

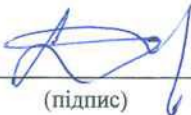
**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ І СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА
на тему:**

**РОЗРОБКА ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ ЗАСОБАМИ РУШІЯ
UNITY**

Здобувача вищої освіти першого (бакалаврського) рівня,
4 курсу, КБ-19-1 групи, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»



(підпис)

Якубовського Василя Сергійовича
(прізвище, ім'я, по батькові)

Науковий керівник



(підпис)

Омельчук Антон Анатолійович,
к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми



(підпис)

Федотова - Півень Ірина
Миколаївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних
технологій і систем від «26» травня 2023 № 13

Завідувач
кафедри



(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н.,
доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Ірпінь – 2023

Реєстрація № 18

05.06.2023
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

29.05.2023
(дата)


(підпис)

Результат захисту

5 (відмінно)
(оцінка)

14.06.2023
(дата)


(підпис)

Голова комісії:

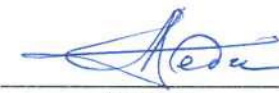

(підпис)

В.М. Франчук
(ініціали, прізвище)

Члени комісії:


(підпис)

М.М. Філоненко
(ініціали, прізвище)

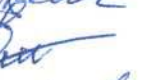

(підпис)

О.В. Редич
(ініціали, прізвище)


(підпис)

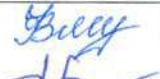
Є.В. Котух
(ініціали, прізвище)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		14.06.2023 Франчук В.М.  Філоненко М.М.  Редич О.В.  Котух С.В.  Суховещька Л.П. 
2. Дотримання вимог щодо оформлення	від 0 до 10	9		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	47		

Оцінювання кваліфікаційної роботи науковим керівником - 47 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
94	A	5 (відмінно)	14.06.2023	Франчук В.М.  Філоненко М.М.  Редич О.В.  Котух С.В.  Суховещька Л.П. 

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій

Кафедра комп'ютерних та інформаційних технологій і систем

Спеціальність: 122 «Комп'ютерні науки»

Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую

Завідувач кафедри


«16» 11

Філоненко М.М.

2022

**Завдання на випускню кваліфікаційну (дипломну) роботу
здобувачу вищої освіти**

Якубовському Василю Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної (дипломної) роботи

Розробка ігрового штучного інтелекту засобами рушія Unity

Затверджена наказом ректора від «15» листопада 2022 № 1205

2. Строк здачі закінченої роботи «26» травня 2023

3. Мета роботи: створення ігрового штучного інтелекту на базі платформи розробки комп'ютерних ігор Unity

Об'єкт дослідження: ігровий штучний інтелект

Предмет дослідження: навчання штучного інтелекту за допомогою методів навчання з підкріпленням з використанням рушія Unity

База дослідження: багатоплатформовий інструмент для розроблення відеоігор і застосунків Unity

4. Вихідні дані до роботи: положення про порядок виконання, оформлення та захисту кваліфікаційних робіт (проектів) в державному податковому університеті

5. Перелік графічного матеріалу: рисунки, формули та графіки, що пояснюють суть застосовуваних методів та засобів

6. Зміст випускної кваліфікаційної (дипломної) роботи (перелік питань за кожним розділом): Вступ, Розділ 1. Аналіз предметної області, 1.1. Розвиток штучного інтелекту, 1.2. Штучний інтелект в комп'ютерних іграх, 1.3. Огляд ігрових рушіїв з підтримкою ІІІ, Розділ 2. Проектування та реалізація штучного інтелекту в ігровому застосунку, 2.1. Методи та алгоритми реалізації штучного інтелекту, 2.2. Реалізація методу штучного інтелекту в Unity, Розділ 3. Пропозиції щодо удосконалення (покращення) одержаних результатів, 3.1. Порівняння результатів навчання штучного інтелекту з іншими роботами, 3.2. Перспективи кваліфікаційної роботи, Висновки, Додатки

7. Календарний план виконання роботи

№	Назва етапів випускної кваліфікаційної (дипломної) роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури ДР	До 15.11.2022	15.11.2022
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику, огляд літератури за темою і вибір конкретних напрямів досліджень)	14.04.2023	14.04.2023
3	II розділ (проведені теоретичні і (або) експериментальні дослідження та проєкті рішення)	15.05.2023	15.05.2023
4	III розділ (пропозиції щодо удосконалення (покращення) одержаних результатів)	22.05.2023	22.05.2023
5	Висновки	22.05.2023	22.05.2023
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	26.05.2023	26.05.2023
7	Отримання відгуку керівника	06.06.2023	06.06.2023
8	Отримання зовнішньої рецензії	08.06.2023	08.06.2023

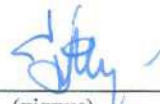
8. Дата видачі завдання «16» листопада 2022

9. Керівник випускної кваліфікаційної роботи


(підпис)

Омельчук А. А.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми


(підпис)

Федотова-Півень І
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Якубовський В.С.
(прізвище, ініціали)

АНОТАЦІЯ

Розробка ігрового штучного інтелекту засобами рушія Unity.

Мета роботи полягає в розробці ігрового штучного інтелекту та впровадження його в ігровий застосунок.

Завданнями роботи є розробка ігрового штучного інтелекту з використанням рушія Unity.

Предметом дослідження є навчання штучного інтелекту за допомогою навчання з підкріпленням з використанням рушія Unity.

Об'єктом дослідження виступає ігровий штучний інтелект.

Базою дослідження виступає багатоплатформовий інструмент для розроблення відеоігор і застосунків Unity.

За результатами дослідження сформульовані відповідні висновки.

Одержані результати можуть бути використані у сфері розробки ігор та ігрових застосунків.

Дипломна робота містить 54 сторінок, 0 таблиць, 20 рисунків, список використаних джерел з 27 найменувань, 5 додатків.

Ключові слова: Штучний інтелект, Ігровий штучний інтелект, Навчання з підкріпленням, Unity, PPO.

Рік виконання дипломної роботи: 2023

Рік захисту роботи: 2023

ABSTRACT

Development of game artificial intelligence using the Unity engine.

The purpose of the work is the development of game artificial intelligence and its implementation in a game application.

The tasks of the work are the development of game artificial intelligence using the Unity engine.

The subject of research is learning artificial intelligence using reinforcement learning using the Unity engine.

The object of research is game artificial intelligence.

The basis of the research is the cross-platform tool for developing video games and applications Unity.

According to the results of the research, appropriate conclusions are formulated.

The obtained results can be used in the field of development of games and game applications.

The thesis contains 54 pages, 0 tables, 20 figures, a list of used sources from 27 titles, 5 appendices.

Keywords: Artificial Intelligence, Gaming Artificial Intelligence, Reinforcement Learning, Unity, PPO.

Year of completion of the qualification work: 2023

Year of work defense: 2023

ЗМІСТ

ВСТУП	6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	8
1.1. Розвиток штучного інтелекту.....	8
1.2. Штучний інтелект в комп'ютерних іграх	12
1.3. Огляд ігрових рушіїв з підтримкою ШІ	18
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ В ІГРОВОМУ ЗАСТОСУНКУ	23
2.1. Методи та алгоритми реалізації штучного інтелекту.....	23
2.2. Реалізація методу штучного інтелекту в Unity.....	32
РОЗДІЛ 3. ПРОПОЗИЦІЇ ЩОДО УДОСКОНАЛЕННЯ (ПОКРАЩЕННЯ) ОДЕРЖАНИХ РЕЗУЛЬТАТІВ	42
3.1. Порівняння результатів навчання штучного інтелекту з іншими роботами.....	42
3.2. Перспективи кваліфікаційної роботи	48
ВИСНОВКИ.....	49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	51
ДОДАТКИ.....	55
Додаток А.....	55
Додаток Б	56
Додаток В	57
Додаток Г	61
Додаток Д.....	68

ВСТУП

Штучний інтелект (ШІ) – це галузь комп'ютерних наук, яка займається створенням комп'ютерних систем та алгоритмів, здатних виконувати завдання, що зазвичай потребують інтелекту людини, таких як розпізнавання образів, мовного розуміння, прийняття рішень, самонавчання та інших. Актуальність штучного інтелекту полягає в тому, що він здатний розв'язувати багато проблем, які досі вважалися складними або неможливими для людей або традиційних комп'ютерних програм [8].

Застосування штучного інтелекту вже сьогодні можна побачити в різних галузях, таких як медицина, фінанси, транспорт, промисловість, наука та інші. Штучний інтелект може бути використаний для покращення ефективності роботи, зниження витрат, підвищення якості послуг та продуктів, а також для створення нових можливостей та інновацій.

Однак, з появою штучного інтелекту також виникають нові проблеми, такі як проблеми конфіденційності даних, етичні проблеми та соціальні наслідки. В цьому контексті актуальність штучного інтелекту зростає, оскільки він стає все більш необхідним для розвитку бізнесу та соціального прогресу, але потребує уважного вивчення та розумного застосування.

Ігровий штучний інтелект – це галузь штучного інтелекту, яка займається створенням інтелектуальних комп'ютерних систем для комп'ютерних ігор. Актуальність ігрового ШІ полягає в тому, що він може підвищити рівень геймплею, зробити гру більш цікавою та захоплюючою для гравців.

Ігровий штучний інтелект може допомогти розширити можливості гри, зробивши її більш складною та різноманітною. Він здатний створювати ігрових персонажів з різними характеристиками та поведінкою, які можуть бути використані в якості опонентів для гравців.

Також, ігровий ШІ може вплинути на зростання кількості електронних ігор та їх розширення на різні платформи, такі як смартфони, планшети, різні ігрові консолі, комп'ютери та інші. Штучний інтелект може допомогти розробникам створювати більш складні та реалістичні ігри зі збалансованою геймплейною механікою та завданнями.

Проте, разом з актуальністю ігрового ШІ з'являється його вплив на поведінку гравців, що може мати як позитивні, так і негативні наслідки. Тому важливо розробляти та використовувати ігровий ШІ відповідально та етично.

У кваліфікаційні роботі розробляється ігровий алгоритм, який матиме змогу аналізувати ситуацію і вирішувати проблеми, які виникають в процесі гри. Завданням є розробити алгоритм, який буде мати високу ефективність і можливості прогнозування майбутніх подій в грі.

Об'єктом дослідження виступає ігровий штучний інтелект.

Предметне дослідження ігрового штучного інтелекту включає аналіз методів та алгоритмів для створення ігрового ШІ, що зможе функціонувати разом з людьми або самостійно. Ігрові штучні інтелекти використовуються для моделювання рішення проблем з високою точністю, а також для створення розумних агентів, які можуть вирішувати складні проблеми за допомогою алгоритмів машинного навчання.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Розвиток штучного інтелекту

Штучний інтелект – це машинний інтелект, який імітує поведінку й мислення людини та може бути запрограмований для вирішення конкретних задач, також це комбінація технік машинного та глибокого навчання [11].

Аналізуючи визначення штучного інтелекту як здатності комп'ютерів до виконання розумових завдань, що властиві людині, можна зробити висновок, що на даний момент ще не було створено повноцінного штучного інтелекту.

Системи штучного інтелекту в основному працюють з великими обсягами позначених навчальних даних, аналізуючи їх для виявлення кореляцій та шаблонів, які використовуються для прогнозування майбутніх станів. Нові методи штучного інтелекту, такі як генеративні моделі, швидко розвиваються і можуть створювати реалістичний контент, такий як текст, зображення, музику та інші медіа.

Машинне навчання, що використовує програми штучного інтелекту, може швидко перетворити величезний обсяг даних, що створюється щодня, на корисну інформацію. Зараз, коли технології штучного інтелекту впроваджуються в більше продуктів та послуг, організації повинні бути налаштовані на потенційну можливість штучного інтелекту створювати системи, які можуть бути упередженими або дискримінаційними, ненавмисно або навмисно.

Розрізняють чотири основні типи штучного інтелекту:

- реактивні машини;
- обмежена пам'ять;
- теорія розуму;
- самосвідомість.

Реактивна машина заснована на головних принципах штучного інтелекту і, як підказує її назва, використовує свій інтелект виключно для сприйняття навколишнього світу і реагування на нього. Оскільки реактивна машина не може зберігати пам'ять, то вона не може використовувати минулий досвід для прийняття рішень в реальному часі.

Реактивні машини призначені для виконання обмеженої кількості спеціалізованих завдань, оскільки вони сприймають світ безпосередньо. Однак це обмеження не є способом зменшення витрат, а натомість забезпечує більшу надійність та стабільність такого типу штучного інтелекту, оскільки реактивна машина буде реагувати на однакові стимули однаково кожен раз.

Хоча реактивний машинний штучний інтелект обмежений за обсягом і нелегко змінюється, він може досягти рівня складності та пропонує надійність, коли створюється для виконання повторюваних завдань.

Штучний інтелект з обмеженою пам'яттю має здатність зберігати попередні дані та прогнози під час збору інформації та зважування потенційних рішень — по суті, заглядаючи в минуле, щоб знайти підказки щодо того, що може статися далі. Штучний інтелект з обмеженою пам'яттю є складнішим і надає більші можливості, ніж реактивні машини [8].

Для створення штучного інтелекту з обмеженою пам'яттю необхідно постійно навчати модель аналізувати та використовувати нові дані, або створити середовище, яке дозволяє автоматично навчати та оновлювати моделі. У машинному навчанні для використання штучного інтелекту з обмеженою пам'яттю потрібно виконати шість кроків: створення навчальних даних, створення моделі машинного навчання, можливість моделі робити прогнози, можливість моделі отримувати зворотний зв'язок від людини або навколишнього середовища, збереження цих даних у вигляді зворотного зв'язку та повторення цих кроків як цикл.

Існують три основні моделі машинного навчання, які використовують штучний інтелект з обмеженою пам'яттю:

1. Навчання з підкріпленням, яке вчиться робити кращі прогнози шляхом повторних спроб і помилок.

2. Довгострокова пам'ять використовує інформацію з минулого для передбачення майбутніх елементів у послідовності. Вона зберігає найсвіжіші дані як найважливіші для прогнозування, тоді як дані з далекого минулого відкидаються, але все ж використовуються для узагальнення та формування висновків.

3. Еволюційні генеративні змагальні мережі є моделями, які постійно розвиваються та змінюються з часом, використовуючи попередні результати для дослідження нових можливостей. Ці моделі використовують моделювання, статистику та випадковість для прогнозування результатів на кожному етапі еволюційного процесу, шукаючи найкращий шлях до досягнення мети.

Теорія розуму є лише теоретичною і досі не досягла технологічних та наукових можливостей, необхідних для створення повноцінного штучного інтелекту.

Ця концепція базується на передумові психології, яка стверджує, що інші живі істоти мають свої думки та емоції, які впливають на поведінку людини. Якщо застосовувати цю концепцію до машин зі штучним інтелектом, то вони можуть зрозуміти, що відчувають люди, тварини та інші машини, і приймати рішення, використовуючи цю інформацію. Для цього машини повинні мати можливість сприймати та оброблювати концепцію «розуму», емоційні реакції при прийнятті рішень та інші психологічні концепції у реальному часі, створюючи двосторонні відносини між людьми та штучним інтелектом.

У майбутньому, коли теорія розуму буде створена в штучному інтелекті, останнім етапом буде досягнення самосвідомості штучного інтелекту. Самосвідомий штучний інтелект матиме свідомість на рівні людини та здатність

розуміти своє існування у світі, а також присутність та емоційний стан інших. Він зможе зрозуміти, що може знадобитися іншим, не тільки на основі того, що вони йому повідомляють, але й на основі того, як вони це роблять.

Самосвідомість в штучному інтелекті базується на пізнанні людьми передумов свідомості та розробленні способів, які дозволять відтворити її в машинах.

Штучний інтелект поділяється на дві категорії: вузький та загальний штучні інтелекти.

Вузький штучний інтелект – це штучний інтелект, який запрограмований на виконання одного завдання. Системи ANI виконують команди в режимі реального часу, під час обробки використовують готову базу даних [11].

Вузький штучний інтелект є найбільш успішною реалізацією штучного інтелекту в наш час, який зустрічається навколо нас у всіх сферах життя. Він спрямований на виконання конкретних завдань, і за останні десятиліття мав численні прориви, які значно поліпшили життя людей.

Кілька прикладів вузького штучного інтелекту включають:

- пошук Google;
- програмне забезпечення для розпізнавання зображень;
- Siri, Alexa та інші персональні помічники;
- автомобілі з автопілотом.

Машинне та глибоке навчання відіграли важливу роль у розвитку багатьох вузьких систем штучного інтелекту.

Машинне навчання використовує статистичні методи для того, щоб навчити комп'ютер поступово покращувати завдання, не потребуючи мільйонів рядків коду. Це можливо завдяки подачі комп'ютерним даним та використанню навчання з вчителем (з використанням позначених наборів даних) або навчання без вчителя (з використанням немаркованих наборів даних).

Глибоке навчання – це тип машинного навчання, який виконує вхідні дані через біологічно натхненну архітектуру нейронної мережі. Нейронні мережі містять ряд прихованих шарів, через які обробляються дані, що дозволяє машині «глибше» заглиблюватися в своє навчання, створюючи зв'язки та зважуючи вхідні дані для найкращих результатів [8].

Загальний штучний інтелект відноситься до програмних систем, які імітують логіку людини. Отже, даний алгоритм штучного інтелекту закодований на виконання складних інтелектуальних завдань для демонстрації людської присутності [11].

Пошуки загального штучного інтелекту є складним завданням, але створення машини з інтелектом на рівні людини, яка зможе застосовуватись у будь-якій сфері, залишається Святим Граалем для багатьох дослідників штучного інтелекту.

Пошук універсального алгоритму навчання та дії, який був би застосовний в будь-якому середовищі не є новим, проте, з плином часу труднощі зі створенням машин, які мають повний набір когнітивних здібностей, не зменшуються.

1.2. Штучний інтелект в комп'ютерних іграх

Розробники ігор мають на меті створити для гравців захоплюючий інтерактивний досвід. Цей досвід формується завдяки різноманітним ігровим елементам, таким як графіка, звуки, тривалість гри, сюжет, виклики та взаємодія з іншими об'єктами (гравці, вороги тощо). Тому, застосування штучного інтелекту стає незамінним для розробників, що дозволяє керувати зростаючою складністю геймплею.

Майже всі відеоігри включають елементи штучного інтелекту в певній формі. Однак, використання сучасних методів штучного інтелекту може бути корисним для ігрових компаній у різних аспектах бізнесу. Наприклад,

застосування штучного інтелекту для покращення реалістичності візуальних ефектів, створення контенту, балансування складності гри та надання неігровим персонажам інтелекту може покращити взаємодію з користувачем, зменшуючи витрати і час, витрачений на розробку гри.

Штучний інтелект в іграх дає гравцеві більшу адаптивність та чутливість при проходженні відеоігор, що забезпечуються завдяки тому, що неігрові персонажі поведуться творчо, ніби ними керує гравець-людина.

Слід відзначити, що існує різниця між справжнім штучним інтелектом та ігровим штучним інтелектом. Ігровий штучний інтелект має обмежений функціонал, оскільки він не потребує реалізації всіх можливостей штучного інтелекту, за відсутністю потреби у цьому в ігровому середовищі. Ігровий штучний інтелект відрізняється від справжнього штучного інтелекту тим, що він проектується та будується для виконання обмеженого числа задач у межах ігрового середовища. Ігровий штучний інтелект не є повноцінним штучним інтелектом, оскільки не має деяких функцій, таких як почуття та самосвідомість. Такі функції не є необхідними для реалізації ігрового штучного інтелекту, який створюється для виконання конкретних завдань у межах ігрового середовища. Суть створення штучного інтелекту полягає в тому, щоб надавати людині всебічну допомогу при виконанні різноманітних завдань, але ігровий штучний інтелект зосереджується на виконанні завдань у межах ігрового середовища, які можуть бути виконані людиною.

Однією з основних функцій ігрового штучного інтелекту є прийняття рішень. Кожен агент, який підпорядковується штучному інтелекту, має свій власний набір правил та інструкцій для прийняття рішень. В залежності від складності завдань, які необхідно виконати, умови, за якими приймаються рішення, можуть мати складну структуру.

Для розробки агентів у галузі ігрового штучного інтелекту використовують дві загальні методики: машини станів та ієрархічні дерева вибору рішень. Перший

метод використовується для складних систем, тоді як другий – для більш простих. У практиці часто використовують комбінацію обох методів

Через це підходи до створення ігрового штучного інтелекту відрізняються, а саме, притаманні різні види спрощень, які в свою чергу дадуть бажаний результат – наявність ілюзії штучного інтелекту у неігрових персонажів. Наприклад, у іграх жанру шутеру боти мають інформацію про точне положення гравця на мапі, тому вони мають змогу слідкувати крізь стіни за ним, що в свою чергу приводить до миттєвої смерті гравця, оскільки в нього не було часу на реакцію. Щоб вирішити дану проблему розробники штучно знижують влучність ботів [16].

Ігрових персонажів із штучним інтелектом поділяють на декілька категорій:

- Неігрові персонажі (NPC) – це категорія персонажів, які є дружніми, нейтральними або ворожими до гравця. Зазвичай, ці персонажі включають торговців, напарників та інших.

- Боти – це категорія персонажів, що ворожо налаштовані проти гравця. Вони створені з метою наближення до персонажа гравця за своїми можливостями. Зазвичай, їх використовують у невеликих кількостях або тільки в безпосередніх боях для ознайомлення гравця з новим типом ворога. Пізніше їх кількість може зрости.

- Моби – це категорія ворожих персонажів зі зниженим рівнем штучного інтелекту, які з'являються часто у великих кількостях.

Використання штучного інтелекту в іграх має свої обмеження. Розробка реалістичних NPC, які б динамічно наповнювали всесвіт гри та створювали реалістичне середовище для гравців, виявляється надзвичайно складною. NPC, що базуються на штучному інтелекті, зазвичай розробляються для ефективної реакції на певні дії гравців. Вони можуть виконувати свої спеціалізовані функції досить ефективно, але при цьому залишаються передбачуваними, що поступово зводить їх вплив на гравців та утримання їх уваги.

Більшість сучасних ігор використовують попередньо запрограмованих NPC як супротивників, проте розвиток штучного інтелекту дозволяє додавати цим персонажам інтелект, що робить їх менш передбачуваними та більш привабливими для гравців. Більш складний штучний інтелект дозволяє NPC ставати розумнішими та реагувати на умови гри в нових й унікальних способах під час гри. Ігрові компанії все частіше працюють над NPC на основі штучного інтелекту. Наприклад, SEED (EA) тренує персонажів NPC, імітуючи кращих гравців в іграх, що дозволяє значно скоротити час розробки NPC, оскільки жорстке кодування їхньої поведінки є виснажливим і тривалим процесом.

Завдяки здатності штучного інтелекту передбачувати можливі майбутні результати у грі, він може стати практично непереможним. В одному з відомих прикладів – грі в хрестики-нулики - штучний інтелект реалізує Мінімаксий принцип, що призводить до нічийх у будь-якій грі, незалежно від ходів гравців. Штучний інтелект також показав успішні результати у грі в шахи, Го та StarCraft 2.

Історія використання штучного інтелекту в відеоіграх сягає з 1950-х років, коли була розроблена комп'ютерна шахова гра з алгоритмом MiniMax для аналізу положення фігур на дошці і вибору найкращого ходу. У 70-х роках відеоігри почали революціонізувати ринок розваг, а штучний інтелект відіграв ключову роль у концепції провідних брендів. Зародження штучного інтелекту в відеоіграх відбулося задовго до того, як галузь стала необхідною для майже кожної людини.

У 1972 році штучний інтелект мав можливість створити траєкторію руху м'яча після удару гравця для його повернення. У грі Space Invaders наростаюча швидкість руху інопланетян була залежна від апаратних особливостей мікрокомп'ютерів, а не від програмного коду, проте в 70-х роках XX ст. мікропроцесори не змогли одночасно обробляти велику кількість спрайтів ворогів, що призводило до повільної роботи користувацького інтерфейсу.

У 1980-х роках Pac-Man, популярна гра, отримала систему пошуку маршрутів, яка дозволяла ворогам легко визначати шлях, яким рухається гравець,

та попереджати його. Donkey Kong також використовував штучний інтелект зі складнішим рівнем гри, який вимагав від гравців максимальних зусиль. Однак, в той час штучний інтелект був нестійким, тому що він не міг навчитися на своїх помилках та пристосовуватися до поведінки гравця, що вимагало від гравців пізнати деякі закономірності, щоб прогресувати в грі.

У кінці 80-х розробникам ігор довелося приділити серйозну увагу розвитку ігрового штучного інтелекту, оскільки він став невід'ємною частиною якісного ігрового продукту. Також у цей період з'явилося багато нових жанрів ігор, серед яких стратегії в реальному часі (RTS) найбільше стимулювали розвиток ігрового штучного інтелекту. Ці стратегії почали розвивати штучний інтелект в іграх як самостійну та важливу галузь ігрової індустрії.

У 1990-х роках розробники віддавали багато уваги розвитку інтелекту неігрових персонажів в іграх, але багато NPC були запрограмовані без можливості вчитися. Такі моделі поведінки були попередньо задані і не могли змінюватися під впливом гравця. Проте це не лише технічна складність, але й забезпечення стійкості геймплею, оскільки непередбачувана поведінка NPC може зіпсувати ігровий досвід.

Гра Petz стала дуже популярною на свій час завдяки тому, що її NPC могли навчатися новим речам. Ця відеогра дозволяла гравцям навчити оцифровану домашню тваринку різним трюкам, аналогічним тим, які вчать справжніх собак чи кішок.

Розробники продовжували розвивати напрямок у сфері штучного інтелекту, і до 2011 року алгоритми створення місій у відеоіграх генерували прості, але унікальні завдання. У грі ARPG, The Elder Scrolls V: Skyrim, ці алгоритми випадковим чином налаштовували місця, ворогів і цілі, тим самим знімаючи велике навантаження з дизайнерів ігрового світу.

За останні кілька років штучний інтелект став відігравати все більшу роль не лише в самих іграх, але й у процесах розробки ігор. Експерименти з методами

глибокого навчання дозволили штучному інтелекту запам'ятовувати послідовність зображень або тексту та використовувати отримані знання для моделювання всесвіту гри. Зараз штучний інтелект використовується для створення творів мистецтва, схожих на Пікассо, або електронних листів, які нібито написані людьми. Ця ж технологія зараз адаптується для розробки ігор, за допомогою експериментів зі штучним інтелектом, здатних писати ігрові рівні (створення будівель або пейзажів), використовуючи лише фотографії [13].

Можна припустити, що штучний інтелект через деякий проміжок часу зможе створювати цілі ігри з нуля, майже не потребуючи допомоги розробників-людей.

Штучному інтелекту часто дозволяється шахраювати, намагаючись перемогти людей. Однак потреба в обмані розкриває межі досяжного штучного інтелекту. Проте в іграх, де потрібна стратегія та креативність, люди зазвичай здатні перемагати штучний інтелект. Це пояснюється тим, що граючи такі ігри, люди можуть навчитися на власних помилках, тоді як штучний інтелект ще не може робити цього. Це обмежує можливості використання штучного інтелекту в таких іграх. Штучний інтелект має обмеження, які стосуються не тільки ігор, але і відсутності контексту поза навчальними даними. Штучний інтелект, що використовує машинне навчання, потребує великої кількості навчальних даних для досягнення успішних результатів. На даний момент, недостатня кількість навчальних даних є загальною проблемою.

Можливість використання штучного інтелекту в іграх обмежена зазначеними факторами, але це стосується лише деяких видів ігрових додатків. Розвиток технологій штучного інтелекту знаходиться в постійному процесі, і в найближчому майбутньому ці проблеми можуть бути успішно вирішені.

Фінальною метою розробки ігрового штучного інтелекту є досягнення рівня, при якому гравцеві буде складно відрізнити справжню людину від бота. Але в

ігровому штучному інтелекті завжди будуть присутні штучні спрощення його поведінки задля забезпечення комфорту гравця.

1.3. Огляд ігрових рушіїв з підтримкою ШІ

Unreal Engine – це ігровий рушій, який розробляється і підтримується компанією Epic Games. Це середовище для розробки комп'ютерних ігор, а також для створення візуалізацій в архітектурній, медичній та інших галузях. Unreal Engine заснований на мові програмування C++, має потужний графічний редактор, підтримує розробку ігор для різних платформ, включаючи ПК, консолі та мобільні пристрої [13].

Серед переваг Unreal Engine, можна виділити:

- Висока якість графіки: Unreal Engine дозволяє створювати ігри з дуже реалістичною графікою, що дозволяє ігровим розробникам створювати неймовірно красиві та деталізовані світи;
- Швидкість розробки: Unreal Engine має потужні засоби для швидкої розробки ігор, такі як візуальний редактор, пакети з готовими рішеннями, мережеві засоби та інструменти для оптимізації продуктивності;
- Кросплатформність: Unreal Engine підтримує різні платформи, такі як Windows, Mac, Linux, Android, iOS, PlayStation та Xbox, що дозволяє розробникам створювати ігри для різних платформ з використанням одного коду;
- Відкритий код: Unreal Engine має відкритий код, що дозволяє розробникам змінювати та адаптувати його до своїх потреб;
- Підтримка віртуальної реальності: Unreal Engine має потужні засоби для створення віртуальної реальності, що дозволяє розробникам створювати ігри та додатки для віртуальної реальності з високою якістю графіки та реалістичними ефектами.
- Наявність офіційної та користувацької підтримки;

- Наявність торгового майданчика асетів (представлення будь-якого об'єкту, який використовується в ігровому рушії).

Unreal Engine є одним з провідних ігрових рушіїв, які підтримують розробку штучного інтелекту в іграх. Unreal Engine надає розробникам широкий спектр інструментів для роботи зі штучним інтелектом, таких як Behavior Trees, розпізнавання облич, системи прогнозування поведінки й так далі. Крім того, Unreal Engine підтримує інтеграцію з різними фреймворками штучного інтелекту, такими як TensorFlow і PyTorch. Більшість великих ігрових проєктів, які використовують штучний інтелект, базуються на Unreal Engine [13].

GameMaker Studio - це ігровий рушій, розроблений Марком Овермарсом та є потужним інструментом для розробки ігрових проєктів. Він спрямований на розробку двовимірних ігор. Рушій підтримує різні бібліотеки. Також, присутні інструменти для роботи зі скалярними та векторними величинами, математичними виразами. У GameMaker Studio вбудований фізичний рушій Box2D, що дозволяє проводити маніпуляції з фізикою та фізичними об'єктами [7].

GameMaker Studio не має вбудованих засобів для розробки штучного інтелекту, але його можна інтегрувати з бібліотеками та іншими інструментами для створення штучного інтелекту. Наприклад, GameMaker можна використовувати з OpenAI Gym, яке надає набір середовищ для навчання з підкріпленням, таких як Atari та MuJoCo. Крім того, можна використовувати різноманітні бібліотеки та фреймворки для розробки штучного інтелекту, такі як TensorFlow, PyTorch, Keras та інші, та інтегрувати їх у свої проєкти, розроблені з використанням GameMaker Studio.

Unity – це програмний інструментарій, який надає інтегроване середовище розробки та ігровий рушій для створення 2D та 3D ігор для різних платформ, включаючи персональні комп'ютери, консолі, мобільні пристрої та віртуальну реальність [8].

Unity надає розробникам можливість створювати графічний інтерфейс, моделювати 3D-об'єкти, програмувати поведінку об'єктів та імпортувати різні типи ресурсів, такі як зображення, звуки та анімації. Цей інструментарій підтримує різні мови програмування, зокрема C# та Unity Script. Unity використовують як початківці, так і професіонали, оскільки він має простий інтерфейс та готові компоненти, що дозволяє зосередитися на творчості та розробці гри, не витрачаючи багато часу на реалізацію базових функцій.

На даний момент Unity є одним з найбільш популярних ігрових рушіїв, який постійно оновлюється. Гри та додатки, створені з використанням Unity, можуть працювати на різних платформах, включаючи Windows, Android, iOS, Linux.

Unity має низку переваг:

- Unity дозволяє гнучко підтримувати та модифікувати ігрові додатки завдяки своїй готовності до роботи з різними типами проєктів. Крім того, наявність потужної системи скриптів, яка працює з мовою програмування C#, є ще однією перевагою Unity.
- Unity дозволяє створювати продукти з широким спектром застосувань, які можуть бути підтримані на багатьох різних платформах та системах.
- Зручність – однією з головних переваг Unity порівняно з іншими аналогами є його інтуїтивно зрозумілий інтерфейс. Це значно сприяє швидкому створенню ігрових прототипів і уникненню низькорівневого програмування. Доступна можливість використання візуального редактора логіки забезпечує додаткову зручність при розробці.
- Unity забезпечує широкий спектр можливостей для взаємодії, включаючи багато компонентів для вирішення різних задач, які розділені на відповідні категорії. Крім того, Unity має документоване API з доступом до всіх систем, що дозволяє використовувати різні інструменти та функції для розробки.

- Каталог ресурсів, доступних в Unity, включає в себе Asset Store - магазин активів, бібліотеку готових ресурсів та інструментів, що допомагає підвищити продуктивність та якість ігрових додатків.

Структура проєктів, що створені з використанням рушія Unity, базується на взаємодії багатьох ігрових об'єктів, компонентів, скриптів та префабів на ігровій сцені.

У межах гри, сцена є обмеженим ігровим простором, на якому розміщуються та взаємодіють об'єкти згідно зі специфікацією, визначеною розробниками під час створення гри.

Ігрові об'єкти, також відомі як "Game Object" в термінології Unity, є базовими функціональними одиницями в ігровому середовищі, а їх взаємодія на сцені визначає сам процес гри. При створенні ігрового об'єкту він автоматично отримує базовий компонент - Transform, що відповідає за опис його позиції, повороту та масштабу в просторі ігрового середовища. За допомогою додавання різних компонентів до об'єкта можна значно розширити його функціональні можливості та забезпечити реалізацію будь-якого бажаного сценарію гри [8].

Компоненти - це модулі, що містять конкретні дані про властивості як ігрових об'єктів, так і подій, що відбуваються в грі. Кожен компонент має своє визначене призначення, і може впливати на поведінку, зовнішній вигляд та інші функції ігрових об'єктів. Unity надає безліч компонентів з різними призначеннями, які розробники використовують для вирішення потреб проєкту [8].

Unity використовує концепцію Ігрових Об'єктів (Game Object), але для повторного використання цих ігрових одиниць зазвичай зберігають їх у вигляді шаблону, який називається префаб. Префаби дуже корисні при створенні складних ігрових об'єктів з різними компонентами та налаштуваннями, оскільки вони значно полегшують процес формування нових ігрових одиниць. Кожен префаб може містити власні налаштування, які відрізняються від базового об'єкту.

Механізм використання префабів дозволяє багаторазово використовувати ігрову одиницю на сцені (або сценах) без потреби створення нового об'єкту з нуля [8].

Структура та взаємозв'язок компонентів в ігровому движку Unity визначають подальшу структуру та можливості гри. Ігровий рушій надає широкий набір інструментів для побудови складних ігрових об'єктів. Unity працює на двох потоках: головному, де змінюється кадр та життєвий цикл усіх скриптів та програм загалом, та фізичному, що відповідає за обробку об'єктів з урахуванням фізики. Цей ігровий рушій містить усі необхідні інструменти, такі як TensorFlow або Caffè, для розробки ігор з використанням штучного інтелекту і вбудовані інструменти для створення різноманітних ігрових агентів й налаштування їх поведінки, та, тому, вибраний як середовище для розробки даного проекту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ШТУЧНОГО ІНТЕЛЕКТУ В ІГРОВОМУ ЗАСТОСУНКУ

2.1. Методи та алгоритми реалізації штучного інтелекту

Після огляду наявної інформації про алгоритми машинного навчання в галузі ігрової розробки, можна виділити кілька методів, які є найбільш популярними. Серед них є навчання з підкріпленням, імітаційне навчання та зворотне навчання з підкріпленням. У даному контексті найбільш ефективним методом є навчання з підкріпленням.

Навчання з підкріпленням - один із способів взаємодії, в ході якого випробувана система (агент) навчається, взаємодіючи із середовищем. Якщо спростити, то робота будується за принципом "дають - бери, б'ють - біжи", отримуючи після будь-яких взаємодій з середовищем або шкоду, або нагороду [8].

Цей алгоритм має на меті зменшення кількості спроб для дослідження різних елементів середовища, використовуючи довгострокове планування (наприклад, "якщо зараз зробити цей вибір, то можливо отримати більшу нагороду в майбутньому"). Для ефективної роботи цього алгоритму потрібні середовища зі складними, але формальними правилами, які дозволяють обмежити можливості агента [8].

Зазвичай, система навчання з підкріпленням складається з двох компонентів - агента та середовища. Коли середовище надсилає агенту поточний стан, агент застосовує свої знання та вибирає дію для виконання. Після цього середовище повертає агенту наступний стан та нагороду за попередню дію. Агент використовує ці дані, щоб підлаштувати свої знання та оновити стратегію вибору дій. Цей процес повторюється до тих пір, поки система не досягне кінцевого стану, що закінчує виконання алгоритму.

Навчання з підкріпленням передбачає наявність агента, який взаємодіє з оточуючим середовищем, виконуючи певні дії. Отримуючи нагороду за свої дії від середовища, агент продовжує їх виконувати. Алгоритми з частковим навчанням мають на меті знайти стратегію, яка дозволяє агенту вибрати одну з дій у станах навколишнього середовища. Зазвичай середовище описується як Марківський процес прийняття рішень з кінцевим набором станів, і тому алгоритми навчання з підкріпленням пов'язані з динамічним програмуванням. Імовірності виграшів та переходу станів у Марківському процесі прийняття рішень зазвичай є випадковими величинами, але стаціонарними для даної задачі. У відмінну від навчання з учителем, при навчанні з підкріпленням не надаються точні пари "вхідні дані-відповідь", і субоптимальні рішення (які дають локальний екстремум) не обмежуються явно. Мета навчання з підкріпленням полягає в пошуку балансу між дослідженням невивчених областей та застосуванням наявних знань (exploration vs exploitation). Питання про баланс між вивченням та застосуванням досліджується у задачі про «багаторукогого бандита» [8].

У навчанні з підкріпленням, найпростіша формальна модель включає наступні складові:

- Множина станів оточення (states) S : Визначає всі можливі стани, в яких може перебувати агент у взаємодії з середовищем.
- Множина дій (actions) A : Представляє всі можливі дії, які агент може виконати у кожному стані.
- Функція виграшів (rewards): Це безліч реальних чисел, які відображають нагороди, отримані агентом за виконані дії в різних станах.

У будь-який момент часу t , агент знаходиться у певному стані $s_t \in S$ і може вибрати одну з можливих дій $a \in A(s_t)$. Після вибору дії, агент переходить до нового стану s_{t+1} і отримує виграш r_t . Взаємодія з середовищем відбувається шляхом послідовної зміни станів та отримання виграшів [8].

Задача агента, який навчається з підкріпленням, полягає у тому, щоб розробити стратегію $\pi: S \rightarrow A$, яка вибирає оптимальну дію в кожному стані, з метою максимізації суми виграшів $R = r_0 + r_1 + \dots + r_n$. Це справедливо як для Марківського процесу прийняття рішень з термінальним станом, так і для невизначеного часу гри:

$$R = \sum_t \gamma^t r_t \quad (2.1)$$

Для задач Марківського процесу прийняття рішень (МППР), в яких немає термінальних станів використовується множник дисконтування γ (де $0 \leq \gamma \leq 1$) для майбутніх виграшів [8].

Навчання з підкріпленням є ефективним методом, особливо для вирішення задач, пов'язаних з вибором між короткостроковими та довгостроковими вигодами.

Множина S визначається як безліч станів оточення.

Гра агента із середовищем:

- ініціалізація стратегії $\pi_1(a|s)$ та стану середовища s_1 ;
- для всіх $t = 1 \dots T$:
- агент вибирає дію $a_t \sim \pi_t(a|s_t)$;
- середовище генерує нагороду $r_t + 1 \sim p(r|a_t, s_t)$ та новий стан $s_{t+1} \sim p(s|a_t, s_t)$;
- агент коригує стратегію $\pi_{t+1}(a|s)$.

Це марківський процес прийняття рішень (МППР), якщо $P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \dots, s_1, a_1) = P(s_{t+1} = s', r_{t+1} = r | s_t, a_t)$, МППР називається фінітним, якщо $|A| < \infty, |S| < \infty$

Після визначення функції виграшу, наступним кроком є визначення алгоритму, який забезпечить знаходження оптимальної стратегії. Наївний підхід до вирішення цього завдання передбачає такі кроки:

- випробувати усі можливі стратегії;
- вибрати стратегію з найбільшим очікуваним виграшем.

Перша складність цього підходу полягає у тому, що кількість доступних стратегій може бути дуже великою або навіть нескінченною. Друга проблема виникає, якщо виграші є стохастичними - щоб точно оцінити виграш від кожної стратегії, потрібно багаторазово застосувати кожну з них. Цим проблемам можна запобігти, якщо застосувати певну структурування та, можливо, дозволити результатам, отриманим від однієї стратегії, використовувати для оцінки інших. Двома основними підходами для реалізації цих ідей є оцінка функції корисності та пряма оптимізація стратегій [8].

Підхід з використанням функції корисності ґрунтується на безлічі оцінок очікуваного виграшу лише для однієї стратегії, яка може бути поточною або оптимальною. Цим методом намагаються оцінити очікуваний виграш зі стану s при дотриманні стратегії π

$$V(s) = E[R|s, \pi] \quad (2.2)$$

або очікуваний виграш при виборі рішення a в стані s та дотриманні π .

$$Q(s, a) = E[R|s, \pi, a] \quad (2.3)$$

Якщо ми використовуємо функцію корисності Q для вибору оптимальної стратегії, ми завжди можемо вибрати оптимальні дії, які максимізують значення цієї функції. У випадку використання функції корисності V , потрібно мати модель оточення у вигляді ймовірностей $P(s'|s, a)$, що дозволяє побудувати функцію корисності.

$$Q(s, a) = \sum s' V(s') P(s'|s, a) \quad (2.4)$$

Інший підхід полягає у використанні методу "виконавець-критик", де модель розділяється на дві частини: критик, який оцінює корисність стану V , і виконавець, який вибирає дії в кожному стані. Якщо ми маємо фіксовану стратегію π , то оцінити очікуваний виграш $E[R]$ при $\gamma = 1$ можна просто шляхом усереднення безпосередніх виграшів. Найбільш очевидний спосіб оцінки при $\gamma \in (0, 1)$ - це усереднити сумарний виграш після кожного стану. Однак для цього необхідно, щоб МППР досяг термінального стану і завершився.

Отже, будування шуканої оцінки при $\gamma \in (0,1)$ не є очевидним. Однак, можна помітити, що R утворюють рекурсивне рівняння Беллмана:

$$E[R|s_t] = r_t + \gamma E[R|s_{t+1}] \quad (2.4)$$

Шляхом використання методу градієнтного спуску з квадратичною функцією помилок та підстановки наявних оцінок V , можна розробити алгоритм навчання з тимчасовими впливами. У випадку, коли i стан, i дії є дискретними, можна використовувати табличні оцінки для кожного стану.

Multi-Armed Bandit - це модель в машинному навчанні та теорії прийняття рішень, що використовується для вивчення торгових стратегій, при яких необхідно вибрати найбільш прибутковий варіант зі знанням того, що кожен вибір може мати різну ймовірність успіху та винагороду [8].

Назва Multi-Armed Bandit відображає аналогію з грою в казино, де гравець зіткнувся з великою кількістю однакових ігрових автоматів (бандитів), кожен з яких має різні виплати та ймовірності виграшу, і гравцю потрібно вирішити, на які автомати варто грати, щоб отримати найбільший виграш [8].

Формулювання.

A - безліч можливих дій (ручок автомата),

$p_a(r)$ – невідомий розподіл нагороди $r \in R \forall a \in A$,

$\pi_t(a)$ – стратегія агента в момент $t \forall a \in A$.

Гра агента із середовищем:

- ініціалізація стратегії $\pi_1(a)$;
- для всіх $t = 1 \dots T$:
- агент вибирає дію (ручку) $a_t \sim \pi_t(a)$;
- Середовище генерує нагороду $r_t = p_{a_t}(r)$;
- агент коригує стратегію $\pi_{t+1}(a)$.

$Q_t(a) = \sum_{i=1}^t r_i [a_i = a] \sum_{i=1}^t 1 [a_i = a] \rightarrow \max$ - середня нагорода у t іграх, $Q^*(a) = \lim_{t \rightarrow \infty} Q_t(a) \rightarrow \max$ - цінність дії a .

Маємо автомат - N -рукого бандита, і на кожному кроці ми вибираємо, за яку з N ручок автомата тягнути, набір дій $A = 1, 2, \dots, N$.

Вибір дії a_t на кроці t вимагає винагороди $R(a_t)$, а $R(a) \forall a \in A$ є випадковою величиною з невідомим розподілом. Стан нашого середовища не змінюється крок за кроком, а це означає, що набір станів S тривіальний і ні на що не впливає, тому його можна ігнорувати. Для простоти слід вважати, що кожна дія відповідає певному розподілу в часі. Якщо ці розподіли відомі, то очевидною стратегією є обчислення математичного очікування кожного розподілу, вибір дії з найбільшим математичним сподіванням і тепер робити це на кожному кроці [8].

Проблема полягає в тому, що розподіл невідомий, але можна оцінити математичне сподівання деякої випадкової величини з невідомим розподілом. Для K експериментів ξ_k оцінкою математичного сподівання є середнє арифметичних результатів експериментів:

$$E(\xi) = \frac{1}{K} \sum_{k=1}^K \xi_k \quad (2.5)$$

Це завдання використовується як модель для розуміння того, як може виникати конфлікт між експлуатацією (exploitation) та дослідженням (exploration).

ϵ -жадібність — це стратегія в машинному навчанні та теорії прийняття рішень, яка використовується в «багаторуких бандитах» та інших проблемах вибору між кількома варіантами з різними винагородами [8].

Стратегія полягає в тому, щоб вибрати випадкове число (ϵ) в діапазоні від 0 до 1, і якщо це число менше за деяке значення ϵ , вибрати випадковим чином між можливими варіантами, а якщо число більше за ϵ , вибрати варіант із найвищою винагородою [8].

Таким чином, ϵ -жадібні стратегії балансують між дослідженням менш вигідних варіантів (шляхом випадкового вибору) та експлуатацією вигідних варіантів (вибір варіанту з найвищою винагородою).

Нагорода для стратегії з різними ϵ

Слід ввести параметр $\epsilon \in (0, 1)$.

На кожному кроці t

- Отримаємо значення - випадкової величини рівномірно розподіленої на відріжку $(0,1)$;
- Якщо $\alpha \in (0, \varepsilon)$, то виберемо дію $a_t \in A$ випадково і рівномірно, інакше як у жадібній стратегії виберемо дію з максимальною оцінкою математичного очікування;
- Оновлюємо оцінки так само, як у жадібній стратегії.

Якщо $\varepsilon = 0$, це звичайна жадібна стратегія. Але якщо $\varepsilon > 0$, то на відміну від жадібної стратегії, на кожному кроці відбувається «дослідження» випадкових дій з імовірністю ε .

Стратегія Softmax — це стратегія машинного навчання та теорії прийняття рішень, яка використовується в «багаторуких бандитах» та інших проблемах із множинним вибором із відомими виграшами [8].

Основна ідея алгоритму Softmax полягає в зменшенні втрат у дослідженнях шляхом більш рідкого вибору дій, які раніше приносили менше винагороди. Щоб досягти цього, ваговий коефіцієнт обчислюється для кожної дії з урахуванням вибору дії. Чим більше $Q_t(a)$, тим більша ймовірність вибору a :

$$\pi_t + 1(a) = \exp(Q_t(a)/\tau) / \sum_{b \in A} \exp(Q_t(b)/\tau) \quad (2.6)$$

$\tau \in (0, \infty)$ – параметр, за допомогою якого можна налаштувати поведінку алгоритму [8].

При $\tau \rightarrow \infty$ стратегія прагне до рівномірної, тобто Softmax менше залежатиме від значення виграшу і вибирати дії більш рівномірно (exploration).

При $\tau \rightarrow 0$ стратегія прагне до жадібної, тобто алгоритм буде більше орієнтуватися на відомий середній виграш дій (exploitation).

Експонента використовується для того, щоб ця вага не дорівнювала нулю навіть у діях, де винагорода дорівнює нулю. Евристичний: має сенс, щоб параметр τ зменшувався з часом.

Попередні алгоритми використовували дані про середні виграші після прийняття рішення. Проблема полягає в тому, що якщо дія дає винагороду з певною ймовірністю, то спостереження будуть неточними, і ми можемо помилково визначити найприбутковішу дію.

Алгоритм верхнього довірчого інтервалу (UCB) — це сімейство алгоритмів, які намагаються вирішити цю проблему, використовуючи дані не лише про середній виграш, але й про достовірність значення виграшу [8].

Також як softmax в UCB при виборі дії використовується ваговий коефіцієнт, який є верхньою межею довірчого інтервалу значення виграшу:

$$\pi_t + 1(a) = Q_t(a) + ba \quad (2.7)$$

$ba = 2\ln \sum aPaPa - - - - - \sqrt{\quad}$ - це бонусне значення, яке відображає ступінь невідомості або неперевіреності певної дії порівняно з іншими [8].

Хоча UCB не використовує випадковість для вибору дій і не має параметрів, які можна налаштовувати, результати його роботи можуть бути менш точними у порівнянні з іншими алгоритмами. Це пояснюється тим, що UCB частіше вибирає недосліджені дії.

Агент формує функцію корисності Q на основі винагороди, одержаної від середовища, що дозволяє йому вибирати стратегію поведінки більш обґрунтовано, а не випадково. Однією з переваг Q -навчання є можливість порівняти очікувану корисність доступних дій без необхідності формувати модель середовища. Цей підхід зазвичай використовується для ситуацій, які можуть бути представлені у вигляді МППР [8].

Фундаментальною концепцією Q -навчання є те, що агент навчається оцінювати функцію корисності Q , яка вказує на очікувану винагороду, яку агент може отримати, виконавши дію a у стані s та в подальшому дотримуючись оптимальної стратегії вибору дій.

Алгоритм Q-навчання є дуже ефективним методом навчання з підкріпленням, який може знайти застосування у різних сферах, таких як ігри, керування роботами, автономна навігація та багато інших.

Таким чином, алгоритм це функція якості від стану та дії:

$$Q: S \times A \rightarrow R \quad (2.8)$$

Функція Q ініціалізується випадковими значеннями перед тим, як почати навчання. В процесі навчання на кожному кроці часу t агент вибирає дію a_t , отримує нагороду r_t , переходить у новий стан $s_t + 1$, який може залежати від попереднього стану s_t та дії a_t , і оновлює функцію Q . Оновлення функції використовує виважену середню між попереднім значенням та новим значенням, де ваговим коефіцієнтом виступає навчальна швидкість агента:

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) * Q(s_t, a_t) + \alpha * (r_t + \gamma * \max_a Q(s_{t+1}, a)) \quad (2.8)$$

де r_t це нагорода, отримана при переході зі стану s_t у стан $s_t + 1$, і це швидкість навчання ($0 < \alpha \leq 1$). Алгоритм закінчується, коли агент перетворюється на термінальний стан $s_t + 1$ [8].

Проксимальна оптимізація політики - це алгоритм навчання з підкріпленням для навчання стратегій поведінки агента в середовищі. Він був розроблений для того, щоб вирішити проблему збільшення функції втрат у класичних методах, які використовують градієнтний спуск для оптимізації стратегій. Проксимальна оптимізація політики є поєднанням двох підходів: апроксимації функції переваг і відхилю стратегії від свого початкового значення з обмеженням [24].

За допомогою навчання з підкріпленням можливо легко реалізувати функцію вартості, запустити градієнтний спуск на ній і бути дуже впевненими, що отримаємо чудові результати з відносно невеликим налаштуванням гіперпараметрів. Шлях до успіху в навчанні з підкріпленням не такий очевидний — алгоритми мають багато рухомих частин, які важко налагодити, і вони вимагають значних зусиль для налаштування, щоб отримати хороші результати. Проксимальна оптимізація політики встановлює баланс між простотою

впровадження, складністю вибірки та простотою налаштування, намагаючись обчислити оновлення на кожному кроці, яке мінімізує функцію витрат, забезпечуючи при цьому відносно невелике відхилення від попередньої політики [8].

Проксимальна оптимізація політики, використовує нову цільову функцію, яка зазвичай не зустрічається в інших алгоритмах:

$$L^{CLIP}(\theta) = \widehat{E}_t[\min(r_t(\theta)) \widehat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \widehat{A}_t)] \quad (2.9)$$

де θ – це параметр політики

$\widehat{E}_t$ – емпіричне очікування за кроками в часі

r_t – це відношення ймовірності відповідно до нової та старої політики

$\widehat{A}_t$ – це передбачувана перевага в часі

ϵ – це гіперпараметр, який зазвичай 0,1 або 0,2

Він реалізує спосіб оновлення довірчої області, який сумісний зі стохастичним градієнтним спуском, і спрощує алгоритм, видаляючи покарання Кульбака–Лейблера і потребу в адаптивних оновленнях. У тестах цей алгоритм продемонстрував найкращу продуктивність у завданнях безперервного контролю та майже відповідає продуктивності ACER на Atari, незважаючи на те, що він набагато простіший у застосуванні.

2.2. Реалізація методу штучного інтелекту в Unity

Unity ML-Agents - це відкрите програмне забезпечення для розробки імітаційних середовищ зі штучним інтелектом в середовищі Unity. Він надає інструменти для навчання агентів з підкріпленням, навчання зі зміщенням, навчання з вчителем і навчання без вчителя [8].

Unity ML-Agents дозволяє розробникам і дослідникам легко створювати власні імітаційні середовища з штучним інтелектом, де агенти можуть взаємодіяти з об'єктами в середовищі та навчатися розв'язувати різноманітні завдання. Він

також надає інструменти для візуалізації та аналізу навчального процесу, що допомагає вдосконалювати агентів і оптимізувати їх навчання [8].

На початку створив волейбольне майданчик для, подальшого налаштування штучного інтелекту в ігрові об'єкти:

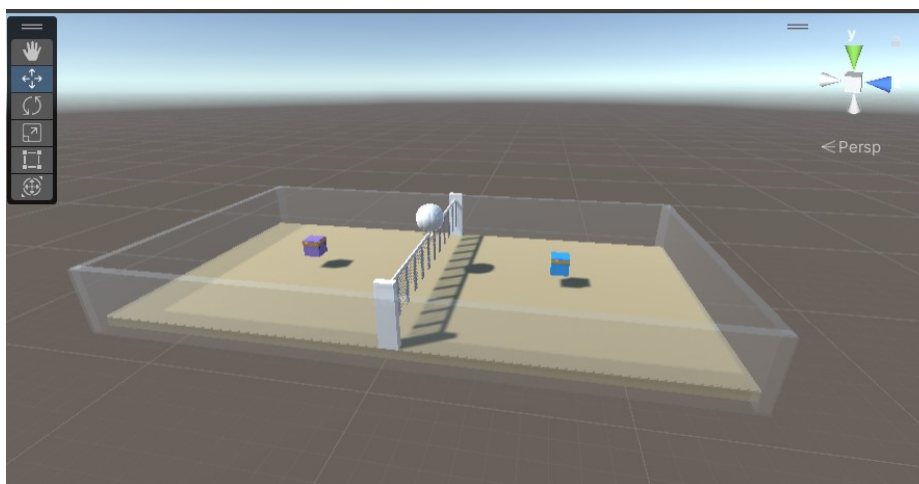


Рис. 2.1. Ігрове середовище в Unity

Голи представлені тонким шаром поверх землі.

Коли ігровий об'єкт встановлено як тригер, він більше не реєструє жодних зіткнень на основі фізики. Незважаючи на те, що цілі розташовані над земним шаром, технічно агенти рухаються на коллайдері наземного шару, який ми створили раніше.

Налаштування тригерів дозволяє нам використовувати `OnTriggerEnter` метод пізніше, який визначатиме, коли кулька вдарилася в коллайдер.

Є три невидимі межі:

- `OuterBoundaries` (перевірка виходу м'яча за межі)
- `BlueBoundary` (перевіряє, чи м'яч потрапляє на синю сторону корту)
- `PurpleBoundary` (перевіряє, чи м'яч потрапляє на фіолетову сторону корту)

Було додано сценарії, які визначають поведінку середовища (наприклад, що відбувається, коли м'яч вдаряється об підлогу або коли починається епізод).

VolleyballSettings.cs (Додаток А) – це перший сценарій, який буде просто містити деякі константи, які ми будемо повторно використовувати протягом усього проекту. VolleyballController.cs (Додаток Б) – це сценарій додається до ігрового об'єкта і дає нам змогу визначити, коли м'яч потрапив у наш кордон або тригер воріт.

В цьому сценарії використав методи Start () та OnTriggerEnter ().

Start () (Рис. 2.2.) – це метод викликається під час першого відтворення середовища.

Отримані коллайдери PurpleGoal і BlueGoal (компоненти, які реєструють зіткнення на основі фізики) за допомогою GetComponent<Collider> методу та призначив батьківський ігровий об'єкт VolleyballArea змінній envController для полегшення використання пізніше.

```
void Start()
{
    envController = GetComponentInParent<VolleyballEnvController>();
    purpleGoalCollider = purpleGoal.GetComponent<Collider>();
    blueGoalCollider = blueGoal.GetComponent<Collider>();
}
```

Рис. 2.2. Метод Start ()

OnTriggerEnter () (Рис. 2.3.) – це метод викликається, коли м'яч потрапляє на коллайдер.

Деякі сценарії для виявлення:

- М'яч вдаряється об підлогу/голи;
- М'яч виходить за межі поля;
- М'яч перекидається через сітку (щоб заохотити м'яч до тренування пізніше).

Цей метод виявить кожен сценарій і передасть цю інформацію envController.

```

void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("boundary"))
    {
        // м'яч вийшов за межі поля
        envController.ResolveEvent(Event.HitOutOfBounds);
    }
    else if (other.gameObject.CompareTag("blueBoundary"))
    {
        // м'яч влучив у синю сторону
        envController.ResolveEvent(Event.HitIntoBlueArea);
    }
    else if (other.gameObject.CompareTag("purpleBoundary"))
    {
        // м'яч влучив у фіолетову сторону
        envController.ResolveEvent(Event.HitIntoPurpleArea);
    }
    else if (other.gameObject.CompareTag("purpleGoal"))
    {
        // м'яч влучив у фіолетові ворота (блакитна сторона поля)
        envController.ResolveEvent(Event.HitPurpleGoal);
    }
    else if (other.gameObject.CompareTag("blueGoal"))
    {
        // м'яч влучив у сині ворота (фіолетовий бічний майданчик)
        envController.ResolveEvent(Event.HitBlueGoal);
    }
}

```

Рис. 2.3. Метод OnTriggerEnter ()

VolleyballEnvController.cs (Додаток В) – це сценарій містить усю основну логіку середовища: максимальну кількість кроків, які він має виконувати, як мають з'явитися м'яч і агенти, коли має закінчитися епізод, як мають призначатися нагороди тощо.

В цьому сценарії використав методи FixedUpdate, ResetScene та ResolveEvent.

FixedUpdate (Рис. 2.4.) - Це викликається механізмом Unity кожного разу, коли відбувається оновлення кадру.

Це контролюватиме максимальну кількість оновлень (тобто «кроків»), які виконує середовище, перш ніж ми перервемо епізод (наприклад, якщо кулька десь застрягне).

```

void FixedUpdate()
{
    resetTimer += 1;
    if (resetTimer >= MaxEnvironmentSteps && MaxEnvironmentSteps > 0)
    {
        blueAgent.EpisodeInterrupted();
        purpleAgent.EpisodeInterrupted();
        ResetScene();
    }
}

```

Рис. 2.4. Метод FixedUpdate ()

ResetScene (Рис. 2.5.) керує початковою поведінкою відродження.

Наша мета — вивчити модель, яка дозволяє нашому агенту повертати м'яч зі свого боку майданчика незалежно від того, куди м'яч відправляється. Щоб допомогти з тренуваннями, ми рандомізуємо стартові умови агентів і м'яча в деяких розумних межах

```

public void ResetScene()
{
    resetTimer = 0;

    lastHitter = Team.Default; // скинути останнього гравця

    foreach (var agent in AgentsList)
    {
        // випадкові вихідні позиції та повороти
        var randomPosX = Random.Range(-2f, 2f);
        var randomPosZ = Random.Range(-2f, 2f);
        var randomPosY = Random.Range(0.5f, 3.75f); // залежить від висоти стрибка
        var randomRot = Random.Range(-45f, 45f);

        agent.transform.localPosition = new Vector3(randomPosX, randomPosY, randomPosZ);
        agent.transform.eulerAngles = new Vector3(0, randomRot, 0);

        agent.GetComponent<Rigidbody>().velocity = default(Vector3);
    }

    // повернути м'яч до початкових умов
    ResetBall();
}

```

Рис. 2.5. Метод ResetScene ()

ResolveEvent () (Рис. Д.1.) – це метод, який вирішить сценарії, які ми визначили раніше в VolleyballController.cs.

Ми можемо використовувати цей метод для призначення винагород різними способами, щоб заохотити різні типи поведінки. Загалом, добре тримати винагороди в межах $[-1,1]$.

Якщо говорити простіше, наша мета наразі — навчити агентів, які можуть відбивати м'яч туди-сюди та підтримувати м'яч у грі. Ми призначатимемо

винагороду +1 кожного разу, коли агент перекине м'яч через сітку, використовуючи метод `AddReward(1f)` у відповідному сценарії.

Наразі ми не призначатимемо жодних винагород, якщо забитий гол або м'яч вибитий за межі поля. Якщо станеться будь-який із цих сценаріїв, ми просто завершимо серію (Рис. Д.1.).

Ми хочемо, щоб наш агент навчився, які дії виконувати в певному стані навколишнього середовища — наприклад, якщо м'яч знаходиться на нашому боці майданчика, наш агент повинен дістати його до того, як він впаде на підлогу.

Метою навчання з підкріпленням є вивчення найкращої політики, яка максимізує можливі винагороди. Теорія того, як алгоритми навчання з підкріпленням досягають цього, виходить за рамки цієї серії, але курси, якими я поділився у вступі до серії, розкривають це дуже глибоко.

Під час навчання агент виконуватиме дії:

1. Навмання (щоб визначити, які дії призводять до винагороди, а які ні)
2. З його поточної політики (оптимальна дія з урахуванням поточного стану)

ML-Agents надає зручний компонент `Decision Quester`, який оброблятиме чергування між ними під час навчання

`VolleyballAgent.cs` (Додаток Г) – це сценарій містить усю логіку, яка визначає дії та спостереження агентів. Він уже містить деякі допоміжні методи:

- `Start` – викликається під час першого відтворення середовища. Захоплює батьківське середовище `Volleyball` і зберігає його у змінній `envController` для легкого посилання на його методи пізніше.
- `Initialize` – викликається під час першої ініціалізації агента. Захоплює деякі корисні константи та об'єкти. Також встановлюється `agentRot` для забезпечення симетрії, щоб обидва агенти могли спільно використовувати ту саму політику.

- MoveTowards, CheckIfGrounded та Jump – із прикладів проектів ML-Agents. Використовується для стрибків.
- OnCollisionEnter – викликається, коли Агент з чимось стикається. Використовується для оновлення lastHitter, щоб визначити, який агент отримує штраф, якщо м'яч потрапляє за межі поля, або винагороджується, якщо його потрапляють через сітку.

Додавання агента в Unity ML-Agents зазвичай передбачає розширення базового Agent класу та впровадження таких методів:

- OnActionReceived ();
- Heuristic () (Рис. Д.3.);
- CollectObservation () (Рис. Д.4.);
- OnEpisodeBegin () (Зазвичай використовується для скидання початкових умов. Я не реалізував це тут, оскільки логіка скидання вже визначена на рівні середовища в VolleyballEnvController. Це має більше сенсу, оскільки нам також потрібно скинути кульку на додаток до агенти.).

На високому рівні особа, яка запитує рішення, вибере дію, яку має виконати наш агент, і ініціює її OnActionReceived. Це в свою чергу викликає MoveAgent.

MoveAgent – це метод, який вирішує вибрану дію.

У цьому MoveAgent методі почав з оголошення векторних змінних для напрямків і обертальних рухів наших агентів а також додав «заземлену» перевірку, щоб перевірити, чи можливий стрибок агента.

```
public void MoveAgent(ActionSegment<int> act)
{
    var grounded = CheckIfGrounded();
    var dirToGo = Vector3.zero;
    var rotateDir = Vector3.zero;
```

Рис. 2.6. «Заземлена» перевірка та оголошення векторних змінних

Дії, передані в цей метод (actionBuffers.DiscreteActions), будуть масивом цілих чисел, які ми будемо зіставляти з деякою поведінкою. Неважливо, який порядок ми їм призначаємо, якщо вони залишаються послідовними:

```

var dirToGoForwardAction = act[0];
var rotateDirAction = act[1];
var dirToGoSideAction = act[2];
var jumpAction = act[3];

```

Рис. 2.7. Масиви цілих чисел

В Unity кожен об'єкт має transform клас, який зберігає його положення, обертання та масштаб. Ми будемо використовувати його, щоб створити вектор, що вказує на правильний напрямок, у якому ми хочемо, щоб рухався наш агент.

Виходячи з попереднього порядку призначення, ось як ми зіставимо наші дії з поведінкою:

1. dirToGoForwardAction (Рис. 2.8.): Нічого не робити=0 | Рух вперед=1 | Рух назад=2
2. rotateDirAction (Рис. 2.8.): Нічого не робити=0 | Повернути за годинниковою стрілкою=1 | Обертання проти годинникової стрілки=2
3. dirToGoSideAction (Рис. 2.8.): Нічого не робити=0 | Рух ліворуч=1 | Рух праворуч=2
4. jumpAction (Рис. 2.8.): Не стрибай=0 | Стрибок=1

Зіставлення дії з поведінкою:

```

if (dirToGoForwardAction == 1)
    dirToGo = (grounded ? 1f : 0.5f) * transform.forward * 1f;
else if (dirToGoForwardAction == 2)
    dirToGo = (grounded ? 1f : 0.5f) * transform.forward * volleyballSettings.speedReductionFactor * -1f;

if (rotateDirAction == 1)
    rotateDir = transform.up * -1f;
else if (rotateDirAction == 2)
    rotateDir = transform.up * 1f;

if (dirToGoSideAction == 1)
    dirToGo = (grounded ? 1f : 0.5f) * transform.right * volleyballSettings.speedReductionFactor * -1f;
else if (dirToGoSideAction == 2)
    dirToGo = (grounded ? 1f : 0.5f) * transform.right * volleyballSettings.speedReductionFactor;

if (jumpAction == 1)
    if (((jumpingTime <= 0f) && grounded)){
        Jump();
    }

```

Рис. 2.8. Метод MoveAgent

Далі застосував рух за допомогою наданих Unity методів Rotate і AddForce (Рис. Д.2.) та додав логіку для керування поведінкою стрибка.

Щоб перевірити, чи правильно ми вирішили дії, реалізував метод `Heuristic ()` (Рис. Д.3.). Це відобразить дії на введення з клавіатури, щоб ми могли перевірити гру як людина-контролер.

Спостереження – це те, як наш агент «бачить» своє оточення.

У `ML-Agents` є 3 типи спостережень, які ми можемо використовувати:

- Вектори – «пряма» інформація про наше оточення (наприклад, список поплавків, що містить положення, масштаб, швидкість тощо об'єктів)
- `Raycasts` – «промені», які виходять від агента та виявляють об'єкти поблизу
- Візуальний/камерний вхід

У цьому проєкті я запровадив векторні спостереження, щоб усе було просто. Мета полягає в тому, щоб включити лише ті спостереження, які є доречними для прийняття обґрунтованого рішення про те, як діяти.

Щоб додати спостереження, я реалізував метод класу агента `CollectObservations ()` (Рис. Д.4.).

Проксимальна оптимізація політики (PPO) — це алгоритм навчання з посиленням політики. Я вирішив використовувати його тут, оскільки `ML-Agents` забезпечує реалізацію його з коробки. Він дає стабільні результати в цьому середовищі, а також рекомендований `ML-Agents` для використання з `Self-Play`.

Після того, як завершили налаштування агентів, запустив навчання (Рис. Д.5.) та залишив їх тренуватися. Приблизно через 5 мільйонів агентів час від часу торкалися м'яча. На 10 млн агенти почали кидати з льоту. А зробивши приблизно 20 мільйонів кроків, агенти успішно перекидали м'яч вперед та назад.

Результати навчання штучного інтелекту з підкріпленням з алгоритмом проксимальної оптимізації політики:

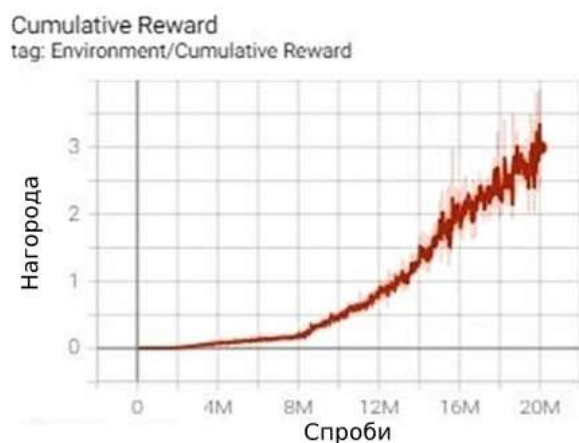


Рис. 2.9. Результат навчання (сума нагород)

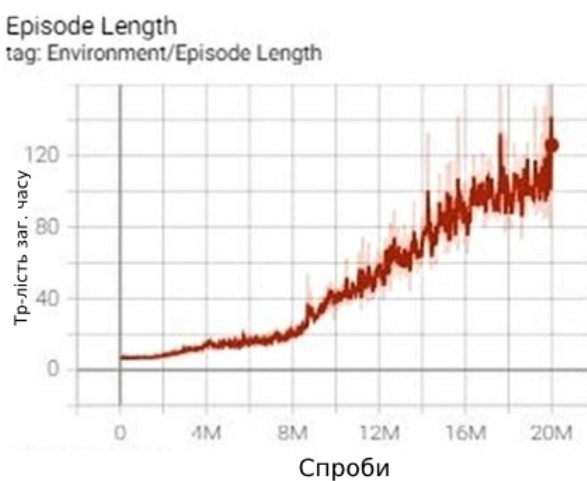


Рис. 2.10. Результат навчання (тр-лість з. часу кожної спроби агента)

На першому графіку (Рис. 2.9.) відображається сума нагород, яку отримує агент за кожну спробу. А на другому графіку (Рис. 2.10.) можна побачити тривалість загального часу кожної спроби агента.

РОЗДІЛ 3. ПРОПОЗИЦІЇ ЩОДО УДОСКОНАЛЕННЯ (ПОКРАЩЕННЯ) ОДЕРЖАНИХ РЕЗУЛЬТАТІВ

3.1. Порівняння результатів навчання штучного інтелекту з іншими роботами.

Було порівнювано одержані результати з такою роботою, як “An Evaluation of the Unity Machine Learning Agents Toolkit in Dense and Sparse Reward Video Game Environments” від авторів Jari Hanksi, Biçak, Kaan Baris [21].

Для перевірки методів підкріплення було створено щільне та розріджене навчальне середовище винагород. Щільне середовище винагород — це гоночна траса (Рис. 3.1.), на якій агент отримує винагороду за проходження ділянок траси та штраф за зіткнення зі стінами. Середовище розрідженої винагороди — це місто (Рис. 3.2.), де агент повинен забрати пасажирів і повернути їх в одне конкретне місце. Агент отримує винагороду лише після виконання всього завдання.

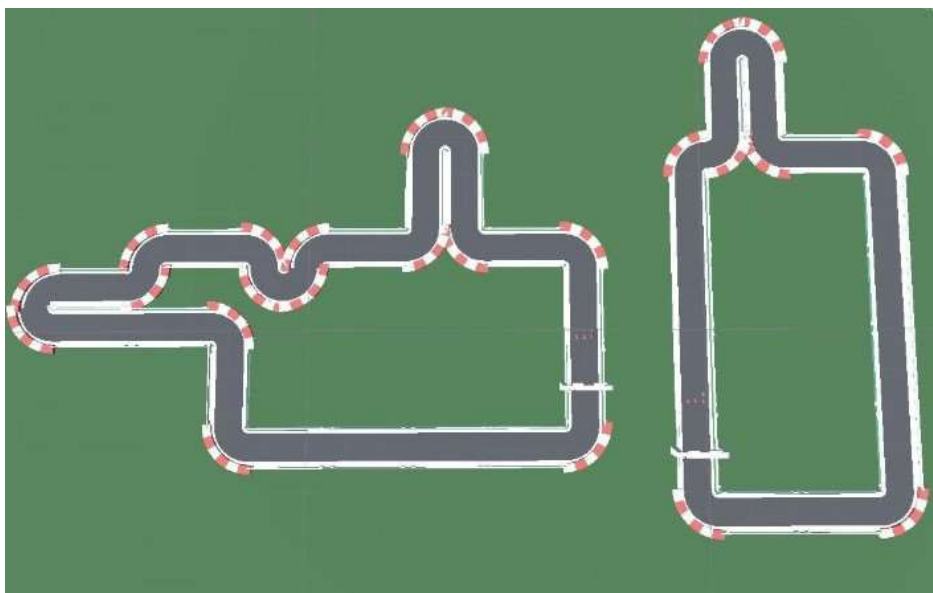


Рис. 3.1. Дві тренувальні доріжки в навчальному середовищі.

Навчальне 3D-середовище було створено в ігровому движку Unity, як показано на Рис. 3.1.. Воно має дві різні тренувальні доріжки з унікальною формою та довжиною. Форма доріжки була обрана довільно. Основна відмінність полягає в тому, що на іншій доріжці машини рухаються за годинниковою стрілкою, а на іншій - проти годинникової стрілки. Також було створено дві тренувальні траси, щоб запобігти переобладнанню моделі, оскільки переобладнання може призвести до того, що агенти гоночних автомобілів не зможуть пройти треки, яких вони раніше не бачили. На кожну з тренувальних трас виділено по чотири машини, які контролюються агентами.

Ще одне 3D-середовище було створено з рідкісними винагородами (Рис. 3.2.). Мета агента - знайти пасажирів в місті і повернути пасажирів до мети. Позиція пасажирів випадкова, а цільова позиція статична.

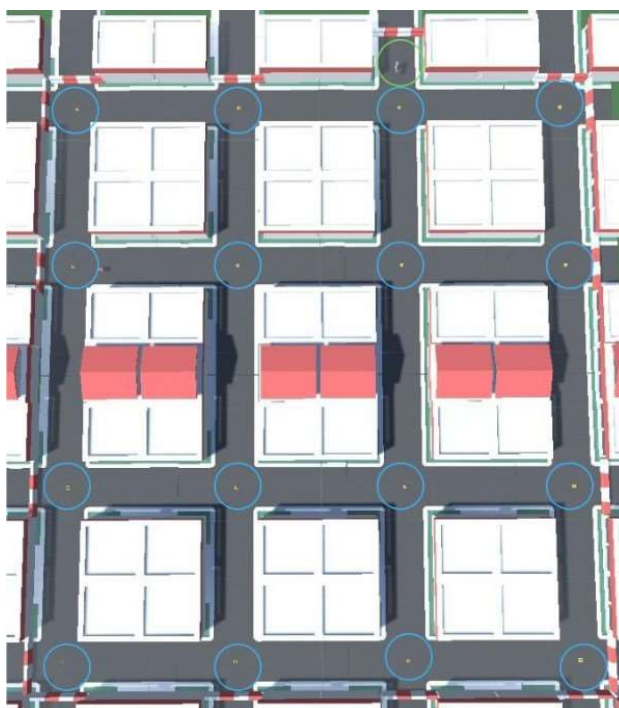


Рис. 3.2. Середовище навчання таксі. Сині кружечки — можливі позиції для пасажирів. Зелене коло — пункт призначення пасажирів.

Агентів було оцінено за сукупною винагородою, яку вони отримували в процесі навчання. Період збору даних проводився до досягнення максимальних кроків під час навчання або після стабілізації сукупних винагород. Чим раніше загальна винагорода стабілізувалася, тим швидше проходило навчання.

Середовище гоночної траси діє як тестове середовище, де не повинно бути великих переваг від використання допоміжних методів посилення з проксимальною оптимізацією політики. Міське середовище є фактичним тестовим середовищем, де очікується підвищення результатів навчання порівняно з лише РРО.

Навчання агентів навчання з підкріпленням проводилося в двох різних тестових середовищах. Середовище гоночної траси включало велику кількість винагород у вигляді контрольних точок для автомобілів. Міське середовище було розрідженим середовищем винагороди, яке не містило жодної винагороди, окрім того, коли агент досягав мети. Агенти були навчені в обох середовищах з використанням різних методів навчання з підкріпленням. Навчальний прогрес агентів спостерігався та оцінювався за даними, зібраними під час навчання. Кумулятивні винагороди відображаються в залежності від часу, який минув протягом навчання. Часовий крок тут означає один цикл навчання, протягом якого агент виконує одну дію та отримує винагороду.

Під час навчання використовувалися різні комбінації методів навчання, а також аналізували та порівнювали час навчання та сукупні винагороди. Час навчання вимірюється в часових кроках, а методи навчання оцінюються на основі кумулятивних винагород, отриманих агентами, і того, чи зміг агент вирішити проблему за максимальні часові кроки. Навчання нейронної мережі проводилося паралельно, щоб прискорити процес навчання.

Для проксимальної оптимізації політики (РРО) використовувалося такі допоміжні методи:

1. Random network distillation;

2. Behavior Cloning;
3. Curiosity;
4. Generative Adversarial Imitation Learning;
5. Generative Adversarial Imitation Learning + Behavior Cloning + Curiosity.

Кожен рядок у діаграмі представляє різні методи навчання:

- Помаранчевий: PPO + RND
- Синій: PPO
- Червоний: PPO + Curiosity
- Світло-блакитний: PPO + Behavior Cloning
- Рожевий: GAIL + Curiosity
- Сірий: PPO + GAIL

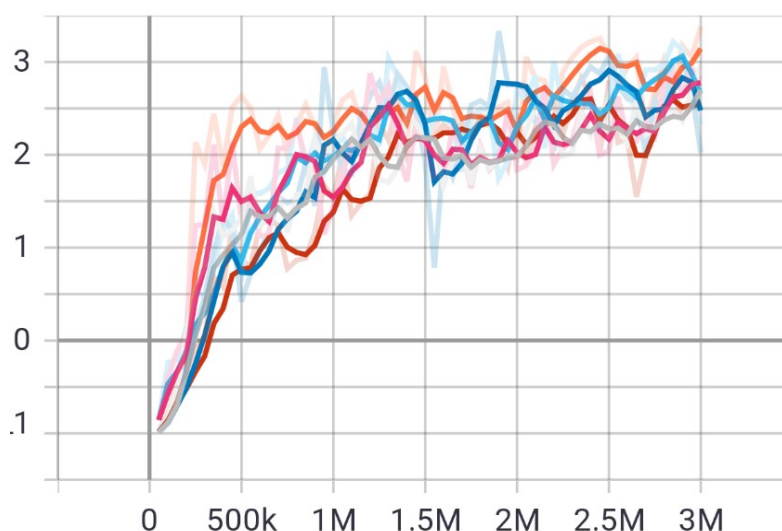


Рис. 3.3. Середовище гоночної траси: накопичувальна винагорода

Агент PPO + RND показав найкращі результати та отримав високу сукупну винагороду за найкоротший час. Найповільнішим агентом був PPO + Curiosity. Кожен із агентів навчився виконувати завдання однаково, і найшвидший агент пройшов навчання за 500 тисяч кроків, а найповільніший – за 1,5 мільйона кроків. Окрім невеликих відмінностей у продуктивності, між методами немає великих відмінностей.

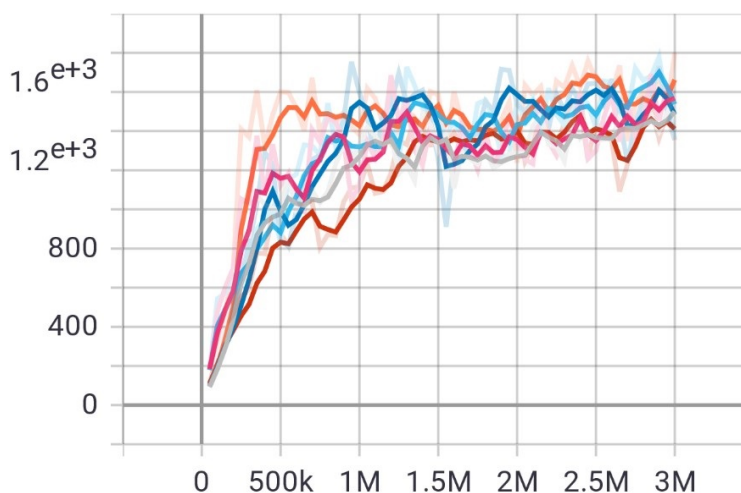


Рис. 3.4. Середовище гоночної траси: тривалість епізоду

Агент PPO + RND досяг найдовшої тривалості епізоду раніше в групі, тобто він зміг пройти весь трек, не зіткнувшись зі стінами раніше, ніж інші агенти. Агент PPO + RND був повністю навчений лише після 500 тис. часових кроків. PPO показав найгірші результати серед усіх агентів, і він отримує високі сукупні винагороди приблизно за 1,5 мільйона часових кроків. Наприкінці навчання немає великих відмінностей у продуктивності кожного методу, і кожен з агентів виконує завдання за однаковий час.

Результати навчання в середовищі міської траси. Кожен рядок у діаграмі представляє різні методи навчання:

- Помаранчевий: PPO + GAIL + BC + Curiosity
- Синій: PPO + GAIL + Curiosity
- Червоний: PPO
- Світло-блакитний: PPO + Curiosity
- Рожевий: PPO + RND

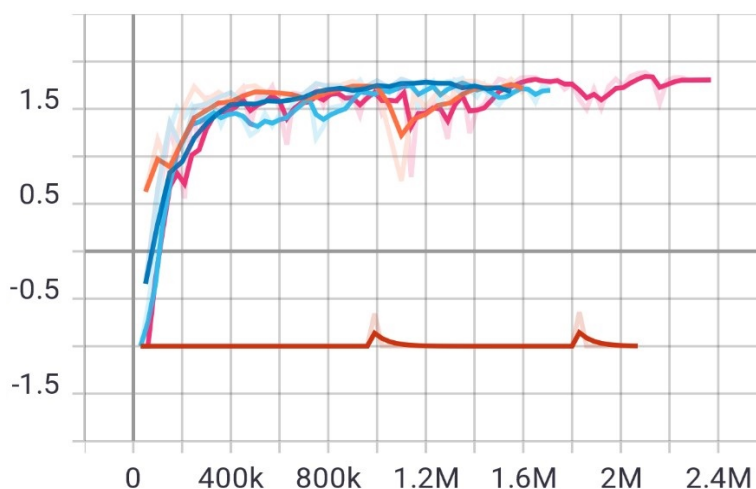


Рис. 3.5. Міське середовище: накопичувальна винагорода

Як видно на рисунку 3.5., навчання проводилося з використанням різних методів. Навчання було зупинено після приблизно 1,6-2 мільйонів кроків, коли сукупні винагороди стали рівними.

У міському середовищі лінія червоного агента є рівною та отримує постійні негативні винагороди, що означає, що він не зміг досягти мети, а навчання не вдалося.

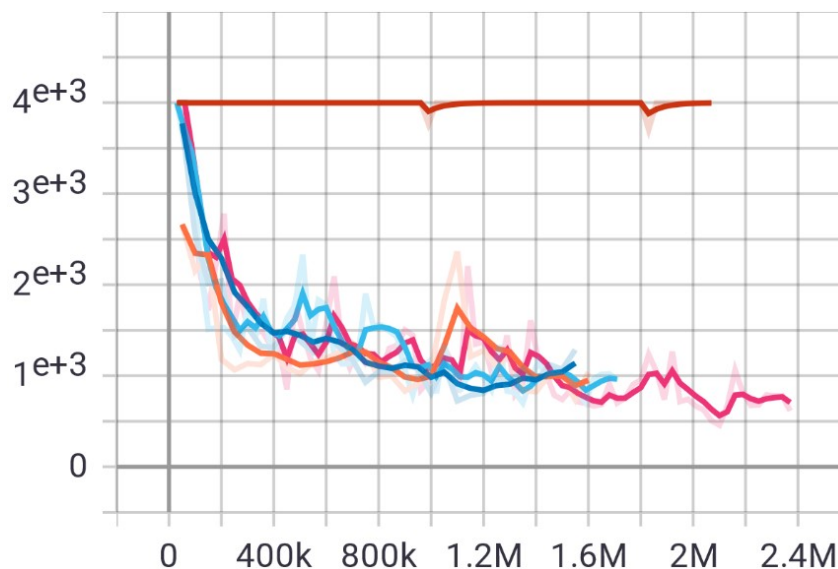


Рис. 3.6. Міське середовище: тривалість епізоду

Як видно на рисунку 3.6., тривалість епізоду стала меншою, коли більшість агентів навчилися виконувати завдання швидше. Червоний агент витратив

максимально дозволена тривалість епізоду на те, щоб знайти нагороди. Це ознака того, що червоний агент так і не зміг виконати свою мету і шукав нагороду випадковим чином.

3.2. Перспективи кваліфікаційної роботи

Порівнюючи отримані результати з цією роботою, ми можемо запропонувати наступні удосконалення:

- Розширити обсяг експериментальних досліджень, використовуючи більш широкий набір тестових даних.
- Вдосконалити методологію аналізу результатів, враховуючи додаткові метрики та статистичні показники.
- Провести додаткові експерименти з різними конфігураціями параметрів алгоритму для виявлення оптимальних налаштувань.
- Враховувати розрізнені фактори, які можуть вплинути на результати, такі як обсяг даних, розмір вибірки тощо.
- Провести порівняльний аналіз з альтернативними алгоритмами, щоб визначити переваги та обмеження розробленого підходу.
- Розглянути можливість подальшої оптимізації алгоритму для підвищення точності та швидкодії.

Ці пропозиції можуть сприяти подальшому удосконаленню отриманих результатів та покращенню ефективності дослідження.

ВИСНОВКИ

У даній дипломній роботі було проведено розробку ігрового штучного інтелекту з використанням рушія Unity. Досліджена предметна область, проведений аналіз існуючих рішень, методів та алгоритмів, пов'язаних з штучним інтелектом в ігрових застосунках.

У розділі аналізу предметної області було проведено ретельний огляд етапів розвитку штучного інтелекту та його застосування в комп'ютерних іграх. Були розглянуті різні аспекти та види штучного інтелекту. Також було проведено огляд сучасних ігрових рушіїв, що підтримують штучний інтелект. Цей аналіз надав важливу базу для подальшого проектування та реалізації штучного інтелекту в ігровому застосунку.

У розділі проектування та реалізації штучного інтелекту було проаналізовано та використано різні методи та алгоритми для успішної реалізації штучного інтелекту в середовищі Unity. Було проведено детальне проектування архітектури системи, вибір необхідного алгоритму та його реалізацію. Застосований алгоритм має у своєму складі алгоритми прийняття рішень, машинного навчання та імітації поведінки. Результатом роботи є успішна реалізація штучного інтелекту, що забезпечує більш реалістичний та цікавий геймплей у ігровому застосунку.

Навчання з підкріпленням - один із способів взаємодії, в ході якого випробувана система (агент) навчається, взаємодіючи із середовищем, отримуючи після будь-яких взаємодій з середовищем або шкоду, або нагороду. Було вирішено використовувати проксимальну оптимізацію політики, оскільки ML-Agents забезпечує реалізацію його «з коробки». Він дає стабільні результати в цьому середовищі, а також рекомендований ML-Agents для використання з Self-Play.

Пропозиції щодо удосконалення одержаних результатів включають порівняння результатів навчання штучного інтелекту з іншими роботами в галузі, а також розгляд перспектив кваліфікаційної роботи для подальшого розвитку штучного інтелекту в ігровій сфері. Додаткові дослідження та вдосконалення можуть сприяти покращенню штучного інтелекту в ігрових застосунках та впровадженню нових інноваційних рішень.

В цілому, робота демонструє успішну реалізацію ігрового штучного інтелекту з використанням рушія Unity та надає підґрунтя для подальших досліджень і в цьому напрямку. Результати даної роботи можуть бути використані для створення більш реалістичних та цікавих ігрових додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бабори́га Р. С., Система ігрового штучного інтелекту на базі Unity. Київ, 2021. URL: https://er.nau.edu.ua/bitstream/NAU/52110/1/%D0%A4%D0%9A%D0%9A%D0%9F%D0%86_2021_123_%D0%91%D0%B0%D0%B1%D0%BE%D1%80%D0%B8%D0%B3%D0%B0%D0%A0%D0%A1.pdf (дата звернення: 26.03.2023).
2. Велігурський О. С., Розробка підсистеми ігрового штучного інтелекту. Київ, 2020. URL: <http://ekmair.ukma.edu.ua/handle/123456789/18245?locale-attribute=uk> (дата звернення: 26.03.2023).
3. Гаранін О. М., Моїсеєнко Н. В., Адаптивний штучний інтелект у RPG-грі на ігровому рушії Unity. Кривий ріг, 2018. URL: <https://ceur-ws.org/Vol-2292/paper16.pdf> (дата звернення: 26.03.2023).
4. Глинська К.С., Реалізація штучного інтелекту для гри в шахи. Покровськ, 2017. URL: http://irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/Npdntu_inf_2017_2_6.pdf (дата звернення: 26.03.2023).
5. Качура М., Ігри, у які грають роботи. Навіщо штучний інтелект вчать грі у шахи й Dota 2. 2021. URL: <https://nauka.ua/article/igri-u-yaki-grayut-roboti-navishcho-shi-vchat-gri-u-shahi-j-dota-2-ta-recept-peremog-vid-algoritmiv> (дата звернення: 26.03.2023).
6. Кривий. В. М. Програмна система моделювання складної поведінки ігрового штучного інтелекту : дипломна робота магістра: 121 Інженерія програмного забезпечення / В. М. Кривий ; Хмельниц. нац. ун-т – Хмельницький, 2021 – 116 с.. URL: <http://elar.khmnmu.edu.ua/jspui/handle/123456789/10969> (дата звернення: 26.03.2023).

7. Марченко М. А., Інформаційне та програмне забезпечення ігрової системи жанру казуальних головоломок. Суми, 2020. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/79518/1/Marchenko_%20Bachelous%20paper_paper.pdf (дата звернення: 26.03.2023).
8. Муравльов Р.Ф., Розробка та дослідження методів штучного інтелекту у ігрових застосунках. Харків, 2021. URL: <https://openarchive.nure.ua/server/api/core/bitstreams/0b292c09-1d40-4612-89ff-20bde2e5a4ce/content> (дата звернення: 26.03.2023).
9. Омельчук А.А., Якубовський В.С., Розробка ігрового штучного інтелекту засобами Unity. 2023. URL: https://www.researchgate.net/publication/370068210_ROZROBKA_IGROVOGO_STUCNOGO_INTELEKTU_ZASOBAMI_UNITY (дата звернення: 02.04.2023).
10. Под'ячев А. Д. Інтелектуальна система моделювання поведінки персонажів в комп'ютерних іграх. Миколаїв, 2022. URL: <https://krs.chmnu.edu.ua/jspui/bitstream/123456789/2328/1/601%D0%9F%D0%BE%D0%B4%60%D1%8F%D1%87%D0%B5%D0%B2%20%D0%90%D0%BD%D0%B4%D1%80%D1%96%D0%B9%20%D0%94%D0%B5%D0%BD%D0%B8%D1%81%D0%BE%D0%B2%D0%B8%D1%87.pdf> (дата звернення: 26.03.2023).
11. Романюк О. Н., Корягіна Д. О., Використання штучного інтелекту в іграх. Вінниця, 2022. URL: <https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/35633/%D0%9A%D0%BE%D1%80%D1%8F%D0%B3%D1%96%D0%BD%D0%B0.pdf?sequence=1&isAllowed=y> (дата звернення: 26.03.2023).
12. Солодкий Д. О., Система автоматизованого генерування VR-сцен. Київ, 2020. URL: https://ela.kpi.ua/bitstream/123456789/36544/1/Solodkiy_bakalavr.pdf (дата звернення: 26.03.2023).

13. Стефанюк О.П. Створення комп'ютерної гри з використанням ігрового рушія Unreal Engine 4.27 з елементами штучного інтелекту: кваліфікаційна робота освітнього рівня “Бакалавр” 122 — “Комп'ютерні науки” / О.П. Стефанюк — Тернопіль : ТНТУ, 2022. URL: <https://elartu.tntu.edu.ua/handle/lib/38264> (дата звернення: 02.04.2023).

14. Супруненко М.Ю. Методи забезпечення адаптивності ігрового штучного інтелекту. Харків, 2021. URL: <https://openarchive.nure.ua/server/api/core/bitstreams/9becbfed-5ce8-4d6f-aa48-2690e820651b/content> (дата звернення: 26.04.2023).

15. Філоненко А.К., Розробка програмного забезпечення для створення комп'ютерних ігор з використанням штучного інтелекту. Київ, 2018. URL: https://er.knutd.edu.ua/bitstream/123456789/11201/1/NRMSE2018_V2_P137-138.pdf (дата звернення: 26.03.2023).

16. Череповська С. М., Інформаційне та програмне забезпечення ігрової системи з елементами штучного інтелекту. Суми, 2020. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/82303/1/Cherepovska_mag_rob.pdf;jsessionid=08D5A8DB87EE D3B1F1A5EC2A47C6E174 (дата звернення: 26.03.2023).

17. Чернов І. І., Розробка 2D - гри на платформі Unity. Миколаїв, 2022. URL: <https://krs.chmnu.edu.ua/jspui/bitstream/123456789/2394/1/402%20%D0%A7%D0%B5%D1%80%D0%BD%D0%BE%D0%B2%20%D0%86%D0%BB%D0%BB%D1%8F%20%D0%86%D0%B3%D0%BE%D1%80%D0%BE%D0%B2%D0%B8%D1%87.pdf> (дата звернення: 26.03.2023).

18. Штучний інтелект у дизайні та розробці ігор. Де ми зараз і що далі?. 2022. URL: <https://www.imena.ua/blog/artificial-intelligence-in-design-and-gamedev/> (дата звернення: 26.03.2023).

19. Шпита Г. В., Методологія імплементації навчання з підкріпленням у відео ігровому штучному інтелекті. Київ, 2019. URL: https://ela.kpi.ua/bitstream/123456789/27795/1/Shpyta_magistr.pdf (дата звернення: 26.03.2023).
20. Belova K., How Artificial Intelligence (AI) Upends Game Development. 2021. URL: <https://pixelplex.io/blog/how-ai-enhances-game-development/> (дата звернення: 02.04.2023).
21. Bhattacharya S. Gaming AI: Unity with Deep Reinforcement Learning. 2021. URL: <https://medium.com/nerd-for-tech/gaming-ai-unity-with-deep-reinforcement-learning-e8be9d14f248> (дата звернення: 26.04.2023).
22. Hanski J., Biçak, Baris K. An Evaluation of the Unity Machine Learning Agents Toolkit in Dense and Sparse Reward Video Game Environments. 2021. URL: <https://www.diva-portal.org/smash/get/diva2:1563588/FULLTEXT01.pdf> (дата звернення: 26.04.2023).
23. Hocking J. Unity in Action: Multiplatform game development in C# – 2019. – 248 с. URL: <https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence> (дата звернення: 02.04.2023).
24. Schardon L. Best Unity AI Tutorials – Unity Game Development Guide. 2023. URL: <https://gamedevacademy.org/best-unity-ai-tutorials/> (дата звернення: 26.04.2023).
25. Schulman J., Klimov O., Wolski F., Dhariwal P., Radford A. Proximal Policy Optimization. 2017. URL: <https://openai.com/research/openai-baselines-ppo> (дата звернення: 26.03.2023).
26. Moontessa, Ігри в розвитку штучного розуму. 2018. URL: <https://igrovymir.com/2018/10/19/ai/> (дата звернення: 26.03.2023).
27. Zhang J. An introduction to machine learning with Unity ML-Agents. 2021. URL: <https://www.gocoder.one/blog/introduction-to-unity-ml-agents/> (дата звернення: 26.04.2023).

ДОДАТКИ

Додаток А

Файл VolleyballSettings.cs

```
using UnityEngine;
public class VolleyballSettings : MonoBehaviour
{
    public float agentRunSpeed = 1.5f;
    public float agentJumpHeight = 2.75f;
    public float agentJumpVelocity = 777;
    public float agentJumpVelocityMaxChange = 10;
    // Уповільнює стрейф і рух назад
    public float speedReductionFactor = 0.75f;
    public Material blueGoalMaterial;
    public Material purpleGoalMaterial;
    public Material defaultMaterial;
    // Це спрямована вниз сила, яка прикладається під час падіння, щоб стрибки виглядали менш плаваючими
    public float fallingForce = 150;
}
```

Додаток Б

Файл VolleyballSettings.cs

```

using UnityEngine;
public class VolleyballController : MonoBehaviour
{
    [HideInInspector]
    public VolleyballEnvController envController;
    public GameObject purpleGoal;
    public GameObject blueGoal;
    Collider purpleGoalCollider;
    Collider blueGoalCollider;
    void Start()
    {
        envController = GetComponentInParent<VolleyballEnvController>();
        purpleGoalCollider = purpleGoal.GetComponent<Collider>();
        blueGoalCollider = blueGoal.GetComponent<Collider>();
    }
    void OnTriggerEnter(Collider other)
    {
        {
            if (other.gameObject.CompareTag("boundary"))
            {
                // м'яч вийшов за межі поля
                envController.ResolveEvent(Event.HitOutOfBounds);
            }
            else if (other.gameObject.CompareTag("blueBoundary"))
            {
                // м'яч влучив у синню сторону
                envController.ResolveEvent(Event.HitIntoBlueArea);
            }
            else if (other.gameObject.CompareTag("purpleBoundary"))
            {
                // м'яч влучив у фіолетову сторону
                envController.ResolveEvent(Event.HitIntoPurpleArea);
            }
            else if (other.gameObject.CompareTag("purpleGoal"))
            {
                // м'яч влучив у фіолетові ворота (блакитна сторона поля)
                envController.ResolveEvent(Event.HitPurpleGoal);
            }
            else if (other.gameObject.CompareTag("blueGoal"))
            {
                // м'яч влучив у синні ворота (фіолетовий бічний майданчик)
                envController.ResolveEvent(Event.HitBlueGoal);
            }
        }
    }
}

```

Додаток В

Файл VolleyballEnvController.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public enum Team{
    Blue = 0,
    Purple = 1,
    Default = 2
}
public enum Event{
    HitPurpleGoal = 0,
    HitBlueGoal = 1,
    HitOutOfBounds = 2,
    HitIntoBlueArea = 3,
    HitIntoPurpleArea = 4
}
public class VolleyballEnvController : MonoBehaviour{
    int ballSpawnSide;
    VolleyballSettings volleyballSettings;
    public VolleyballAgent blueAgent;
    public VolleyballAgent purpleAgent;
    public List<VolleyballAgent> AgentsList = new List<VolleyballAgent>();
    List<Renderer> RenderersList = new List<Renderer>();
    Rigidbody blueAgentRb;
    Rigidbody purpleAgentRb;
    public GameObject ball;
    Rigidbody ballRb;
    public GameObject blueGoal;
    public GameObject purpleGoal;
    Renderer blueGoalRenderer;
    Renderer purpleGoalRenderer;
    Team lastHitter;
    private int resetTimer;
    public int MaxEnvironmentSteps;
    void Start(){
        // Використовується для керування стартовими позиціями агента та м'яча
        blueAgentRb = blueAgent.GetComponent<Rigidbody>();
        purpleAgentRb = purpleAgent.GetComponent<Rigidbody>();
        ballRb = ball.GetComponent<Rigidbody>();
    }
}

```

```

// Початкова сторона появи м'яча
// -1 = поява синьої сторони, 1 = поява фіолетової сторони
var spawnSideList = new List<int> { -1, 1 };
ballSpawnSide = spawnSideList[Random.Range(0, 2)];
// Візуалізація землі, для того, щоб візуалізувати, який агент забив
blueGoalRenderer = blueGoal.GetComponent<Renderer>();
purpleGoalRenderer = purpleGoal.GetComponent<Renderer>();
RenderersList.Add(blueGoalRenderer);
RenderersList.Add(purpleGoalRenderer);
volleyballSettings = FindObjectOfType<VolleyballSettings>();
ResetScene();
}
// Відстежує, який агент останнім контролював м'яч
public void UpdateLastHitter(Team team){
lastHitter = team;
}
// Вирішує сценарії, коли м'яч потрапляє на тригер, і призначає нагороди
public void ResolveEvent(Event triggerEvent){
switch (triggerEvent){
case Event.HitOutOfBounds:
if (lastHitter == Team.Blue){
// застосувати штраф до синього агента
}
else if (lastHitter == Team.Purple){
// застосувати штраф до фіолетового агента
}
}
// завершити епізод
blueAgent.EndEpisode();
purpleAgent.EndEpisode();
ResetScene();
break;
case Event.HitBlueGoal:
// перемагає синій
// зробити підлогу синьою
StartCoroutine(GoalScoredSwapGroundMaterial(volleyballSettings.blueGoalMaterial, RenderersList, .5f));
// завершити епізод
blueAgent.EndEpisode();
purpleAgent.EndEpisode();
ResetScene();
break;
case Event.HitPurpleGoal:

```

```

        // фіолетовий перемагає
        // зробити підлогу фіолетовим
        StartCoroutine(GoalScoredSwapGroundMaterial(volleyballSettings.purpleGoalMaterial, RenderersList, .5f));
        // завершити епізод
        blueAgent.EndEpisode();
        purpleAgent.EndEpisode();
        ResetScene();
        break;
    case Event.HitIntoBlueArea:
        if (lastHitter == Team.Purple){
            purpleAgent.AddReward(1);
        }
        break;
    case Event.HitIntoPurpleArea:
        if (lastHitter == Team.Blue){
            blueAgent.AddReward(1);
        }
        break;
    }
}

IEnumerator GoalScoredSwapGroundMaterial(Material mat, List<Renderer> rendererList, float time){
    foreach (var renderer in rendererList){
        renderer.material = mat;
    }
    yield return new WaitForSeconds(time); // wait for 2 sec
    foreach (var renderer in rendererList){
        renderer.material = volleyballSettings.defaultMaterial;
    }
}

void FixedUpdate(){
    resetTimer += 1;
    if (resetTimer >= MaxEnvironmentSteps && MaxEnvironmentSteps > 0){
        blueAgent.EpisodeInterrupted();
        purpleAgent.EpisodeInterrupted();
        ResetScene();
    }
}

// Скидання умови появи агента та кулі.
public void ResetScene(){
    resetTimer = 0;
    lastHitter = Team.Default; // скинути останнього гравця
}

```

```

foreach (var agent in AgentsList){
    // випадкові вихідні позиції та повороти
    var randomPosX = Random.Range(-2f, 2f);
    var randomPosZ = Random.Range(-2f, 2f);
    var randomPosY = Random.Range(0.5f, 3.75f);
    // залежить від висоти стрибка
    var randomRot = Random.Range(-45f, 45f);
    agent.transform.localPosition = new Vector3(randomPosX, randomPosY, randomPosZ);
    agent.transform.eulerAngles = new Vector3(0, randomRot, 0);
    agent.GetComponent<Rigidbody>().velocity = default(Vector3);
}
// повернути м'яч до початкових умов
ResetBall();
}
// Скидання умови появи м'яча
void ResetBall(){
    var randomPosX = Random.Range(-2f, 2f);
    var randomPosZ = Random.Range(6f, 10f);
    var randomPosY = Random.Range(6f, 8f);
    // альтернативна сторона появи м'яча
    // -1 = поява синьої сторони, 1 = поява фіолетової сторони
    ballSpawnSide = -1 * ballSpawnSide;
    if (ballSpawnSide == -1){
        ball.transform.localPosition = new Vector3(randomPosX, randomPosY, randomPosZ);
    }
    else if (ballSpawnSide == 1){
        ball.transform.localPosition = new Vector3(randomPosX, randomPosY, -1 * randomPosZ);
    }
    ballRb.angularVelocity = Vector3.zero;
    ballRb.velocity = Vector3.zero;
}
}

```

Додаток Г

Файл VolleyballAgent.cs

```
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Policies;

public class VolleyballAgent : Agent
{
    public GameObject area;
    Rigidbody agentRb;
    BehaviorParameters behaviorParameters;
    public Team teamId;

    // Щоб отримати місцезнаходження м'яча для спостережень
    public GameObject ball;
    Rigidbody ballRb;

    VolleyballSettings volleyballSettings;
    VolleyballEnvController envController;

    // Контролює поведінку під час стрибків
    float jumpingTime;
    Vector3 jumpTargetPos;
    Vector3 jumpStartingPos;
    float agentRot;

    public Collider[] hitGroundColliders = new Collider[3];
    EnvironmentParameters resetParams;

    void Start()
    {
        envController = area.GetComponent<VolleyballEnvController>();
    }

    public override void Initialize()
    {
        volleyballSettings = FindObjectOfType<VolleyballSettings>();
        behaviorParameters = gameObject.GetComponent<BehaviorParameters>();
    }
}
```

```

agentRb = GetComponent<Rigidbody>();
ballRb = ball.GetComponent<Rigidbody>();

// для симетрії між гравцями
if (teamId == Team.Blue)
{
    agentRot = -1;
}
else
{
    agentRot = 1;
}
}

/// <summary>
/// Плавно переміщує тверде тіло в позицію.
/// <param name="targetPos">Цільова позиція.</param>
/// <param name="rb">Тверде тіло, яке потрібно перемістити.</param>
/// <param name="targetVel">Швидкість цілі під час
/// рух.</param>
/// <param name="maxVel">Максимально можлива швидкість.</param>
void MoveTowards(
    Vector3 targetPos, Rigidbody rb, float targetVel, float maxVel)
{
    var moveToPos = targetPos - rb.worldCenterOfMass;
    var velocityTarget = Time.fixedDeltaTime * targetVel * moveToPos;
    if (float.IsNaN(velocityTarget.x) == false)
    {
        rb.velocity = Vector3.MoveTowards(
            rb.velocity, velocityTarget, maxVel);
    }
}

/// <summary>
/// Перевірте, чи агент на землі, щоб увімкнути/вимкнути стрибки
/// </summary>
public bool CheckIfGrounded()
{
    hitGroundColliders = new Collider[3];
    var o = gameObject;

```

```

Physics.OverlapBoxNonAlloc(
    o.transform.localPosition + new Vector3(0, -0.05f, 0),
    new Vector3(0.95f / 2f, 0.5f, 0.95f / 2f),
    hitGroundColliders,
    o.transform.rotation);

var grounded = false;
foreach (var col in hitGroundColliders)
{
    if (col != null && col.transform != transform &&
        (col.CompareTag("walkableSurface") ||
         col.CompareTag("purpleGoal") ||
         col.CompareTag("blueGoal")))
    {
        grounded = true; //тоді ми заземлені
        break;
    }
}

return grounded;
}

/// <summary>
/// Викликається, коли агент стикається з м'ячем
/// </summary>
void OnCollisionEnter(Collision c)
{
    if (c.gameObject.CompareTag("ball"))
    {
        envController.UpdateLastHitter(teamId);
    }
}

/// <summary>
/// Починає послідовність стрибків
/// </summary>
public void Jump()
{
    jumpingTime = 0.2f;
    jumpStartingPos = agentRb.position;
}

/// <summary>

```

```

/// Вирішує рух агента
/// </summary>
public void MoveAgent(ActionSegment<int> act)
{
    var grounded = CheckIfGrounded();
    var dirToGo = Vector3.zero;
    var rotateDir = Vector3.zero;
    var dirToGoForwardAction = act[0];
    var rotateDirAction = act[1];
    var dirToGoSideAction = act[2];
    var jumpAction = act[3];

    if (dirToGoForwardAction == 1)
        dirToGo = (grounded ? 1f : 0.5f) * transform.forward * 1f;
    else if (dirToGoForwardAction == 2)
        dirToGo = (grounded ? 1f : 0.5f) * transform.forward * volleyballSettings.speedReductionFactor * -
1f;

    if (rotateDirAction == 1)
        rotateDir = transform.up * -1f;
    else if (rotateDirAction == 2)
        rotateDir = transform.up * 1f;

    if (dirToGoSideAction == 1)
        dirToGo = (grounded ? 1f : 0.5f) * transform.right * volleyballSettings.speedReductionFactor * -1f;
    else if (dirToGoSideAction == 2)
        dirToGo = (grounded ? 1f : 0.5f) * transform.right * volleyballSettings.speedReductionFactor;
    if (jumpAction == 1){
        if (((jumpingTime <= 0f) && grounded)){
            Jump();
        }
    }
    transform.Rotate(rotateDir, Time.fixedDeltaTime * 200f);
    agentRb.AddForce(agentRot * dirToGo * volleyballSettings.agentRunSpeed,
        ForceMode.VelocityChange);

    // змушує агента фізично «стрибати»
    if (jumpingTime > 0f)
    {
        jumpTargetPos =
            new Vector3(agentRb.position.x,

```

```

        jumpStartingPos.y + volleyballSettings.agentJumpHeight,
        agentRb.position.z) + agentRot * dirToGo;

        MoveTowards(jumpTargetPos, agentRb, volleyballSettings.agentJumpVelocity,
        volleyballSettings.agentJumpVelocityMaxChange);
    }

    // забезпечує спрямовану вниз силу для завершення стрибка
    if (!(jumpingTime > 0f) && !grounded)
    {
        agentRb.AddForce(
            Vector3.down * volleyballSettings.fallingForce, ForceMode.Acceleration);
    }

    // контролює послідовність стрибків
    if (jumpingTime > 0f)
    {
        jumpingTime -= Time.fixedDeltaTime;
    }
}

public override void OnActionReceived(ActionBuffers actionBuffers)
{
    MoveAgent(actionBuffers.DiscreteActions);
}

public override void CollectObservations(VectorSensor sensor)
{
    // Ротація агента (1 float)
    sensor.AddObservation(this.transform.rotation.y);

    // Вектор від агента до кулі (напрямок до кулі) (3 поплавки)
    Vector3 toBall = new Vector3((ballRb.transform.position.x - this.transform.position.x) * agentRot,
    (ballRb.transform.position.y - this.transform.position.y),
    (ballRb.transform.position.z - this.transform.position.z) * agentRot);

    sensor.AddObservation(toBall.normalized);

    // Відстань від м'яча (1 поплавок)
    sensor.AddObservation(toBall.magnitude);
}

```

```

// Швидкість агента (3 числа)
sensor.AddObservation(agentRb.velocity);

// Швидкість м'яча (3 поправки)
sensor.AddObservation(ballRb.velocity.y);
sensor.AddObservation(ballRb.velocity.z * agentRot);
sensor.AddObservation(ballRb.velocity.x * agentRot);
}

// Для людини-контролера
public override void Heuristic(in ActionBuffers actionsOut)
{
    var discreteActionsOut = actionsOut.DiscreteActions;
    if (Input.GetKey(KeyCode.D))
    {
        // обертати праворуч
        discreteActionsOut[1] = 2;
    }
    if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
    {
        // вперед
        discreteActionsOut[0] = 1;
    }
    if (Input.GetKey(KeyCode.A))
    {
        // повернути вліво
        discreteActionsOut[1] = 1;
    }
    if (Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow))
    {
        // назад
        discreteActionsOut[0] = 2;
    }
    if (Input.GetKey(KeyCode.LeftArrow))
    {
        // рухатися вліво
        discreteActionsOut[2] = 1;
    }
    if (Input.GetKey(KeyCode.RightArrow))
    {
        // рухатися праворуч

```

```
        discreteActionsOut[2] = 2;
    }
    discreteActionsOut[3] = Input.GetKey(KeyCode.Space) ? 1 : 0;
}
}
```

Додаток Д

```
public void ResolveEvent(Event triggerEvent)
{
    switch (triggerEvent)
    {
        case Event.HitOutOfBounds:
            if (lastHitter == Team.Blue)
            {
                // застосувати штраф до синього агента
            }
            else if (lastHitter == Team.Purple)
            {
                // застосувати штраф до фіолетового агента
            }

            // завершити епізод
            blueAgent.EndEpisode();
            purpleAgent.EndEpisode();
            ResetScene();
            break;

        case Event.HitBlueGoal:
            // перемagaє синій

            // зробити підлогу синьою
            StartCoroutine(GoalScoredSwapGroundMaterial(volleyballSettings.blueGoalMaterial, RenderersList, .5f));

            // завершити епізод
            blueAgent.EndEpisode();
            purpleAgent.EndEpisode();
            ResetScene();
            break;

        case Event.HitPurpleGoal:
            // фіолетовий перемagaє

            // зробити підлогу фіолетовим
            StartCoroutine(GoalScoredSwapGroundMaterial(volleyballSettings.purpleGoalMaterial, RenderersList, .5f));

            // завершити епізод
            blueAgent.EndEpisode();
            purpleAgent.EndEpisode();
            ResetScene();
            break;
    }
}
```

Рис. Д.1. Метод ResolveEvent ()

```

transform.Rotate(rotateDir, Time.fixedDeltaTime * 200f);
agentRb.AddForce(agentRot * dirToGo * volleyballsSettings.agentRunSpeed,
    ForceMode.VelocityChange);

// змушує агента фізично «стрибати»
if (jumpingTime > 0f)
{
    jumpTargetPos =
        new Vector3(agentRb.position.x,
            jumpStartingPos.y + volleyballsSettings.agentJumpHeight,
            agentRb.position.z) + agentRot * dirToGo;

    MoveTowards(jumpTargetPos, agentRb, volleyballsSettings.agentJumpVelocity,
        volleyballsSettings.agentJumpVelocityMaxChange);
}

// забезпечує спрямовану вниз силу для завершення стрибка
if (!(jumpingTime > 0f) && !grounded)
{
    agentRb.AddForce(
        Vector3.down * volleyballsSettings.fallingForce, ForceMode.Acceleration);
}

// контролює послідовність стрибків
if (jumpingTime > 0f)
{
    jumpingTime -= Time.fixedDeltaTime;
}

```

Рис. Д.2. Методи Rotate і AddForce та логіка керування поведінкою стрибка

```

// Для людини-контролера
public override void Heuristic(in ActionBuffers actionsOut)
{
    var discreteActionsOut = actionsOut.DiscreteActions;
    if (Input.GetKey(KeyCode.D))
    {
        // обернути праворуч
        discreteActionsOut[1] = 2;
    }
    if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
    {
        // вперед
        discreteActionsOut[0] = 1;
    }
    if (Input.GetKey(KeyCode.A))
    {
        // повернути вліво
        discreteActionsOut[1] = 1;
    }
    if (Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow))
    {
        // назад
        discreteActionsOut[0] = 2;
    }
    if (Input.GetKey(KeyCode.LeftArrow))
    {
        // рухатися вліво
        discreteActionsOut[2] = 1;
    }
    if (Input.GetKey(KeyCode.RightArrow))
    {
        // рухатися праворуч
        discreteActionsOut[2] = 2;
    }
    discreteActionsOut[3] = Input.GetKey(KeyCode.Space) ? 1 : 0;
}

```

Рис. Д.3. Метод Heuristic ()

```

public override void CollectObservations(VectorSensor sensor)
{
    // Ротація агента (1 float)
    sensor.AddObservation(this.transform.rotation.y);

    // Вектор від агента до кулі (напрямок до кулі) (3 поплавки)
    Vector3 toBall = new Vector3((ballRb.transform.position.x - this.transform.position.x) * agentRot,
    (ballRb.transform.position.y - this.transform.position.y),
    (ballRb.transform.position.z - this.transform.position.z) * agentRot);

    sensor.AddObservation(toBall.normalized);

    // Відстань від м'яча (1 поплавок)
    sensor.AddObservation(toBall.magnitude);

    // Швидкість агента (3 числа)
    sensor.AddObservation(agentRb.velocity);

    // Швидкість м'яча (3 поплавки)
    sensor.AddObservation(ballRb.velocity.y);
    sensor.AddObservation(ballRb.velocity.z * agentRot);
    sensor.AddObservation(ballRb.velocity.x * agentRot);
}

```

Рис. Д.4. Метод CollectObservations ()



```

Version information:
ml-agents: 0.30.0,
ml-agents-envs: 0.30.0,
Communicator API: 1.5.0,
PyTorch: 2.0.0+cu118
C:\Users\Skallifer\anaconda3\envs\mlagents_R20\lib\site-packages\torch\cuda\__init__.py:107: UserWarning: CUDA initializ
ation: The NVIDIA driver on your system is too old (found version 9010). Please update your GPU driver by downloading an
d installing a new version from the URL: http://www.nvidia.com/Download/index.aspx Alternatively, go to: https://pytorch
.org to install a PyTorch version that has been compiled with your version of the CUDA driver. (Triggered internally at
C:\actions-runner\_work\pytorch\pytorch\builder\windows\pytorch\c10\cuda\CUDAFunctions.cpp:109.)
  return torch._C._cuda_getDeviceCount() > 0
[INFO] Listening on port 5004. Start training by pressing the Play button in the Unity Editor.
[INFO] Connected to Unity environment with package version 2.3.0-exp.3 and communication version 1.5.0
[INFO] Connected new brain: Volleyball?team=0
[INFO] Hyperparameters for behavior name Volleyball:

```

Рис. Д.5. Запуск навчання