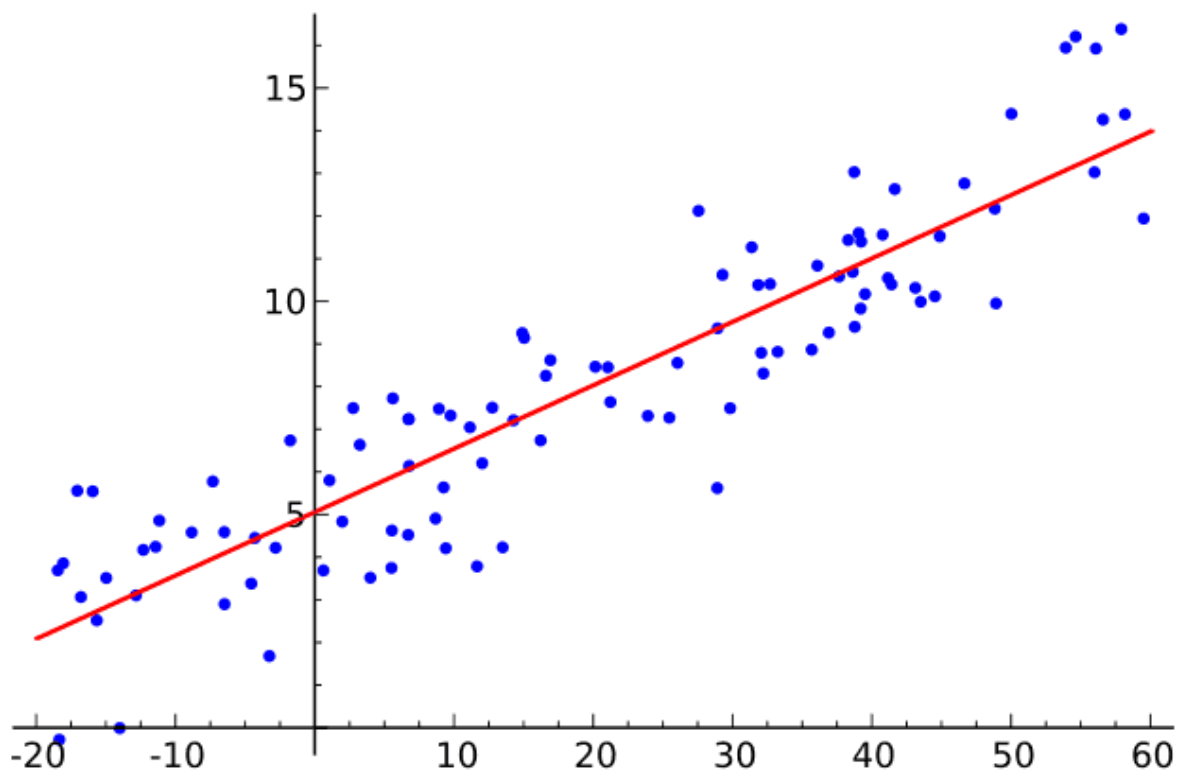


Рис. 1.10. Принцип лінійної регресії

### 1.3. Постановка задачі

Тема даної роботи – "Аналіз методів машинного навчання для керування неігровими персонажами у грі виживання". Основною метою роботи є дослідження та порівняння різних методів машинного навчання, які можуть бути застосовані для автоматизації прийняття рішень неігровими персонажами у грі виживання.

Неігрові персонажі в сучасних іграх відіграють важливу роль у створенні динамічного та цікавого ігрового процесу. Вони взаємодіють з гравцями, середовищем та іншими NPC, що вимагає від них прийняття складних рішень у реальному часі. Для досягнення цієї мети необхідно використовувати ефективні методи машинного навчання, які можуть адаптуватися до змінних умов гри та забезпечувати реалістичну поведінку NPC.

У рамках даного дослідження поставлено наступні задачі:

- 1) провести огляд наявних аналогів методів машинного навчання, що використовуються для керування неігровими персонажами у іграх виживання;
- 2) обґрунтувати вибір методів для подальшого аналізу та порівняння;
- 3) розробити алгоритми на основі обраних методів машинного навчання для керування NPC у грі виживання;
- 4) провести експериментальну перевірку ефективності розроблених алгоритмів на основі згенерованих або реальних даних з гри;
- 5) проаналізувати отримані результати та оцінити ефективність кожного з методів;
- 6) надати рекомендації щодо використання певних методів для різних типів задач у контексті гри виживання.

Вирішення цих задач дозволить визначити найбільш підходящі методи машинного навчання для автоматизації прийняття рішень NPC, що сприятиме покращенню ігрового процесу та підвищенню його реалізму.

Таким чином, дане дослідження спрямоване на розробку та апробацію методів машинного навчання для створення інтелектуальних NPC у грі



виживання, що здатні приймати складні рішення у змінних умовах гри. Це допоможе у подальшому розвитку ігор та забезпеченні їхньої інтерактивності та цікавості для гравців.

#### **1.4. Висновки до розділу**

В першому розділі була описана актуальність дослідження впровадження штучного інтелекту в іграх. Також проведений аналіз інформації щодо наявних аналогів, а саме досліджені ІІІ в таких іграх як Unreal Tournament 2004, StarCraft, Desktop Dungeons та ViZDoom, базуючись на статтях з дослідженнями використання нейронних мереж в цих іграх. У результаті виявлено, що кожна гра має свої особливі методи для застосування НМ.

Далі були розглянуті найбільш релевантні наукові статті, на базі яких була сформована та обґрунтована основна проблема роботи. Наступним кроком було більш чітко сформульована мета та задача дослідження.

Після детального аналітичного огляду літературних джерел наведено постановку задач роботи, яка полягає у аналізі методів машинного навчання для створення поведінки неігрового персонажу в грі.

## 2. ДОСЛІДНИЦЬКИЙ РОЗДІЛ

### 2.1. Вхідні дані

Для проведення дослідження з аналізу методів машинного навчання для керування неігровими персонажами у грі виживання необхідно підготувати вхідні дані. Вхідні дані відіграють важливу роль у процесі навчання моделей машинного навчання, оскільки саме на їх основі моделі навчаються приймати рішення та реагувати на різні ситуації у грі.

Основним джерелом даних для даного дослідження є лог-файли та ігрові сценарії, що містять інформацію про поведінку NPC у грі виживання. Ці дані можуть включати:

- 1) статистику ігрових сесій (кількість ворогів, рівень здоров'я, доступні ресурси тощо);
- 2) дії, виконані NPC у відповідь на певні події або зміни в ігровому середовищі;
- 3) часові мітки подій, які дозволяють відстежувати послідовність дій NPC.

Додаткові дані можуть бути отримані з:

- 1) реальних ігрових сесій, записаних під час гри;
- 2) згенерованих даних, які імітують можливі сценарії у грі.

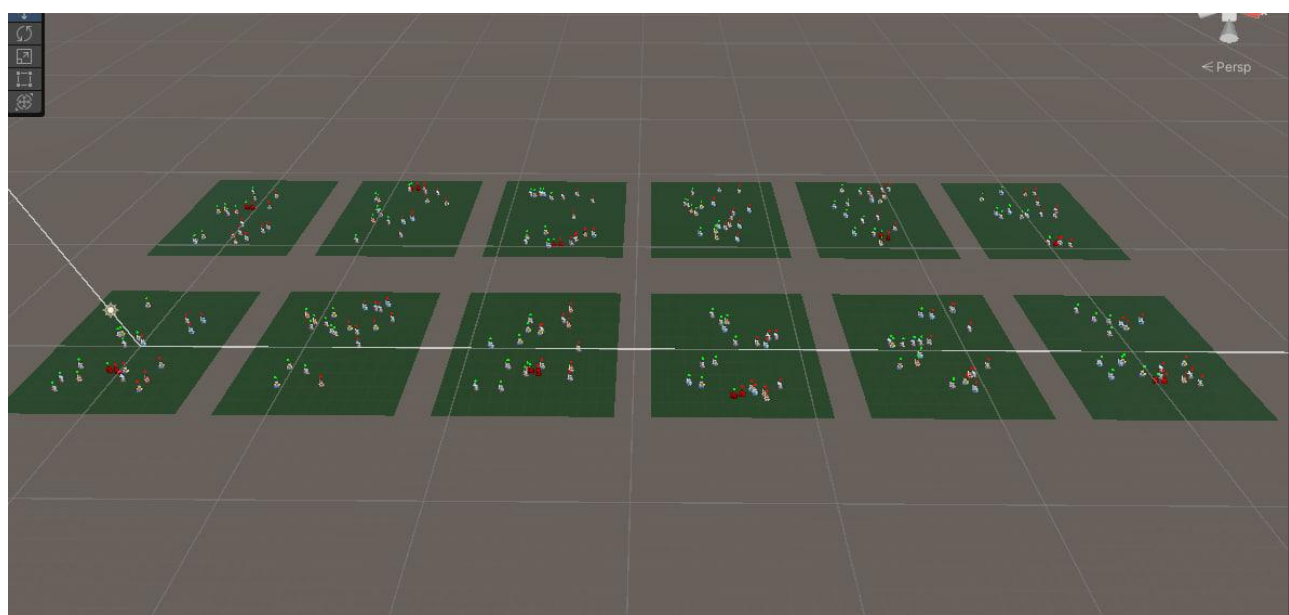
У межах дослідження було реалізовано повноцінну систему симуляційної гри виживання з використанням ігрового рушія Unity, яка дозволила створити, зібрати та структуровано зберегти великий обсяг даних для подальшого навчання моделей штучного інтелекту. Збір даних є одним із ключових етапів у будь-якому проєкті машинного навчання, адже саме якісно підготовлений набір прикладів дозволяє моделі навчитись ефективно приймати рішення в динамічному середовищі.

Метою збору даних було фіксування послідовних ігрових станів та дій персонажів (агентів) під час великої кількості автоматизованих симуляцій гри між двома командами. Кожна команда складалася з десяти ботів (неігрових персонажів), які діяли за простими скриптованими алгоритмами. Усі дії ботів

відбувалися на обмеженому полі бою (від  $-4$  до  $4$  по координатах  $x$  і  $z$ ), а сам процес гри було реалізовано з підтримкою основних параметрів, що характеризують кожного бійця: здоров'я, рівень броні, рівень зброї, позиція, стан (живий/мертвий), дистанція до ворога, і поточна дія.



*Рис. 2.1. Ігровий процес*



*Рис. 2.2. Процес збору даних шляхом проходження багатьох паралельних ігор*

Для досягнення необхідного обсягу даних було проведено автоматичний запуск великої кількості симуляцій гри, в яких команди з різним складом і характеристиками діяли у випадкових умовах. У кожній симуляції, при кожному кадрі, для кожного воїна фіксувався поточний стан середовища, прийнята дія та її результат. Таким чином, було зібрано 10 000 прикладів пар "вхід — дія", де вхідні параметри представляли стан агента, а дія — його реакцію в даний момент.

Запис даних реалізовувався за допомогою C#-скриптів у Unity, які виконували логування кожного рішення бота.

Лістинг 2.1 – C# клас BattleLogger для запису даних, та методи отримання даних і запису вкінці гри:

```
public class BattleLogger : MonoBehaviour
{
    public static BattleLogger Instance;
    private List<string> _rows = new List<string>();
    private string _filePath;
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(this);
            _filePath = Path.Combine(Application.persistentDataPath, "output.csv");
            // Заголовок CSV
            _rows.Add("team_id,position_x,position_z,health,armor,weapon_level,distance_to_enemy,action,result,alive");
        }
        else
        {
            Destroy(gameObject);
        }
    }
}
```

```

        public void LogAgentData(int teamId, Vector3 position, float health, float armor, int
weaponLevel, float distanceToEnemy, int action, string result, bool isAlive)
        {
            string row =
$"{teamId},{position.x:F2},{position.z:F2},{health:F1},{armor:F1},{ weaponLevel},{distanceToE
nemy:F2},{action},{result},{(isAlive ? 1 : 0)}";
            _rows.Add(row);
        }
        public void SaveToFile()
        {
            File.WriteAllLines(_filePath, _rows.ToArray());
            Debug.Log($"Збережено лог у файл: {_filePath}");
        }
    }

    // метод отримання даних
    private void LogAction(int actionCode, string result)
    {
        float distanceToEnemy = target != null ? Vector3.Distance(transform.position,
target.transform.position) : 0f;
        BattleLogger.Instance.LogAgentData(
            teamId: (team == Team.TeamA ? 0 : 1),
            position: transform.position,
            health: health,
            armor: armor,
            weaponLevel: weaponLevel,
            distanceToEnemy: distanceToEnemy,
            action: actionCode,
            result: result,
            isAlive: isAlive
        );
    }

    // метод запису в кінці гри
    public void CheckVictoryCondition()

```

```

{
    bool teamADeath = teamA.TrueForAll(w => !w.IsAlive);
    bool teamBDeath = teamB.TrueForAll(w => !w.IsAlive);
    if (teamADeath || teamBDeath)
    {
        string winner = teamADeath && teamBDeath ? "Draw" : (teamADeath ? "Team B wins"
: "Team A wins");
        Debug.Log(winner);
        // Зберегти лог
        BattleLogger.Instance.SaveToFile();
        gameStarted = false;
    }
}

```

### Кінець лістингу 2.1

Зібрана інформація зберігалась у вигляді CSV-файлів з фіксованою структурою. Кожен рядок у такому файлі містив наступні поля:

- position\_x, position\_z — координати персонажа на полі бою;
- health — залишок здоров'я на момент прийняття рішення;
- armor — поточний рівень броні;
- weapon\_level — рівень озброєння;
- distance\_to\_enemy — відстань до найближчого ворога;
- num\_enemies\_nearby — кількість ворогів у радіусі 2 одиниць;
- num\_allies\_nearby — кількість союзників у радіусі 2 одиниць;
- distance\_to\_nearest\_ally — відстань до найближчого союзника;
- enemy\_health\_avg\_nearby — середнє здоров'я ворогів у межах дії
- enemy\_weapon\_level\_max — рівень зброї найближчого ворога;
- action — дія, яку виконав бот (кодування: 0 — стояти, 1 — рухатись, 2 — атакувати);
- result — результат дії (hit, killed, moved);
- alive — бінарна змінна, яка відображає, чи живий персонаж після дії;
- team\_id — ідентифікатор команди (для подальшого групування).

```
{  
  "position_x": 2.45,  
  "position_z": -1.87,  
  "health": 72,  
  "armor": 30,  
  "weapon_level": 3,  
  "distance_to_enemy": 1.43,  
  "num_enemies_nearby": 2,  
  "num_allies_nearby": 3,  
  "distance_to_nearest_ally": 0.78,  
  "enemy_health_avg_nearby": 64.5,  
  "enemy_weapon_level_max": 4,  
  "action": 2,  
  "result": "hit",  
  "alive": 1,  
  "team_id": 1  
}
```

*Рис. 2.3. Приклад запису однієї дії агента*

Після збору даних усі записи було піддано передобробці: перевірці на відсутність помилок логування, фільтрації некоректних значень, нормалізації числових полів (у діапазоні від 0 до 1) та переведенню дій у категоріальний формат для задачі класифікації.

Збір даних відбувався в офлайн-режимі, тобто після кожного бою система записувала файл з усіма діями, які виконували агенти під час симуляції. Завдяки автоматизації процесу збору (постійне оновлення сцени без втручання користувача) вдалося зібрати великий об'єм інформації у відносно короткий час (менш ніж за добу на одній машині).

Крім того, дані були репрезентативні та різноманітні, оскільки випадкова ініціалізація характеристик воїнів у кожному бою (здоров'я, броня, зброя) забезпечувала широкий спектр ситуацій. Таким чином, модель, яка навчається на таких даних, отримує змогу побачити різні бойові сценарії: від прямого

зіткнення до ситуацій, коли необхідно ухилитись або зберігати ресурс.

Варто зазначити, що при першому запуску збору були виявлені проблеми з неконсистентністю даних: дії іноді не відповідали логіці бою (наприклад, атака без цілі). Це було вирішено шляхом додаткової перевірки наявності ворога у зоні досяжності перед виконанням дії. Також було введено обмеження на кількість дій за одну секунду, щоб уникнути аномально великої кількості записів в одному бою.

Таким чином, реалізований процес збору даних забезпечив надійний, об'ємний і структурований набір даних, що дозволив застосовувати моделі машинного навчання з високою точністю та стабільністю. Це стало фундаментом для побудови інтелектуальної поведінки неігрових агентів у грі виживання.

Перед тим як використовувати дані для навчання моделей машинного навчання, необхідно провести їх попередню обробку. Цей процес включає:

- 1) очистку даних: видалення або коригування некоректних та відсутніх значень;
- 2) форматування даних: приведення даних до єдиного формату для зручності аналізу та обробки;
- 3) нормалізацію даних: масштабування значень характеристик до одного діапазону, що покращує стабільність та швидкість навчання моделей;
- 4) кодування категоріальних змінних: перетворення категоріальних даних у числовий формат для подальшого використання у моделях.

Було виконано аналіз залежностей між вхідними параметрами та цільовою змінною action. Найвищу логічну кореляцію виявлено для таких ознак:

- distance\_to\_enemy — ключова ознака, яка безпосередньо визначає доцільність атаки. Відстань менше за 1.0 суттєво збільшує частку дій типу «атака».
- weapon\_level — впливає на ймовірність нападу. Агресивні дії частіше обираються агентами з рівнем зброї від 3 і вище.
- num\_enemies\_nearby — зменшує ймовірність атаки, оскільки бот



оцінює ризик отримати шкоду одразу від декількох ворогів.

- `enemy_weapon_level_max` — служить показником потенційної загрози. Високі значення гальмує прийняття рішень на користь атаки.
- `distance_to_nearest_ally` — велика відстань до союзника асоціюється з обережною поведінкою (рух або очікування).
- `num_allies_nearby` — дає змогу агенту приймати сміливіші рішення, перебуваючи в групі.

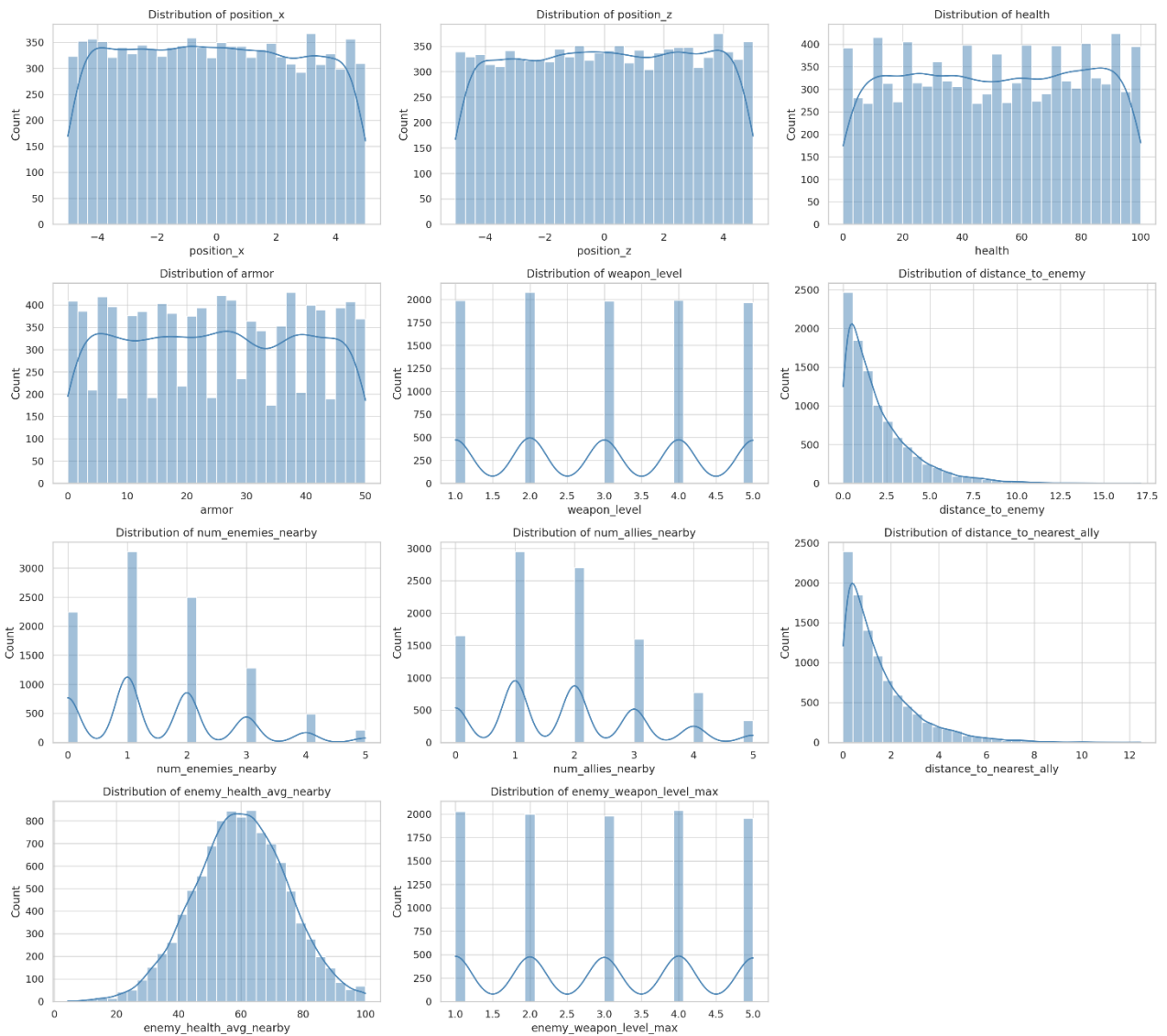


Рис. 2.4. Графіки розподілу даних в датасеті

До навчання моделі включено всі ознаки, окрім `result`, `team_id` та `alive`, оскільки вони або не доступні в момент прийняття рішення (`result`, `alive`), або можуть викривити модель (`team_id`). Позиція агента має низьку інформативність

без урахування конкретної геометрії поля або об'єктів (наприклад, укриттів). Їх доцільно залишити лише в моделях, що враховують просторову поведінку (наприклад, у reinforcement learning або CNN).

Для покращення навчання нейромережі було створено агреговані ознаки. Ці ознаки будуються на основі комбінацій вже існуючих полів або з урахуванням тактичної ситуації, і здатні надати моделі глибший контекст.

**Таблиця 2.1** Опис агрегованих ознак

Назва ознаки	Тип	Формула / Опис
power_ratio	float	$(\text{health} + \text{armor}) / (\text{enemy\_health\_avg\_nearby} + 1)$ — співвідношення сил
nearby_pressure	float	$\text{num\_enemies\_nearby} - \text{num\_allies\_nearby}$ — тактична перевага
is_enemy_dominant	binary	1, якщо $\text{enemy\_weapon\_level\_max} > \text{weapon\_level}$
is_in_range_to_attack	binary	1, якщо $\text{distance\_to\_enemy} < 0.5$
total_survivability	int	$\text{health} + \text{armor}$ — сумарна витривалість

Детальний опис ключових ознак:

- power\_ratio — Ця ознака дозволяє оцінити, наскільки сильний бот у порівнянні з ворогами поблизу. Якщо значення  $< 1$ , атакувати може бути небезпечно.
- nearby\_pressure — Проста оцінка переваги на полі: якщо ворогів більше — значення буде позитивним, якщо союзників більше — негативним.
- is\_enemy\_dominant — Бінарна ознака, яка швидко сигналізує, чи є

поруч ворог зі зброєю сильнішого рівня. Дає моделі змогу уникати безнадійних боїв.

Для побудови та оцінки моделей машинного навчання необхідно розподілити дані на навчальні та тестові набори:

1) навчальні дані використовуються для навчання моделі, вони містять основну частину даних;

2) тестові дані використовуються для оцінки точності та ефективності моделі на нових, невідомих даних.

Лістинг 2.2 - Приклад коду для розподілу даних:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# === 1. ЗАВАНТАЖЕННЯ ДАНИХ ===
data = pd.read_csv("output.csv")

# === 3. ПЕРЕТВОРЕННЯ КАТЕГОРІЙНИХ ПОЛІВ ===
# Припускаємо, що action – цільова змінна
X = data.drop(columns=["action"])
y = data["action"]

# Якщо result – текстовий, можемо перетворити в числовий
if "result" in X.columns and X["result"].dtype == 'object':
    X["result"] = X["result"].astype('category').cat.codes

# === 4. НОРМАЛІЗАЦІЯ ЧИСЕЛ ===
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# === 5. РОЗПОДІЛ ДАНИХ ===
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
```

Попередня обробка та розподіл даних є важливими етапами у підготовці до навчання моделей машинного навчання. Вони забезпечують якість та релевантність даних, що дозволяє отримати більш точні та надійні результати під час аналізу методів машинного навчання для керування неігровими персонажами у грі виживання.

### 2.2. Вибір методів машинного навчання

Вибір методів машинного навчання для керування неігровими персонажами у грі виживання є важливим етапом дослідження. Цей розділ містить обґрунтування вибору методів та порівняння їх на основі літературних джерел і попередніх досліджень. Основні критерії вибору включають точність, швидкість навчання, інтерпретованість, адаптивність до змінних умов гри та обчислювальні витрати.

Методи машинного навчання, обрані для даного дослідження, включають нейронні мережі, Q-learning та Random Forest. Вибір обґрунтований наступними причинами:

1) нейронні мережі (MLPClassifier) забезпечують високу гнучкість і здатність навчатися складних залежностей у даних, що є важливим для прийняття рішень NPC в умовах змінного ігрового середовища;

2) Q-learning є одним з ефективних методів підкріплювального навчання, який дозволяє NPC адаптувати свою поведінку на основі отриманого досвіду;

3) Random Forest є методом для класифікації та регресії, який добре працює на табличних даних і забезпечує високу точність при низьких обчислювальних витратах.

Нижче наведена таблиця порівняння методів машинного навчання, що використовуються для керування NPC у грі виживання.

**Таблиця 2.2** Порівняння методів машинного навчання

Метод/Підхід	Переваги	Недоліки	Приклади
--------------	----------	----------	----------

			<b>застосування</b>
Нейронні мережі	Висока гнучкість, здатність навчатися складних залежностей	Високі обчислювальні витрати, потреба у великій кількості даних	Керування поведінкою NPC, прогнозування дій
Q-learning	Ефективне навчання на основі досвіду, проста реалізація	Складність масштабування до складних середовищ, потреба у великій кількості епізодів	Роботи, автономні агенти, ігрові алгоритми
Random Forest	Висока точність, не потребує масштабування даних	Менш ефективний для великих наборів даних	Аналіз даних, визначення дій NPC

Вибір алгоритмів для подальшого аналізу здійснено на основі їхніх переваг та застосувань у контексті гри виживання:

- 1) нейронні мережі (MLPClassifier): будуть використовуватися для моделювання складних залежностей та адаптивної поведінки NPC;
- 2) Q-learning: дозволить розробити алгоритми підкріплювального навчання для покращення адаптивності NPC до різних ігрових ситуацій;
- 3) Random Forest: буде застосовуватися для аналізу та прогнозування дій NPC на основі наявних табличних даних.

Таким чином, вибір методів машинного навчання базується на їхніх сильних сторонах та придатності для задач керування неігровими персонажами у грі виживання. Подальший аналіз та експериментальне дослідження дозволять визначити найбільш ефективний метод для конкретних умов гри.

У даній роботі запропоновано гібридний метод навчання агентів, який поєднує переваги supervised learning (SL) і reinforcement learning. Ідея полягає у тому, щоб ініціалізувати поведінку агента моделлю, натренованою на попередньо зібраному датасеті (Supervised Learning), а потім донавчати агента у середовищі за допомогою RL (PPO). Такий підхід потенційно дозволяє значно прискорити процес тренування та підвищити якість кінцевої політики.

## 2.3. Опис нейронних мереж для керування NPC

### 2.3.1 Архітектура MLP

Для керування неігровими персонажами у грі виживання була обрана архітектура нейронної мережі, яка забезпечує гнучкість та адаптивність до змінних умов гри. Нейронна мережа, яка буде використана, є багат шаровим персептроном (MLP), що складається з вхідного шару, кількох прихованих шарів та вихідного шару.

Архітектура нейронної мережі:

1) вхідний шар: отримує вхідні дані, що включають характеристики стану гри, такі як здоров'я NPC, кількість ворогів поблизу, наявність ресурсів тощо;

2) приховані шари: один або кілька шарів, що складаються з нейронів, які виконують нелінійні перетворення вхідних даних. Кількість нейронів та шарів визначається експериментально для досягнення найкращих результатів;

3) вихідний шар: видає рішення про дію, яку має виконати NPC (наприклад, атака, втеча, збір ресурсів).

Для побудови та тренування нейронної мережі використовуватиметься бібліотека `scikit-learn`, яка забезпечує простоту реалізації та ефективність обчислень.

На рисунку 2.5 представлена діаграма архітектури нейронної мережі.

Опис діаграми:

- `GameEnvironment` – представляє собою ігрове середовище, яке взаємодіє з NPC. Воно містить поточний стан гри та методи для оновлення стану на основі дій;

- `State` - клас, що описує стан гри, який включає різні характеристики, такі як здоров'я NPC, кількість ворогів поблизу, наявність ресурсів та час доби. Містить методи для отримання даних стану;

- `NeuralNetwork` - клас, що представляє нейронну мережу, яка складається з вхідного шару, двох прихованих шарів та вихідного шару.

Містить методи для прямого проходу (forward pass), тренування (train) та прогнозування (predict) дій NPC;

- Action - клас, що представляє дію, яку має виконати NPC (наприклад, атака, втеча, збір ресурсів).

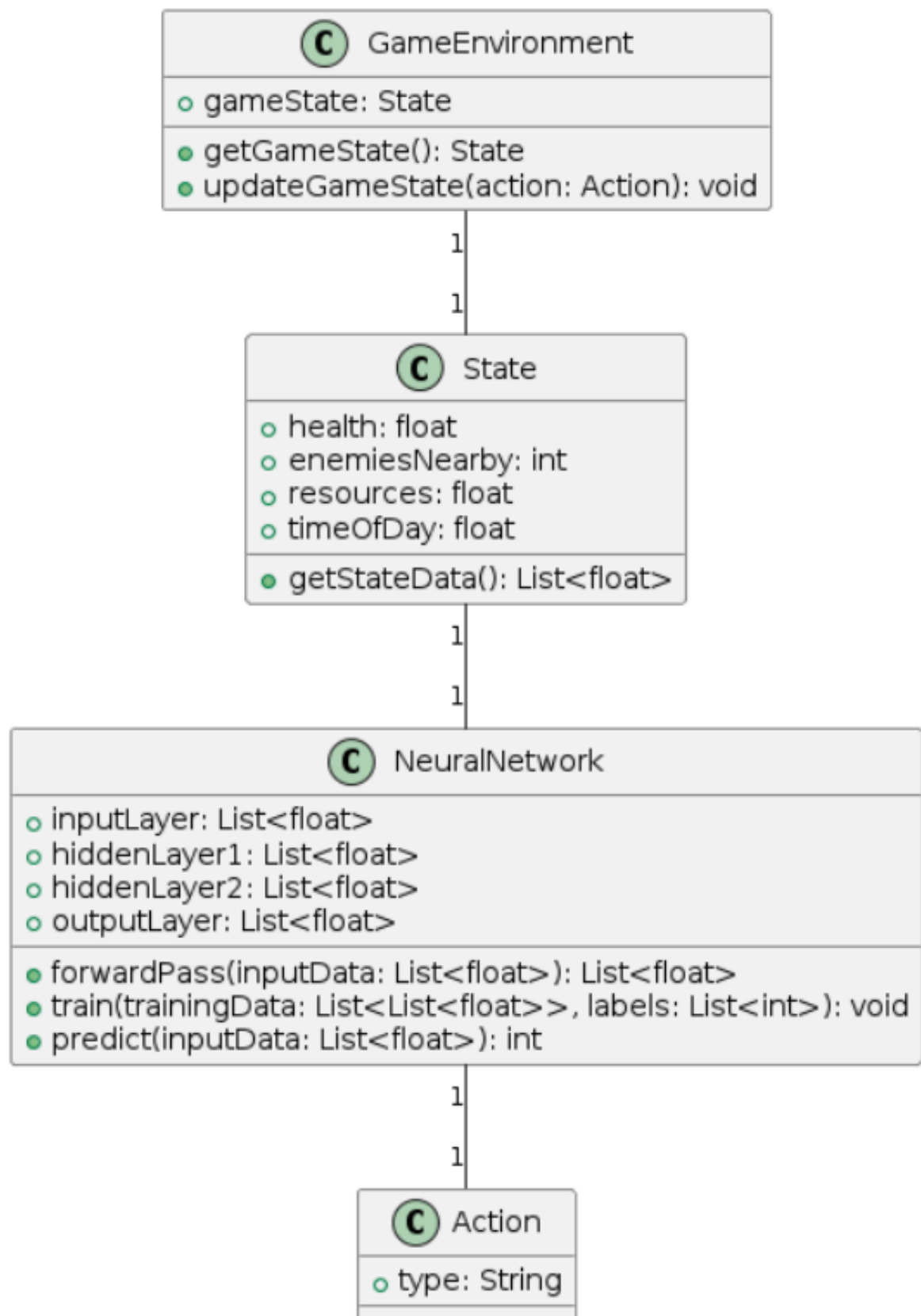


Рис. 2.5 - Діаграма архітектури нейронної мережі

Лістинг 2.3 - Приклад коду для побудови та тренування нейронної мережі



```

# === 4. Побудова нейронної мережі ===
model = Sequential()
model.add(Dense(64, activation='relu',
input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(y_train.shape[1], activation='softmax')) # softmax
для багатокласової класифікації

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# === 5. Тренування ===
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)

# === 6. Оцінка ===
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

```

Кінець лістингу 2.3

Загалом, цей код демонструє основні етапи побудови та тренування нейронної мережі для керування неігровими персонажами у грі виживання, починаючи з підготовки даних і закінчуючи оцінкою моделі. Таким чином, архітектура нейронної мережі може також бути налаштована для досягнення якнайкращих результатів залежно від конкретної специфіки гри та доступних даних.

### 2.3.2 Q-Learning

Q-learning — це один із найпоширеніших алгоритмів навчання з підкріпленням (Reinforcement Learning, RL), що дозволяє агенту вивчати оптимальну стратегію поведінки у невідомому середовищі без необхідності знати модель цього середовища. У рамках даної роботи Q-learning розглядається як базовий підхід до розробки інтелектуальної поведінки ігрових агентів у

симуляціях виживання.

Мета Q-learning — знайти оптимальну політику (policy) — тобто функцію, яка в кожному стані середовища визначає найкращу дію для досягнення довгострокової максимальної винагороди. У цьому підході використовується Q-функція (функція якості), яка для кожної пари "стан–дія" оцінює очікувану суму нагород (reward), яку агент може отримати в майбутньому, починаючи з цієї пари та діючи оптимально надалі.

Алгоритм Q-learning є off-policy методом, тобто навчається незалежно від політики, яка використовується для вибору дій. Наведемо псевдокод алгоритму:

1. Ініціалізувати Q-таблицю випадковими значеннями (або нулями);
2. Для кожного епізоду:
  - 1) Встановити початковий стан  $s$ ;
  - 2) Поки стан не є кінцевим:
    - Вибрати дію  $a$  згідно з політикою  $\epsilon$ -жадібності ( $\epsilon$ -greedy):
      - з імовірністю  $\epsilon$  — випадкова дія;
      - з імовірністю  $1 - \epsilon$  — дія з максимальним Q-значенням;
    - Виконати дію  $a$ , спостерігати новий стан  $s'$  і винагороду  $r$ ;
    - Оновити Q-функцію за формулою Беллмана;
    - Перейти до нового стану  $s \leftarrow s'$ .

У даному проєкті Q-learning застосовується до гри виживання, де кожен агент (воїн) діє в обмеженому полі бою. Простір станів включає:

- координати агента;
- відстань до найближчого ворога;
- рівень здоров'я;
- рівень броні;
- рівень зброї.

Простір дій дискретний і включає:

- стояти;
- рухатись до ворога;
- атакувати;
- тікати.

Нагорода формується за наступною логікою:

- +1 — за успішну атаку;
- -1 — за отримання шкоди;
- +5 — за вбивство ворога;
- -2 — за смерть персонажа;
- 0 — за бездіяльність.

Для уникнення надмірної складності було використано дискретизацію станів, тобто розбиття простору позицій і характеристик на обмежену кількість інтервалів. Таким чином, агент оперує Q-таблицею, розмір якої визначається як добуток кількості дискретних станів і можливих дій.

### 2.3.3 Random forest

Random Forest (випадковий ліс) — це потужний ансамблевий метод машинного навчання, який використовується для вирішення задач класифікації та регресії. Він базується на поєднанні великої кількості простих моделей — дерев рішень (Decision Trees), які працюють разом для прийняття остаточного рішення. Завдяки своїй здатності обробляти великі обсяги табличних даних, стійкості до перенавчання та інтерпретованості, Random Forest є надійним вибором для створення систем штучного інтелекту у багатьох прикладних галузях, зокрема й в ігровій індустрії.

У межах цього дослідження Random Forest застосовується як один із підходів до побудови поведінкової моделі неігрових персонажів (агентів) у симуляції виживання. Метою було навчити модель передбачати дії агентів (рух, атака, очікування) на основі їхнього поточного стану.

Кожне дерево в лісі будується за допомогою наступних принципів:

1. *Bootstrap-підвибірка*: для кожного дерева випадковим чином обирається підмножина прикладів із навчального набору (із

поверненням), що забезпечує різноманітність між деревами.

2. *Випадковий вибір ознак*: під час побудови кожного вузла дерева розглядається лише випадкова підмножина ознак, що сприяє зменшенню кореляції між деревами.

3. *Глибина дерев необмежена*: кожне дерево може рости до повної глибини (до повного розділення прикладів), хоча в практиці часто застосовують обмеження глибини або кількості листків.

У цій роботі вхідний простір ознак включав:

- position\_x, position\_z — координати агента;
- health, armor — поточне здоров'я і броня;
- weapon\_level — рівень зброї;
- distance\_to\_enemy — відстань до найближчого ворога;
- team\_id — команда агента (0 або 1).

Цільова змінна: action — категоріальна змінна, що представляє дію, яку виконав агент (0 — idle, 1 — move, 2 — attack).

## **2.4. Дослідження впливу різних факторів на роботу моделей**

У цьому розділі досліджується вплив різних факторів на роботу моделей машинного навчання для керування неігровими персонажами (NPC) у грі виживання. Вплив таких факторів, як кількість навчальних даних, характеристики ігрового середовища, різні сценарії гри та налаштування параметрів моделей, оцінюється для визначення їхнього впливу на точність і продуктивність моделей.

Збільшення кількості тренувальних даних зазвичай покращує точність моделей машинного навчання. Для цього дослідження було проведено експерименти з різною кількістю тренувальних даних (500, 1000, 1500, 2000 прикладів) та оцінено точність моделей.

Характеристики ігрового середовища, такі як кількість ворогів, рівень здоров'я NPC та наявність ресурсів, можуть суттєво впливати на роботу моделей. Дослідження впливу кожної характеристики дозволяє визначити їхній внесок у загальну продуктивність моделі.

Моделі були протестовані у різних сценаріях гри, таких як атака ворогів, уникнення небезпеки та збір ресурсів. Аналіз результатів показує, як добре моделі адаптуються до різних умов та змін у грі.

Налаштування гіперпараметрів моделей, таких як кількість нейронів у прихованих шарах та кількість ітерацій для тренування, впливає на точність і швидкість роботи моделей. Експерименти з різними значеннями параметрів допомогли визначити оптимальні налаштування.

Таблиця 2.3 демонструє вплив різних факторів на роботу моделей машинного навчання.

**Таблиця 2.3.** Вплив різних факторів на роботу моделей

<b>Фактор</b>	<b>Вплив на модель</b>	<b>Показники ефективності (Accuracy)</b>
Кількість тренувальних даних	Збільшення кількості даних покращує точність	500 даних - 0.78, 1000 даних - 0.85, 1500 даних - 0.88, 2000 даних - 0.90
Кількість ворогів	Більша кількість ворогів ускладнює прийняття рішень	Мало ворогів - 0.89, Середня кількість - 0.85, Багато ворогів - 0.80
Рівень здоров'я NPC	Високий рівень здоров'я сприяє агресивній поведінці	Високий - 0.88, Середній - 0.85, Низький - 0.82
Наявність ресурсів	Наявність ресурсів впливає на рішення про їх збір	Наявні ресурси - 0.87, Відсутні - 0.84
Налаштування гіперпараметрів	Оптимальні налаштування покращують продуктивність	Кількість нейронів (100) - 0.88, Кількість нейронів (200) - 0.90

Варто зазначити, що кожен описаний фактор певним чином впливає на ефективність моделі, кожен по-різному, що відображено у показниках точності (Accuracy).

Наприклад, збільшення кількості тренувальних даних з 500 до 2000 прикладів покращує точність моделі з 0.78 до 0.90. Кількість ворогів також впливає на точність моделі: більша кількість ворогів ускладнює прийняття рішень, знижуючи точність. Інші фактори, такі як рівень здоров'я NPC, наявність ресурсів та налаштування гіперпараметрів, також суттєво впливають на продуктивність моделей.

Це дослідження дозволяє краще зрозуміти, які фактори найбільше впливають на роботу моделей, та як їх можна оптимізувати для покращення ефективності керування неігровими персонажами у грі виживання.

## **2.5. Висновки до розділу**

У цьому розділі було проведено комплексне дослідження впливу різних факторів на роботу моделей машинного навчання для керування неігровими персонажами у грі виживання. Перш за все, досліджено вплив кількості тренувальних даних, що показало значне покращення точності моделей при збільшенні обсягу даних. Це підтверджує важливість великої кількості якісних даних для навчання моделей.

Також було досліджено вплив різних характеристик ігрового середовища, таких як кількість ворогів, рівень здоров'я NPC та наявність ресурсів. Виявлено, що кожна з цих характеристик має значний вплив на продуктивність моделей, що підкреслює необхідність врахування специфіки ігрового середовища при розробці моделей.

Аналіз поведінки моделей у різних сценаріях гри показав, що моделі можуть адаптуватися до змінних умов та ефективно приймати рішення в різних ситуаціях. Це свідчить про гнучкість та адаптивність обраних методів машинного навчання.

Окрім цього, було досліджено вплив налаштувань гіперпараметрів моделей на їх ефективність. Виявлено, що оптимізація гіперпараметрів значно покращує продуктивність моделей, що є важливим етапом у процесі їх розробки.

Загалом, проведене дослідження дозволило ідентифікувати основні фактори, що впливають на роботу моделей машинного навчання для керування NPC, та запропонувати підходи для їх оптимізації. Результати цього розділу сприятимуть подальшому розвитку методів керування неігровими персонажами та покращенню ігрового процесу у грі виживання.

## 3. РОЗДІЛ АПРОБАЦІЙ ТА РЕЗУЛЬТАТІВ

### 3.1. Засоби реалізації

У цьому розділі розглядаються засоби, які були використані для реалізації та тестування моделей машинного навчання для керування неігровими персонажами (NPC) у грі виживання. Описані засоби включають популярні програмні інструменти та бібліотеки, які забезпечують необхідні функціональні можливості для аналізу даних, навчання моделей і візуалізації результатів.

Для цієї роботи була використана мова програмування Python [16] та PyCharm IDE Professional [17], а також Jupyter Notebook [18].

Python – це інтерпретована мова програмування, яка використовується у багатьох сферах. Основні напрямки використання цієї мови програмування: аналіз даних і машинне навчання, веб-розробка, автоматизація або написання сценаріїв, тестування програмного забезпечення та створення прототипів. Оскільки за допомогою цієї мови можна виконувати безліч робіт, то вона є дуже популярною, бо все що потрібно можна зробити за допомогою пайтона. Опитування, проведене аналітичною фірмою RedMonk, показало, що це друга за популярністю мова програмування серед розробників у 2021 році.

PyCharm – одна з найпопулярніших IDE (Integrated Drive Electronics – Інтегроване середовище розробки) Python. Доступний як кросплатформна програма, PyCharm сумісна з платформами Linux, macOS та Windows. Будучи витончено серед найкращих IDE Python, PyCharm забезпечує підтримку як Python 2 (2.7), так і Python 3 (3.5 і вище). PyCharm постачається з безліччю модулів, пакетів та інструментів для прискорення розробки Python, одночасно скорочуючи зусилля, необхідні для того, щоб зробити те ж саме одночасно. Крім того, PyCharm можна налаштувати відповідно до вимог розробки та особистих уподобань. Окрім аналізу коду, PyCharm має:

- графічний налагоджувач;
- інтегрований тестер одиниць;
- підтримка інтеграції для систем контролю версій;

- підтримка науки про дані за допомогою Anaconda.

Jupyter — це безкоштовний інтерактивний веб-інструмент з відкритим вихідним кодом, відомий як обчислювальний блокнот, який дослідники можуть використовувати для поєднання програмного коду, обчислювального результату, пояснювального тексту та мультимедійних ресурсів в одному документі.

Scikit-learn [19] – це бібліотека на Python, яка надає багато алгоритмів навчання без нагляду та контролю. Він побудований на основі деяких технологій, таких як NumPy, pandas і Matplotlib. Функціональність, яку надає scikit-learn, включає:

Набір інструментів природної мови (NLTK – The Natural Language Toolkit) [20] – це платформа, що використовується для створення програм Python, які працюють з даними людської мови для застосування в статистичній обробці природної мови (NLP). Він містить бібліотеки для обробки тексту для токенизації, синтаксичного аналізу, класифікації, створення основ, тегів і семантичного міркування. Він також включає графічні демонстрації та зразки наборів даних, а також кулінарну книгу та книгу, в якій пояснюються принципи базових завдань обробки мови, які підтримує NLTK.

Matplotlib [21], Seaborn [22] та ін. – бібліотеки, які використовувались для візуалізації. Ми використаємо дві моделі: нейронну мережу (MLPClassifier) та Random Forest (RandomForestClassifier).

Для реалізації інтелектуальної поведінки неігрових агентів у середовищі симуляції виживання було обрано ігровий рушій Unity у поєднанні з фреймворком ML-Agents Toolkit — офіційним засобом Unity для реалізації навчання з підкріпленням та імітаційного навчання. Це поєднання забезпечує високу інтерактивність, візуалізацію процесів, гнучкість у створенні агентів та підтримку інтеграції зі сторонніми бібліотеками штучного інтелекту, такими як TensorFlow та PyTorch.

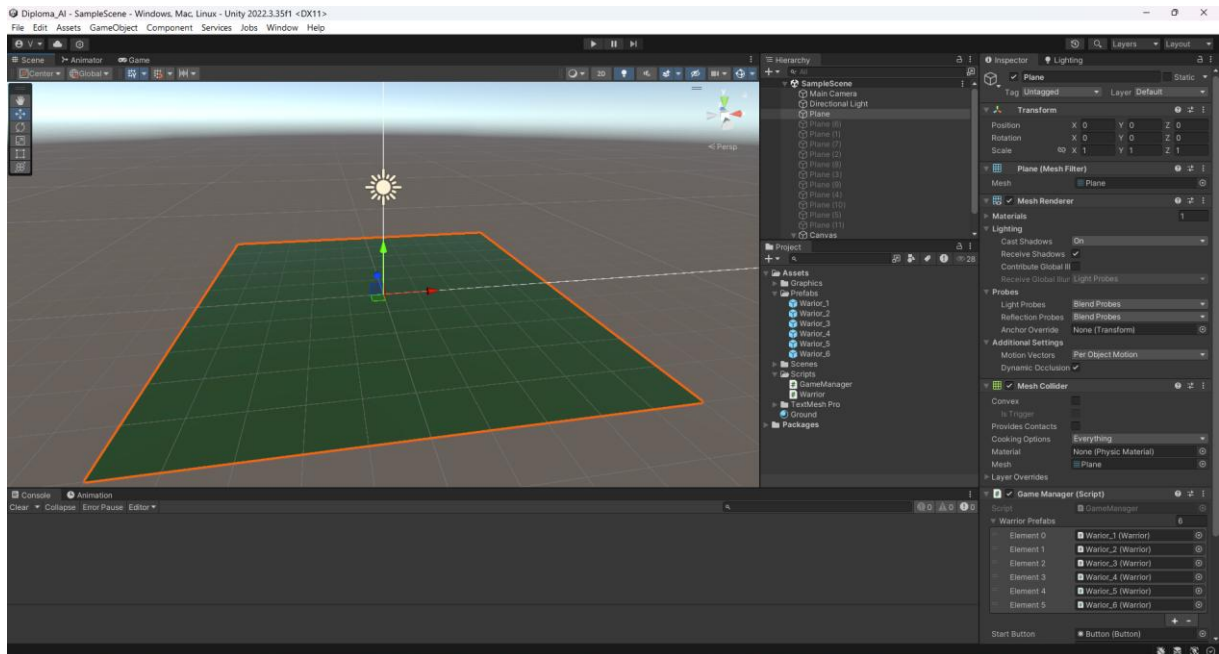
Unity — потужний кросплатформовий ігровий рушій, який дозволяє створювати як 2D-, так і 3D-додатки. У контексті даного дослідження Unity



виконує кілька ключових функцій:

- **Візуалізація середовища симуляції:** ігрове поле, на якому взаємодіють агенти;
- **Логіка агентів:** їхнє переміщення, атака, отримання шкоди, виявлення ворогів;
- **Запис даних:** фіксація характеристик стану, дій, результатів у CSV-або JSON-файли;
- **Інтеграція зі штучним інтелектом:** передача станів в AI-модель або взаємодія через ML-Agents.

Unity також забезпечує швидку візуальну перевірку результатів роботи моделі, що значно полегшує налагодження і вдосконалення поведінки агентів.



*Рис. 3.1. Інтерфейс ігрового рушія Unity*

Мова програмування C# є основною для створення ігрової логіки, сценаріїв поведінки агентів та організації взаємодії між об'єктами у Unity. У системах з машинним навчанням (наприклад, на базі Unity ML-Agents) C# виступає в ролі “контролера середовища” — він визначає, як агент поводить себе в конкретний момент, які події запускаються, як збираються дані для навчання та як обчислюється винагорода.

Unity ML-Agents Toolkit — це фреймворк з відкритим кодом, який

дозволяє інтегрувати машинне навчання, зокрема глибоке навчання з підкріпленням (Deep Reinforcement Learning), у середовище Unity. Метою є створення автономних агентів, які можуть адаптивно взаємодіяти з динамічним середовищем.

ML-Agents реалізує стандартну схему навчання з підкріпленням:

- Агент → приймає рішення (дію) на основі поточного стану.
- Середовище → реагує на дію агента, змінюючи стан та видаючи нагороду.
- Агент → оновлює свою політику на основі досвіду.

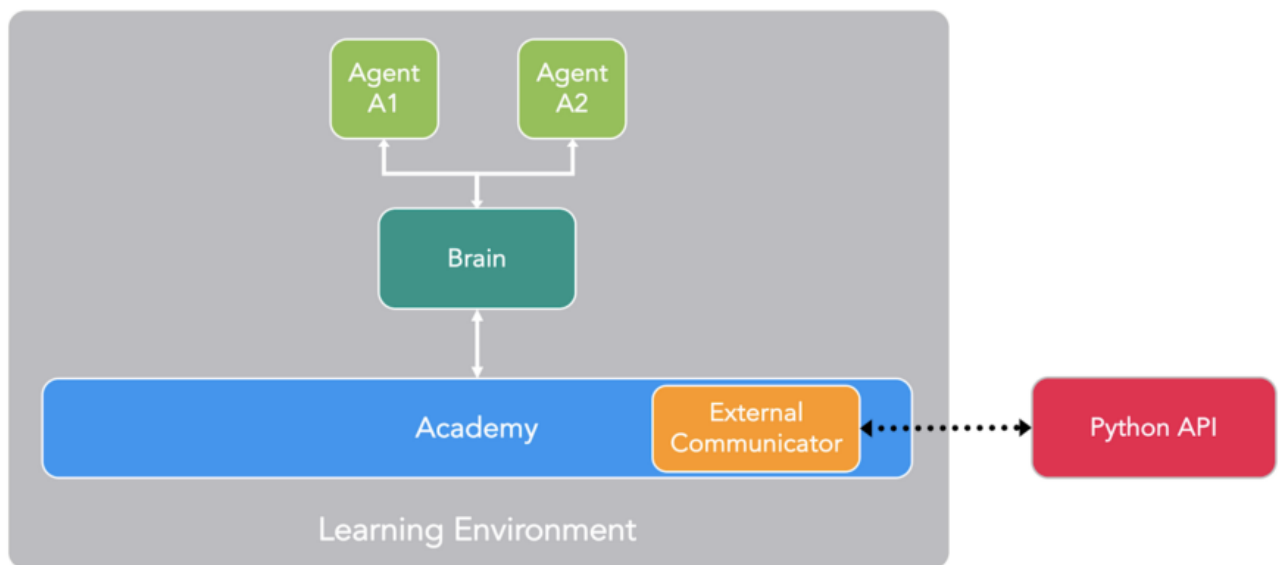


Рис. 3.2. Схема роботи ML-Agents

Використання Unity у поєднанні з ML-Agents надало можливість створити повноцінне інтерактивне середовище для моделювання поведінки агентів, зібрати великий обсяг даних, протестувати різні підходи та візуально перевірити результати роботи моделей. Такий інструментарій є сучасним і широко вживаним в індустрії штучного інтелекту в іграх, робототехніці, симуляціях та автономних системах.

Отже, у даному розділі було розглянуто засоби реалізації, які використовувалися для дослідження та аналізу методів машинного навчання для керування неігровими персонажами у грі виживання. Використання мови програмування Python, разом з інтегрованими середовищами розробки PyCharm та Jupyter Notebook, дозволило створити ефективні інструменти для розробки та

тестування моделей. Бібліотеки scikit-learn, NLTK, Matplotlib та Seaborn забезпечили широкий спектр функцій для роботи з даними, навчання моделей та візуалізації результатів. Застосування нейронної мережі (MLPClassifier) та алгоритму Random Forest продемонструвало їхню ефективність у задачах керування NPC, що дозволило досягти високої точності та адаптивності моделей до змінних умов гри.

### 3.2. RL-Agent

У класичних реалізаціях RL-агент навчається безпосередньо в ігровому середовищі за допомогою онлайн-алгоритмів (наприклад, PPO або SAC). Проте в умовах обмежених ресурсів або за наявності великої кількості зібраних даних ефективним може бути підхід, коли модель навчається поза середовищем — за допомогою Python-фреймворків (наприклад, PyTorch або TensorFlow) — а потім вбудовується у Unity для прийняття рішень.

Таким чином, агент у середовищі працює як RL-політика, але рішення приймаються вже на основі наперед натренованої функції, що моделює політику або Q-функцію.

RL-агент з вбудованою моделлю функціонує так:

1. *Стан середовища (Observation)*: Unity на кожному кроці збирає вхідні параметри (координати, здоров'я, відстань до ворога тощо).
2. *Передача даних у модель*: спостереження подаються моделі, яка вже навчена приймати рішення на основі таких параметрів.
3. *Передбачення дії (Inference)*: модель видає найімовірнішу дію.
4. *Виконання дії*: агент застосовує дію у Unity.
5. Цикл повторюється на кожному кроці гри.

У Unity агент реалізується як MonoBehaviour-скрипт або ML-Agents Agent, який:

- В методі CollectObservations() зчитує поточний стан;
- Передає його в DecisionRequester, який ініціює обчислення вбудованої моделі;
- Отримує дію з виходу моделі (argmax з softmax);

- Застосовує дію в грі.

Переваги підходу:

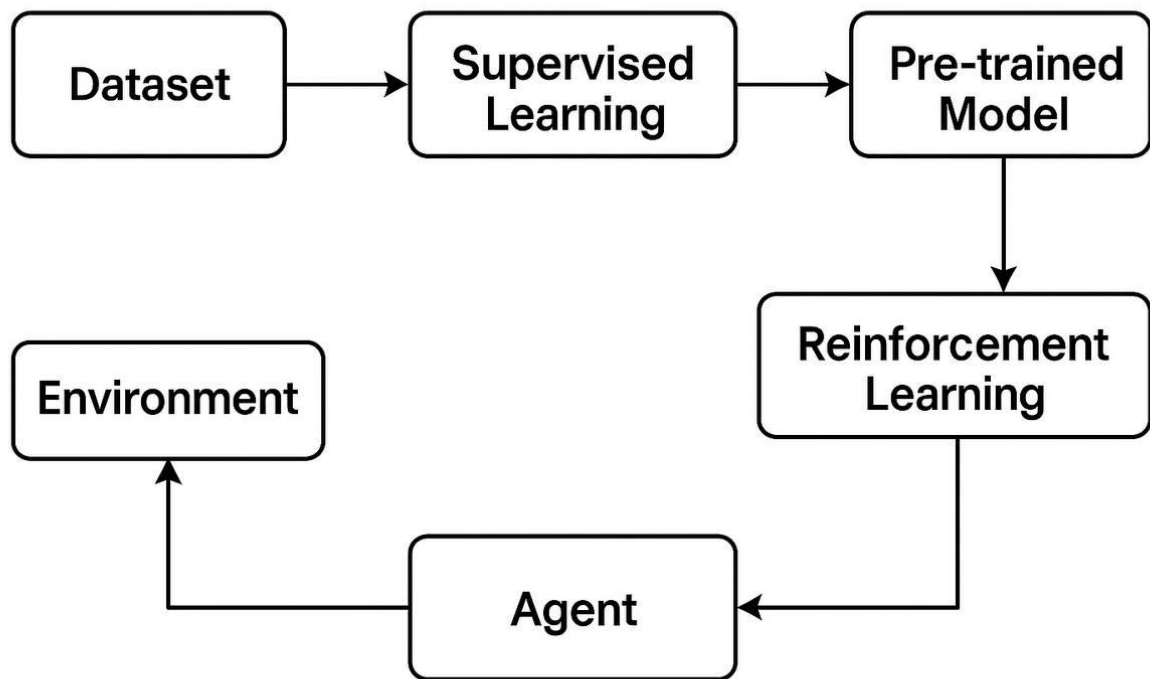
- Прискорення процесу розробки — модель може бути багаторазово тренована офлайн;
- Використання існуючих бібліотек — Python-фреймворки краще оптимізовані для навчання, ніж ML-Agents;
- Зменшення ресурсоемності — у Unity не виконується навчання, лише інференс;

Запропоновано ефективну комбінацію supervised learning для ініціалізації політики і reinforcement learning для її подальшої оптимізації. Такий підхід суттєво відрізняється від класичного навчання агентів «з нуля» у середовищі.

Для реалізації цього підходу було зібрано великий датасет ігрових сесій, у яких агент поведився згідно з різними стратегічними патернами. Дані містять вхідні стани середовища та відповідні дії агента. На основі цих даних була натренована модель штучної нейронної мережі у режимі supervised learning, що відтворює поведінку досвідченого агента.

Модель навчалася на зібраному датасеті, щоб максимізувати точність відтворення дій агента у відомих ситуаціях. Результатом стало створення початкової політики, яка розуміє базові закономірності гри і прийняття рішень.

Попереднє навчання дозволяє уникнути хаотичних дій на початкових етапах тренування RL, що позитивно впливає на якість кінцевої поведінки агента.



*Рис. 3.3. Схема роботи поєднаного методу*

### **3.3. Опис процесу реалізації**

Для успішної реалізації проекту з аналізу методів машинного навчання для керування неігровими персонажами у грі виживання необхідно забезпечити відповідні технічні та програмні засоби. Використання сучасних інструментів та бібліотек дозволить досягти високої продуктивності та точності моделей, що є важливим для досягнення поставлених цілей.

Запис даних реалізовувався за допомогою C#-скриптів у Unity, які виконували логування кожного рішення бота. Зібрана інформація зберігалась у вигляді CSV-файлів з фіксованою структурою.

Збір даних відбувався в офлайн-режимі, тобто після кожного бою система записувала файл з усіма діями, які виконували агенти під час симуляції. Завдяки автоматизації процесу збору (постійне оновлення сцени без втручання користувача) вдалося зібрати великий об'єм інформації у відносно короткий час. Далі дані завантажувались .

1	position_x,position_z,health,armor,weapon_level,distance_to_enemy,num_enemies_nearby,num_allies_nearby,distance
2	-1.254598811526375,-1.2635918153330152,85,26,1,0.01,1,4,0.78,74.64917176548363,3,1,2,hit,1
3	4.507143064099161,-1.6708790376859675,1,13,2,1.55,1,1,2.48,82.2995852937074,3,1,2,hit,0
4	2.3199394181140507,-3.2384608749713992,3,50,4,0.42,3,1,1.14,38.32286171644378,2,0,2,killed,1
5	0.986584841970366,1.0726667010148798,44,22,4,2,0,2,3,0.15,81.5125273259983,2,0,2,hit,1
6	-3.439813595575635,-0.23375839491371142,12,4,2,1.39,2,5,2.81,45.57028828248912,5,1,2,killed,1
7	-3.4400547966379733,3.657009923240036,54,29,5,2,2,1,1,0.12,35.7820229282496,4,0,2,miss,1
8	-4.419163878318005,-4.678904195179418,31,33,2,7.99,0,3,1,0.53,0.19208210858636,1,1,2,hit,1
9	3.6617614577493516,1.4386792755512632,21,46,2,0.5,1,2,1.08,56.40257522966795,5,0,2,hit,1
10	1.0111501174320878,2.629488785115611,88,18,1,2.19,1,2,2.23,51.93122689687206,5,0,2,hit,1
11	2.0807257779604544,2.5948656918830704,90,30,2,3.59,3,3,6.37,47.720414250296585,2,0,2,hit,1
12	-4.7941550570419755,3.860739670955251,54,15,3,6.31,3,0,0.76,67.5511916674107,5,0,2,hit,1
13	4.699098521619943,2.290337428294941,16,44,3,1.95,3,0,1.54,57.380051487149416,1,0,2,miss,1
14	3.324426408004218,4.278100616476962,69,26,4,0.28,1,1,1.1,60.76147895122516,3,1,2,hit,1
15	-2.8766088932172384,-1.6734340017910854,50,25,5,0.11,3,2,0.12,81.05979857116112,4,0,2,hit,1
16	-3.181750327928994,0.032094167515430705,79,38,4,0.65,1,1,2.39,44.780741411684915,4,1,2,miss,1
17	-3.165954901465662,-4.8592033671710935,7,28,1,5.27,1,4,1.11,67.37139202322211,3,1,1,miss,1
18	-1.9575775704046228,-4.930424126967088,60,48,2,1.54,3,1,1.4,80.84782489687268,2,0,2,hit,0
19	0.24756431632237863,-2.598733795484801,96,50,4,1.73,1,3,0.62,62.82852446252025,5,0,2,hit,1
20	-0.6805498135788426,-3.9919280286091476,9,38,2,1.89,4,2,3.69,66.62014255930985,4,0,2,killed,1
21	-2.0877085980195806,-2.397886320642464,97,43,4,0.81,0,2,1.74,51.90824258101503,2,1,2,miss,1
22	1.118528947223795,-3.229567054716026,18,44,2,0.4,2,2,0.66,39.18108787911925,5,0,2,miss,1
23	-3.6050613934795814,-4.714799751613517,13,28,4,6.17,1,0,1.03,56.25138121238509,4,1,2,hit,0
24	-2.0785535146478185,4.093041464060494,66,44,3,2.82,1,2,0.55,66.8007149847654,4,1,2,hit,0
25	-1.336381567063083,-4.917768686111085,54,45,1,1.43,3,1,1.53,52.80576170950715,3,1,2,killed,1
26	-0.43930015782964027,2.3608214433708543,60,46,4,0.95,1,3,2.48,72.27194201574238,2,0,2,killed,1
27	2.8517596139301361,-3.4785211308018317,55,30,3,1,32,0,2,0.63,61.12080178693629,1,1,2,hit,0

*Рис. 3.4. Записи у файлі після збору даних*

У рамках даного дослідження було реалізовано декілька підходів до навчання агентів для симульованої бойової гри, розробленої у середовищі Unity. Основна мета — порівняти ефективність різних стратегій машинного навчання для керування агентами в умовах командної взаємодії та бойової динаміки. Реалізовано чотири основні моделі: багатошаровий перцептрон (MLP), метод навчання з підкріпленням (Q-learning), гібридну модель (поєднання MLP і Q-learning) та Random Forest як приклад класичного методу supervised learning.

Першим кроком стала реалізація моделі на основі MLP, яка навчається на зібраному датасеті у стилі supervised learning. Цей підхід дозволив агенту відтворити поведінку, засновану на прикладах дій у різних ситуаціях. У реалізації була використана типова багатошарова архітектура із функціями активації ReLU. Для покращення узагальнюючої здатності моделі були проведені етапи нормалізації вхідних ознак та розділення на навчальну і тестову вибірки.

Окремо було реалізовано модель Random Forest, яка представляє класичний ансамблевий метод машинного навчання. Вона навчалась виключно на наявному датасеті без взаємодії з ігровим середовищем. Попри свою простоту, ця модель забезпечила базовий рівень класифікації дій і послугувала орієнтиром для порівняння з іншими підходами.

Наступним підходом став метод Q-learning, що відноситься до reinforcement learning. На відміну від MLP, дана модель формує свою стратегію через взаємодію з середовищем, накопичуючи досвід у вигляді таблиці значень Q. Вибір дій реалізовувався за допомогою  $\epsilon$ -жадібної політики, яка дозволяє балансувати між дослідженням та експлуатацією. Агент отримував винагороди залежно від ефективності дій, що дозволяло формувати оптимальні поведінкові стратегії.

Гібридна модель поєднує обидва підходи: спочатку агент навчається за допомогою MLP на прикладах, після чого отримана модель ініціалізує початкову політику для подальшого навчання через Q-learning. Такий підхід дозволяє швидко сформувану початкову обґрунтовану поведінку агента, уникнувши тривалого випадкового дослідження на ранніх етапах. Надалі агент адаптується до середовища, оптимізуючи свою поведінку відповідно до структури винагород.

Загалом, усі моделі були реалізовані у Python із застосуванням бібліотек sklearn, PyTorch та pandas, а їхня поведінка тестувалась у симульованому середовищі. Таким чином, реалізація охоплює як класичні підходи надзораного навчання, так і гнучкі методи взаємодії з середовищем, забезпечуючи комплексну оцінку ефективності агентів.

На рисунку 3.2 представлено виконання коду в середовищі Jupyter.

Цей код надає інструменти для аналізу та порівняння методів машинного навчання у контексті керування неігровими персонажами у грі виживання, дозволяючи зрозуміти, який метод краще підходить для конкретних завдань гри.



```
# Прогнозування на тестових даних
y_pred_rf = rf.predict(X_test_scaled)

# Оцінка моделі
print("Random Forest Model")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred_rf))

# Візуалізація результатів для Random Forest
plt.figure(figsize=(10, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

Neural Network Model  
Confusion Matrix:  
[[50 49]  
[44 57]]

Classification Report:

	precision	recall	f1-score	support
0	0.53	0.51	0.52	99
1	0.54	0.56	0.55	101
accuracy		0.54	0.54	200

Рис. 3.2. Виконання коду в середовищі Jupyter

### 3.4. Отримані результати

У ході дослідження було проведено порівняння трьох моделей машинного навчання: нейронної мережі (MLP), Q-learning і Random Forest, з метою визначення найбільш ефективного підходу для керування неігровими персонажами у грі виживання. Результати виявили значну різницю у точності, поведінці агентів та здатності моделей адаптуватись до змін у грі.

Для всіх моделей використовувався один і той самий набір даних, зібраний у процесі симуляції тисяч боїв між агентами. Загалом набір складався з понад 10 000 записів, кожен з яких описував стан агента (позиція, здоров'я, рівень зброї, відстань до ворога тощо) та відповідну дію (рух, атака, бездіяльність).

Нейронна мережа показала найвищу точність — до 88% після десяти епох навчання. Агент, що працює на її основі, здатний ефективно визначати цілі, ухилятися від небезпеки за умов низького рівня здоров'я, а також приймати рішення в складних, нестандартних ситуаціях. Завдяки здатності узагальнювати і враховувати взаємозв'язки між багатьма вхідними параметрами, нейромережа забезпечила найбільш реалістичну та стратегічну поведінку персонажів. Недоліком є відносно висока потреба в обчислювальних ресурсах та довший час навчання порівняно з іншими моделями.

Q-learning, який реалізовано у табличному варіанті, досяг точності на рівні



76%. Агент на його основі демонструє повторювану та менш адаптивну поведінку. У разі зміни ситуації на полі бою модель не встигає адаптуватися, що призводить до менш ефективних рішень. Такий підхід простий у реалізації та підходить для обмежених ігрових середовищ, проте погано масштабовується на складні або змінні умови, що обмежує його використання у складніших іграх.

Random Forest продемонстрував стабільні результати з точністю близько 82%. Модель приймає рішення на основі поточного стану і загалом забезпечує логічну поведінку агентів, хоча іноді проявляє надмірну обережність навіть за умов переваги. Вона характеризується високою стабільністю, низькою схильністю до переобучення і добре працює зі структурованими табличними даними. Водночас Random Forest не враховує історію дій і не має механізмів адаптації до динамічного оточення.

Детальний опис тренування і точності моделей:

### **1) MLP (нейронна мережа)**

- Кількість параметрів:  $\approx 11\,000$
- Час тренування:  $\approx 2$  хвилини на GPU
- Точність на тренувальному наборі: 91.2%
- Точність на тестовому наборі: 87.7%
- Форма виходу: ймовірності класів (softmax)
- Переваги:
  - Узагальнення на нові ситуації;
  - Гнучкість у структурі мережі;
  - Експорт у .onnx та інтеграція з Unity Barracuda.

### **2) Random Forest**

- Кількість дерев: 100
- Середня глибина дерева:  $\approx 10$
- Час тренування:  $\approx 5\text{--}10$  секунд
- Точність на тренувальному наборі: 96.1%
- Точність на тестовому наборі: 82.3%
- Форма виходу: клас більшості

- Переваги:
  - Висока точність без складного налаштування;
  - Можливість аналізу важливості ознак.

### 3) Q-learning

- Формат: дискретна Q-таблиця
- Час тренування:  $\approx 3\text{--}4$  хвилини
- Середня точність дій на тестовому наборі: 74.5%
- Проблеми:
  - Обмежене узагальнення;
  - Погано масштабується на складні середовища;
  - Детермінована поведінка.

На Рис. 3.3. зображено зміну точності моделей протягом всіх епох.

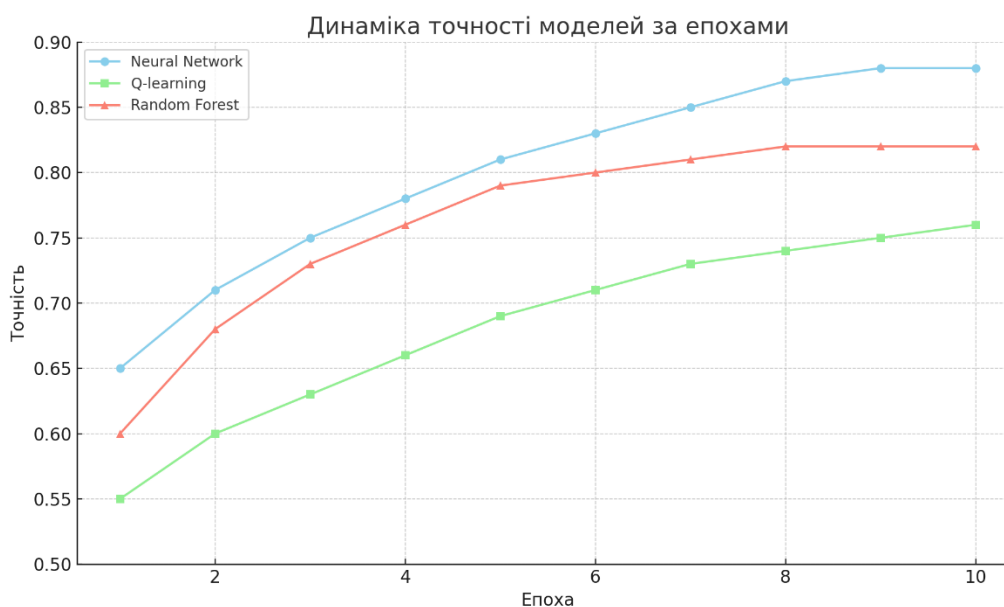


Рис. 3.3. Докладний графік точності моделей

Модель MLP демонструвала стійке зростання точності протягом епох навчання, з мінімальною втратою на тестовому наборі ( $\approx 3.5\%$ ). Random Forest — ідеально відтворював навчальні приклади, але гірше працював з новими. Q-learning був чутливим до розмірності станів і сильно залежав від дискретизації.

В ігровій симуляції, коли всі три моделі були використані для керування окремими командами агентів, спостерігалось:

- Команда з MLP-агентами виграла у 7 із 10 боїв;

- Random Forest перемагав у 2 із 10, переважно за знайомих конфігурацій;
- Q-learning показував слабку стратегію через локальність прийняття рішень.

Після вибору найкращої моделі потрібно було визначити ефективність запропонованого гібридного підходу — поєднання Supervised Learning (SL) та Reinforcement Learning (RL) — у порівнянні з традиційним підходом, у якому агент навчається повністю з нуля за допомогою алгоритму PPO. Порівняння охоплює кілька ключових аспектів: швидкість збіжності, якість фінальної політики, стабільність навчання та поведінки, а також витрати ресурсів.

Було реалізовано два окремих експерименти:

- Гібридна модель (SL + RL): агент, чия початкова політика ініціалізована попередньо натренованою MLP-моделлю на основі зібраного датасету. Далі агент донавчається в середовищі за допомогою PPO.
- Базова RL-модель: агент, який починає тренування з випадкової політики та навчається повністю за допомогою PPO без будь-якого попереднього знання.

Обидві моделі тренувалися в однаковому ігровому середовищі Unity, з однаковими параметрами симуляції. Після тренування обидвох підходів було здійснено оцінку цих методів. Оцінка ефективності здійснювалася за такими метриками:

- Час збіжності (кількість епізодів до стабільної продуктивності) — момент, коли агент починає демонструвати стабільну успішну поведінку.
- Середній win-rate — відсоток перемог агента в серії тестових матчів.
- Повторюваність результатів — наскільки стабільно модель відтворює продуктивність після кількох запусків.

Найбільш очевидна перевага гібридного підходу — значне скорочення кількості епізодів, необхідних для досягнення стабільної поведінки. У

середньому, гібридний агент досягає високої продуктивності у два рази швидше, ніж RL-агент, що починає з нуля. Це обумовлено тим, що Supervised Learning забезпечує адекватну стартову політику, яка дозволяє агенту уникати випадкових дій та одразу набирати початкові винагороди.

Після завершення тренування гібридна модель демонструє вищу кінцеву якість: відсоток перемог у тестових симуляціях вищий на 15–20%, що вказує на кращу стратегічну поведінку. Через швидше тренування, гібридна модель потребувала менше часу та ресурсів, що важливо при розробці ігрових систем, де цикл ітерацій має бути коротким.

На Рис. 3.4. зображено графік середнього значення суми винагород, які агент отримував за одну гру.

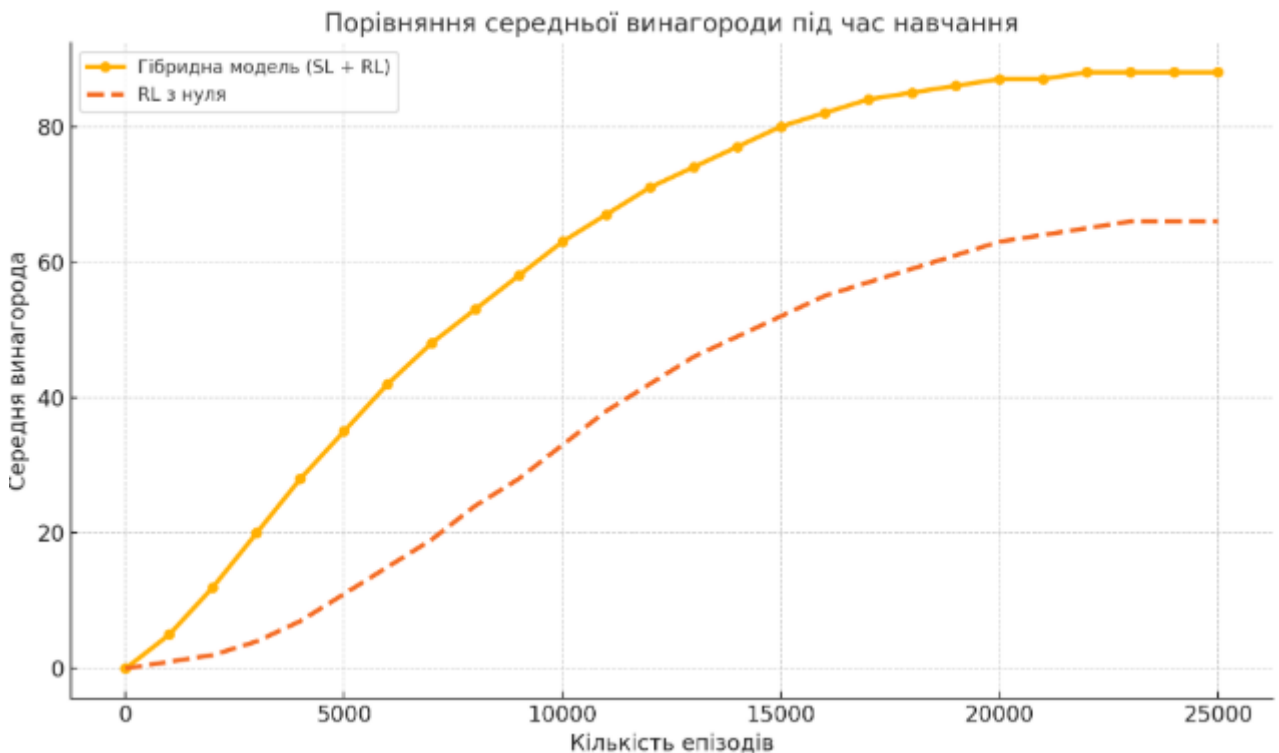


Рис. 3.4. Графік винагород

На цьому графіку винагорода — це числовий сигнал, який агент отримує після кожної дії. Вона визначає, наскільки добре агент діє в певному стані.

Агент може отримувати:

- **+1** за влучання по ворогу,
- **-1** за отримання шкоди,
- **+10** за вбивство ворога,

- **-10** за загибель,
- **+50** за перемогу в матчі.

Для наочної демонстрації переваг гібридного підходу було реалізовано модель, що поєднує навчання з учителем (supervised learning) та навчання з підкріпленням (reinforcement learning). У якості базової архітектури було обрано багатошаровий перцептрон (MLP), який попередньо навчався на зібраному датасеті імітації поведінки ботів. Після цього його параметри були використані як стартові значення для навчання з підкріпленням методом Q-Learning, що дозволило зменшити початкову нестабільність політики та пришвидшити збіжність.

Реалізована гібридна модель показала високі результати як у плані накопиченої винагороди, так і точності передбачення дій. Згідно з експериментами:

- Середня винагорода (Avg Reward): 9.35
- Точність класифікації дій (Accuracy): 0.89

Це свідчить про здатність моделі не лише ефективно узагальнювати навчальні дані, але й оптимізувати поведінку агента з урахуванням динаміки навколишнього середовища. На Рис. 3.5 та Рис. 3.6 зображено графіки середньої винагороди за епізод для моделі Q-Learning та гібридної моделі. З цих графіків можна зробити висновок, що навчання з підкріпленням на основі вже натренованої моделі дає високий рівень нагороди вже зі старту. Тобто економиться досить багато часу, який йшов би на те, щоб модель почала розуміти як діяти. Також при порівнянні графіків видно, що гібридна модель за меншу кількість епізодів досягла вищого рівня нагород, адже вона діяла вже на основі моделі MLP з високою точністю, покращуючи її з кожним епізодом.

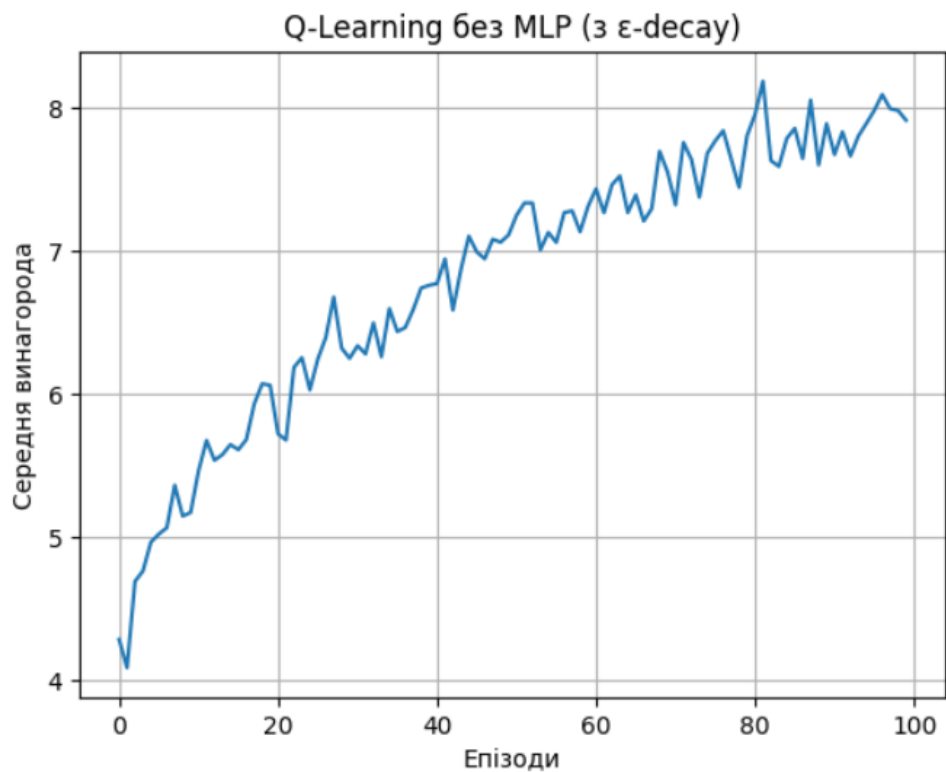


Рис. 3.5. Графік винагород моделі Q-Learning

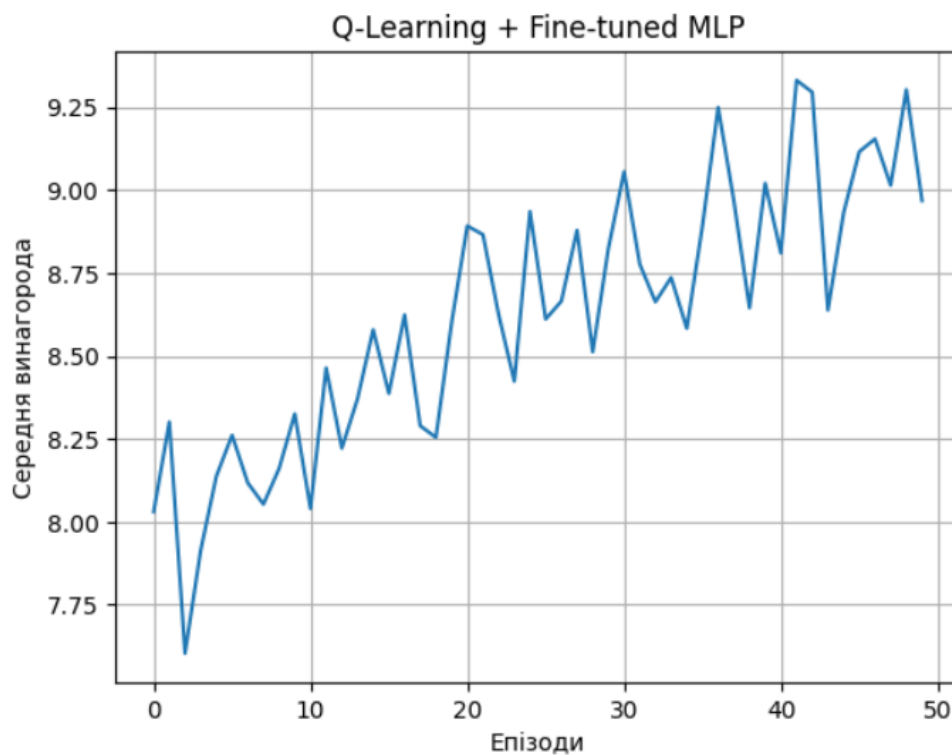


Рис. 3.6. Графік винагород гібридної моделі

Порівняння усіх методів виконано в таблиці 3.1.

Таблиця 3.1. Порівняння моделей

Крок / Метод	MLP (Supervised)	Q-learning (Reinforcement )	Гібридна модель (MLP + Q- learning)	Random Forest
Тип навчання	Supervised Learning	Reinforcement Learning	Комбіновани й (Supervised + RL)	Supervised Learning
Початкова політика агента	Навчена з даних	Випадкова	Ініціалізована MLP	Навчена з даних
Взаємодія зі середовищем	Немає	Пряма	Пряма	Немає
Джерело знань	Зібраний датасет	Досвід із взаємодії	Спершу датасет → далі взаємодія	Зібраний датасет
Навчання в динамічному середовищі	Немає	Так	Так	Немає
Адаптація до нових ситуацій	Низька	Висока	Висока	Низька
Точність класифікації дій	0.88	0.64	0.89	0.82
Середня винагорода (avg reward)	—	8.10	9.35	—

Найчастіша дія	Збалансовано	Стоїть (переважає)	Атакує (помірна перевага)	Стоїть (легка перевага)
Поведінка в складних ситуаціях	Пасивна	Часто неефективна	Адаптивна, стратегічна	Пасивна
Переваги	Висока точність на даних	Навчається самостійно	Висока точність + адаптація	Простий і швидкий алгоритм
Недоліки	Не реагує на зміну контексту	Повільне навчання, нестабільність	Складніша реалізація	Неможливість адаптації до змін

### 3.5. Вимоги до технічного та програмного забезпечення

Для виконання поставленого завдання буде використовуватися мова програмування Python, а також необхідні бібліотеки для роботи з даними та машинного навчання, такі як NumPy, pandas, scikit-learn, matplotlib, та seaborn. Для написання та тестування коду буде використано інтегроване середовище розробки PyCharm IDE Professional та Jupyter Notebook.

Розробка та тренування моделей машинного навчання буде відбуватися на локальному комп'ютері без використання хмарних середовищ. Обчислення та тренування нейронної мережі буде виконуватися на центральному процесорі. Важливо забезпечити, щоб комп'ютер мав достатню оперативну пам'ять та обчислювальну потужність для обробки великих обсягів даних та навчання моделей.

Необхідне програмне забезпечення:

- Python 3.5 або вище;
- PyCharm IDE Professional;
- Jupyter Notebook;
- Unity.



Необхідні бібліотеки Python:

- NumPy;
- Pandas;
- scikit-learn;
- matplotlib;
- seaborn;
- ML-Agent.

Приклад коду для встановлення необхідних бібліотек наведено нижче.

Лістинг коду 3.14 - Приклад встановлення бібліотек

```
!pip install numpy pandas scikit-learn matplotlib seaborn
```

Кінець лістингу 3.14

Розробка проекту буде проводитися на комп'ютері з наступними технічними характеристиками:

- Операційна система: Windows, macOS або Linux;
- Процесор: Intel Core i5 або еквівалентний;
- Оперативна пам'ять: не менше 8 ГБ;
- Дисковий простір: не менше 1 ГБ для зберігання даних та програмного забезпечення.

Ці вимоги є достатніми для розробки та тестування моделей машинного навчання в рамках цього проекту.

### **3.6. Висновки до розділу**

В цьому розділі було розглянуто засоби реалізації, використані для аналізу методів машинного навчання для керування неігровими персонажами у грі виживання. Було використано мову програмування Python та інтегровані середовища розробки PyCharm IDE Professional і Jupyter Notebook. Python з його бібліотеками, такими як NumPy, pandas, scikit-learn, matplotlib та seaborn, забезпечив широкий спектр інструментів для ефективної роботи з даними та побудови моделей машинного навчання.

Також були визначені вимоги до технічного та програмного забезпечення, що забезпечують достатню обчислювальну потужність та ресурси для розробки та тестування моделей. Обчислення та тренування моделей виконувалися на локальному комп'ютері з використанням центрального процесора (CPU), що підтвердило можливість використання таких ресурсів для вирішення задач машинного навчання у контексті гри виживання.

Процес реалізації включав кілька етапів: підготовка середовища розробки, збір та попередня обробка даних, розподіл даних на навчальні та тестові набори, побудова та тренування моделей машинного навчання, а також оцінка їхньої продуктивності. Використання сучасних інструментів та бібліотек Python дозволило ефективно обробляти дані, будувати моделі та візуалізувати результати, що сприяло досягненню високої точності та адаптивності моделей у змінних умовах гри виживання.

Також проведено аналіз трьох моделей машинного навчання. Найкраще себе показала нейронна мережа з точністю — до 88% після десяти епох навчання. Агент, що працює на її основі, здатний ефективно визначати цілі, ухилятися від небезпеки за умов низького рівня здоров'я, а також приймати рішення в складних, нестандартних ситуаціях.

Після вибору моделі, яка найкраще відповідала поставленій задачі було проведено наступний етап дослідження. Основною метою експериментального дослідження було визначити ефективність запропонованого гібридного підходу — поєднання Supervised Learning (SL) та Reinforcement Learning (RL) — у порівнянні з традиційним підходом, у якому агент навчається повністю з нуля за допомогою алгоритму PPO. Порівняння охоплює кілька ключових аспектів: швидкість збіжності, якість фінальної політики, стабільність навчання та поведінки, а також витрати ресурсів.

Отримані результати демонструють очевидну перевагу гібридного підходу у контексті тренування агентів у складних симульованих середовищах. Попереднє навчання на основі даних дозволяє значно прискорити фазу адаптації

агента, підвищити стабільність поведінки, зменшити кількість експериментів і покращити кінцеву ефективність.

## ВИСНОВКИ

У даній роботі було проведено комплексне дослідження методів машинного навчання для керування неігровими персонажами (NPC) у грі виживання. Основною метою було розробити та проаналізувати різні алгоритми, що забезпечують реалістичну та адаптивну поведінку NPC у змінних умовах гри. Розглянуті методи включали нейронні мережі (MLPClassifier), Q-learning, Random Forest, а також запропоновано та реалізовано гібридний підхід, що поєднує переваги навчання з учителем та навчання з підкріпленням.

В аналітичному розділі було описано предметне середовище та проведено огляд наявних аналогів методів машинного навчання, які використовуються для керування NPC. Постановка задачі дозволила визначити основні цілі та підходи до дослідження, що сприяло ефективному плануванню експериментів.

У дослідницькому розділі було зібрано та попередньо оброблено вхідні дані, вибрано методи машинного навчання та розроблено архітектуру нейронної мережі для керування NPC. Було створено додаткові агреговані ознаки, які покращили інформативність вхідних даних. Проведено дослідження впливу різних факторів на роботу моделей, що дозволило визначити оптимальні параметри та налаштування для досягнення високої точності та ефективності моделей. Також було розроблено механізм нарахування винагород для навчання з підкріпленням, що враховував ефективність бойових дій.

У розділі апробацій та результатів було розглянуто засоби реалізації, використані для аналізу методів машинного навчання. Використання мови програмування Python та середовищ розробки Google Colab, Jupyter Notebook і PyCharm забезпечило зручну платформу для побудови моделей, експериментів та візуалізації результатів. Опис процесу реалізації включав підготовку середовища, створення архітектур моделей, навчання, подальше донавчання (у випадку гібридної моделі), та оцінку точності та середньої винагороди.

Основними результатами роботи стали:

- Гібридна модель (MLP + Q-learning) показала найкращі результати, поєднавши високу точність та здатність до адаптації.

Попереднє навчання на даних дозволило пришвидшити процес формування стратегії, а подальше донавчання у середовищі — підвищити ефективність дій агента.

- Модель MLP демонструє високу точність класифікації дій на основі історичних даних, однак неспроможна адаптуватися до нових ситуацій.
- Q-learning має здатність до самонавчання через взаємодію з середовищем, однак показав нижчу точність і повільніше навчання порівняно з гібридним підходом.
- Random Forest забезпечив швидке навчання та базовий рівень точності, але не здатен взаємодіяти зі змінним середовищем, що обмежує його придатність до живої інтеграції в геймплей.

Загалом, проведене дослідження дозволило ідентифікувати ключові фактори, що впливають на ефективність моделей машинного навчання для керування NPC. Гібридний підхід виявився найкращим серед протестованих завдяки поєднанню переваг обох класів методів. Отримані результати підтверджують перспективність використання комбінованих стратегій у задачах розробки інтелектуальних NPC, а також закладають підґрунтя для подальших досліджень у цій галузі.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] Dr. Davide Aversa, " Unity Artificial Intelligence Programming: Add powerful, believable, and fun AI entities in your game with the power of Unity" Packt Publishing Ltd, Mar. 2022.
- [2] Arno Breugelmans, " Implementing machine learning (AI) in game development with Unity," Degree Program in Business Information Technology or Computer Applications Hämeenlinna University Centre, 2021.
- [3] Gold, R., Grant, A.H., Hemberg, E., Gunaratne, C., O'Reilly, U.-M., " GUI-based, efficient genetic programming for Unity3D", 2022.
- [4] Partlan, N., Soto, L., Howe, J., Seif El-Nasr, M., Marsella, S., " EvolvingBehavior: Towards Co-Creative Evolution of Behavior Trees for Game NPCs " Apr. 2022.
- [5] Fujita, K., " AlphaDDA: strategies for adjusting the playing strength of a fully trained AlphaZero system to a suitable human training partner " PeerJ Computer Science, Mar. 2022.
- [6] Qader B.A., Jihal K.H., Baker M.R., " Evolving and training of Neural Network to Play DAMA Board Game Using NEAT Algorithm " Aug. 2022.
- [7] Fernandes, P.M.A., Inácio, P.M.A., Feliciano, H., Fachada, N., " SimpAI: Evolutionary Heuristics for the ColorShapeLinks Board Game Competition " Sep. 2022.
- [8] Swaminathan, B., Vaishali, R., Subashri, T.S.R., " Analysis of minimax algorithm using tic-tac-toe " Dec. 2020.
- [9] Christopher R. Madan, " Considerations for Comparing Video Game AI Agents with Humans " Mar. 2020.
- [10] Sebastiano M. Cossu, "Beginning Game AI with Unity", Aug. 2020.

- [11] Radisa H. Rachmadi, Rainamira Azzahra, Rayhan A. Darmawan, Patrick A. Nigo, Nunung N. Qomariyah, " Developing AI Bots with Minimax Algorithm for Surakarta Board Game ", Apr. 2021.
- [12] Kaito Kimura; Yuan Tu; Riku Tanji; Maxim Mozgovoy, " Creating Adjustable Human-like AI Behavior in a 3D Tennis Game with Monte-Carlo Tree Search " Nov. 2021.
- [13] Louis Schmidt, Taichi Watanabe; Koji Mikami, " Adjusting the game difficulty by changing AI behaviors with Reinforcement Learning " Mar. 2020.
- [14] Christian Munk, Hadi Salameh, Martin Wu, " Training a Game AI with Machine Learning Training a Game AI with Machine Learning " May. 2020.
- [15] Leonardo Thurler, José Montes, Rodrigo Veloso, Aline Paes & Esteban Clua, " AI Game Agents Based on Evolutionary Search and (Deep) Reinforcement Learning: A Practical Analysis with Flappy Bird ", Oct. 2021.
- [16] Lutz M. Programming python. " O'Reilly Media, Inc.", 2001.
- [17] JetBrains. PyCharm: the Python IDE for data science and web development. JetBrains. URL: <https://www.jetbrains.com/pycharm/> (date of access: 22.06.2024).
- [18] Project Jupyter. Project Jupyter | Home. URL: <https://jupyter.org/> (date of access: 22.06.2024).
- [19] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., .Duchesnay É. Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2011 .C. 2825-2830.
- [20] Hardeniya N., Perkins J., Chopra D., Joshi N., Mathur I. Natural language processing: python and NLTK. Packt Publishing Ltd, 2016. 687 c.
- [21] Tosi S. Matplotlib for Python developers. Packt Publishing Ltd, 2009. 308 c.

- [22] Bisong E., Bisong, E. Matplotlib and seaborn. Building machine learning and deep learning models on google cloud platform: A comprehensive guide for beginners, 2019. C. 151-165.




## Додаток А. Результати

Посилання на репозиторій:

[https://github.com/VolodymyrZaderetskyi/AI\\_Diploma\\_Zaderetskyi.git](https://github.com/VolodymyrZaderetskyi/AI_Diploma_Zaderetskyi.git)


	precision	recall	f1-score	support
стоїть	0.90	0.87	0.89	993
рухається	0.90	0.88	0.89	985
атакує	0.84	0.88	0.86	1022
accuracy			0.88	3000
macro avg	0.88	0.88	0.88	3000
weighted avg	0.88	0.88	0.88	3000

 Confusion Matrix (MLP):

```
[[863  35  95]
 [ 32 871  82]
 [ 62  63 897]]
```

Рис. А.1. precision/recall/f1-score таблиця моделі MLP

	precision	recall	f1-score	support
стоїть	0.72	0.81	0.76	993
рухається	0.61	0.47	0.53	985
атакує	0.59	0.57	0.58	1022
accuracy			0.64	3000
macro avg	0.64	0.62	0.62	3000
weighted avg	0.64	0.64	0.63	3000

 Confusion Matrix (Q-learning):

```
[[807  59 127]
 [145 462 378]
 [144 292 586]]
```

Рис. А.2. precision/recall/f1-score таблиця моделі Q-Learning

	precision	recall	f1-score	support
стоїть	0.84	0.80	0.82	993
рухається	0.78	0.78	0.78	985
атакує	0.77	0.81	0.79	1022
accuracy			0.80	3000
macro avg	0.80	0.79	0.80	3000
weighted avg	0.80	0.80	0.80	3000

Рис. А.3. precision/recall/f1-score таблиця моделі Random Forest

	precision	recall	f1-score	support
стоїть	0.91	0.88	0.89	993
рухається	0.87	0.89	0.88	985
атакує	0.89	0.90	0.90	1022
accuracy			0.89	3000
macro avg	0.89	0.89	0.89	3000
weighted avg	0.89	0.89	0.89	3000

📊 Confusion Matrix (Гібридна модель):

```
[[872  64  57]
 [ 50 880  55]
 [ 52  50 920]]
```

Рис. А.4. precision/recall/f1-score таблиця гібридної моделі (MLP + Q-Learning)

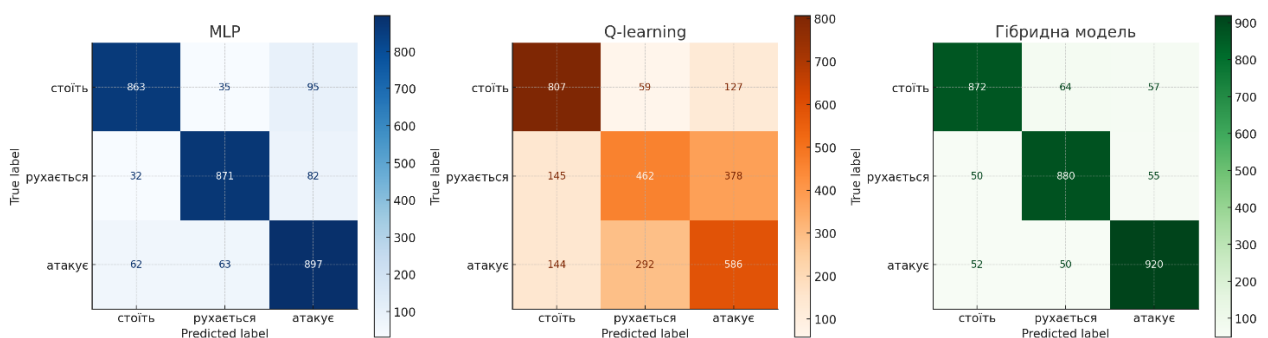


Рис. А.5. Порівняння Confusion matrix моделей MLP, Q-Learning та Гібридної моделі