КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

на тему:

«Інтелектуальний програмний агент для пошуково-рятувальних робіт при ліквідації наслідків надзвичайних ситуацій»

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконав: студент 2 курсу групи ТШІ-21
Константинов А.В
(прізвище та ініціали)
Керівник: Красовська Г.В.
Кандидат технічних наук, доцент
(науковий ступінь, звання)

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань
Студент
(підпис)

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № ____ від «____» травня 2022 р.

Зав. кафедри _____ доц. Іларіонов О.Є.
(підпис)

Київ 2022

РЕФЕРАТ

Тема дипломної роботи “Інтелектуальний програмний агент для пошуково-рятувальних робіт при ліквідації наслідків надзвичайних ситуацій”.

Актуальність кваліфікаційної роботи полягає у необхідності автономного виконання пошуково-рятувальних завдань інтелектуальним агентом незалежно від складності наслідків надзвичайних ситуацій для збереження життя і здоров’я працівникам рятувальних служб.

Об’єктом дослідження роботи є процес пошуку цільового об’єкта інтелектуальним агентом у віртуальному середовищі.

Метою кваліфікаційної роботи є розробка моделі поведінки інтелектуального агента на основі нейронних мереж, що виконує пошуково-рятувальні задачі.

Предметом дослідження є методи та алгоритми навчання нейронної мережі для виконання інтелектуальним агентом пошуково-рятувальних завдань.

У роботі було розглянуто проблему порятунку людей при ліквідації наслідків надзвичайних ситуацій, можливі програмні та людські рішення у цих ситуаціях, релевантні алгоритми для пошуку цільових об’єктів у віртуальному середовищі, а також середовища для розробки та візуалізації роботи віртуальних агентів. Агента було розроблено у середовищі Unity, мова програмування та навчання C# та Python. Для навчання нейронної мережі було використано алгоритм Proximal Policy Optimization.

Ключові слова: нейронна мережа, Unity, інтелектуальний агент, віртуальне середовище, моделі навчання.

Розмір пояснювальної записки – 65 аркушів, містить 39 ілюстрацій, 1 таблиця, 2 додатка.

ABSTRACT

The theme of the thesis work is “Intellectual software agent for poke-repair robots in the elimination of innuendos of supernatural situations”.

The relevance of the qualifications of the work is influenced by the need for an autonomous viconnance of the search and repair services by an intelligent agent, regardless of the complexity of the epidemiological situations for saving life and health to the workers of the ritual services.

The object of follow-up work is the process of searching for a target object by an intelligent agent in a virtual environment.

The method of qualification work is the development of a model of the behavior of an intelligent agent based on neural networks, which is based on the search-and-reference tasks.

The subject of the research is the methods and algorithms for the formation of a neural array for the search by an intelligent agent of search-and-match tasks.

The problem of rescuing people from emergencies, possible software and human solutions in these situations, relevant algorithms for finding targets in a virtual environment, as well as an environment for developing and visualizing the work of virtual agents. The agent was developed in Unity, a C # and Python programming and learning languages. The Proximal Policy Optimization algorithm was used to train the neural network.

Keywords: neural network, Unity, intelligent agent, virtual environment, learning models.

The size of the explanatory note is 65 sheets, contains 39 illustrations, 1 table, 2 appendices.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД МУЛЬТИАГЕНТНОГО ПІДХОДУ ДО РОЗВ'ЯЗАННЯ ПОШУКОВО-РЯТУВАЛЬНИХ ЗАДАЧ ПРИ ЛІКВІДАЦІЇ НАСЛІДКІВ НАДЗВИЧАЙНИХ СИТУАЦІЙ 8

1.1 Основні визначення мультиагентного підходу 8

1.2 Властивості інтелектуальних агентів 9

1.3 Архітектура інтелектуальних мультиагентних систем 12

1.4 Моделі поведінки агентів у мультиагентній системі 14

1.5 Роль інтелектуальних агентів при ліквідації наслідків надзвичайних ситуацій 15

1.6 Постановка задачі пошуку інтелектуальним агентом об'єкта-цілі 19

1.7 Функціональні та системні вимоги до інтелектуального агента 20

Висновки до розділу 1 21

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ ТА МЕТОДІВ НАВЧАННЯ ІНТЕЛЕКТУАЛЬНИМ АГЕНТІВ 22

2.1 Огляд моделей навчання інтелектуальних агентів 22

2.2 Моделі поведінки агентів з використанням нейронних мереж. 30

2.3 Навчання нейронної мережі 31

2.4 Генетичний алгоритм 32

2.5 Q-алгоритм. ц38

2.6 Алгоритм Proximal Policy Optimization 41

2.7 Необхідні компоненти для навчання нейронної мережі 42

2.8 Середовища розробки агентів 44

Висновки до 2 розділу 46

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ 46

3.1 Базовий опис інтелектуального агента у Unity 47

3.2 Налаштування віртуального середовища Unity для навчання інтелектуального агента. 48

3.3 Налаштування модулів навчання 52

3.4 Навчання інтелектуального агенту 56

3.4.1 Вхідні дані для нейронної мережі 56

3.4.2 Початок навчання інтелектуального агента 57

3.5 Результати навчання інтелектуального агента. 60

3.5.1 Аналіз точності моделі 60

ВИСНОВОК 63

ВИКОРИСТАНІ ДЖЕРЕЛА 65

ДОДАТКИ 67

ВСТУП

Інтелектуальні системи є одним з швидко розвиваючихся напрямків, що формувалися на основі досліджень у галузі розподілених комп'ютерних систем, паралельних обчислень та базуючись на розвитку штучного інтелекту. Мультиагентні технології сформовані на принципах автономності певних складових програми або агентів, які можуть організовано та сумісно функціонувати у розподіленій системі. Агентом в даному випадку називається автономний штучний інтелект, або комп'ютерна програма, яка має свою поведінку та здатність взаємодіяти з іншими об'єктами у динамічному середовищі. Кожен агент може бути налаштований під абсолютно різні задачі, від отримання повідомлень з системи та їх перетворень на зрозумілий зміст, так і бути запрограмованим під певні задачі, такі як пошук найкоротших шляхів тощо. І в тому і в іншому випадку може використовуватись штучний інтелект.

Сьогодні агентний підхід має широкий спектр застосування у різних сферах, а саме у розподіленого вирішення складних завдань, реінженірг підприємств, пошуково-рятувальних сферах, банківському та електронному бізнесі тощо. В даній дипломній роботі розглядає саме вирішення інтелектуальним агентом задач пошуково-рятувального характеру.

Наш світ є дуже непередбачуваним, трапляються війни, землетруси, звичайні завали будівель тощо. Не секрет, що стихійні лиха є одним із найпоширенішими у світі, що спричиняють руйнацію будівель. Крім того, не можна виключати такого роду надзвичайні ситуації спричинені сучасними війнами, після яких у завалах можуть залишатись снаряди, які не здетонували, міни, пастки тощо. На місці працюють спеціальні служби з надзвичайних ситуацій разом з піротехніками, які саме намагаються вирішити наслідки таких подій. При цьому люди являються головним рушієм для пошуку людей

під завалами будівель та розмінуванням, й іноді ціною власного життя виконують пошуково-рятувальні задачі.

Агентний підхід в даному випадку може зберегти життя людей, які мають за місію знаходження людей під уламками великих та малих будівель, в розумінні чи потрібно розбирати завали тощо. Виходячи з того, що робота інтелектуального агента або робота може зберегти життя рятувальникам, ця тема є актуальною.

Метою даної кваліфікаційної роботи є розробка інтелектуального агенту, який за допомогою нейронної мережі виконує пошук об’єктів у певному віртуальному середовищі.

Об’єктом дослідження роботи є процес пошуку цільового об”єкта інтелектуальним агентом у віртуальному середовищі.

Предметом дослідження є методи та алгоритми навчання нейронної мережі для виконання інтелектуальним агентом пошуково-рятувальних завдань.

Задачі випускної кваліфікаційної роботи:

- провести аналіз стратегій пошуку інтелектуальним агентом шляху до заданого об'єкта - цілі;
- провести аналіз алгоритмів пошуку найкращого шляху за заданими критеріями;
- провести аналіз алгоритмів навчання нейронної мережі для розв`язку задачі пошуку найкращого шляху;
- провести вибір віртуального середовища для візуального відображення та проведення експериментальних досліджень;
- провести дослідження ефективності роботи обраного алгоритму шляхом емуляції роботи інтелектуального пошукового агента у віртуальному середовищі.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД МУЛЬТИАГЕНТНОГО ПІДХОДУ ДО РОЗВ'ЯЗАННЯ ПОШУКОВО-РЯТУВАЛЬНИХ ЗАДАЧ ПРИ ЛІКВІДАЦІЇ НАСЛІДКІВ НАДЗВИЧАЙНИХ СИТУАЦІЙ

1.1 Основні визначення мультиагентного підходу

Мультиагентний напрям штучного інтелекту є справжнім технологічним досягненням, що виник дуже швидко й паралельно у різних сферах життя таких як охорона здоров’я, промислове виробництво, онлайн бізнес тощо. Говорячи наприклад про європейські країни можна сказати, що технології штучного інтелекту використовуються як звичайними людьми, так і самими виробниками[1].

Португальський вчений Паоло Летальо у своїх роботах зазначає, що штучний інтелект у мультиагентних системах надає рішення для кількох складних проблем в області машинобудування і інформатики, а саме:

1. Оптимізація логістики та виробничих процесів.

2.Розпізнавання образів, наприклад виявлення тенденцій і закономірностей в медичній або виробничій діагностиках.

3. Комп'ютерний зір, наприклад навігація автономних мобільних роботів і аналіз медичних зображень.

4. Розпізнавання мови, наприклад, підтримка людино-машинних інтерфейсів.

5. Інтелектуальне управління, наприклад забезпечення адаптивної та інтелектуальної поведінки для управління процесами.

Поняття агента відповідає апаратно-програмно реалізований сутності, що має поставлену мету та намагається знайти шлях до досягнення цієї мети. Якщо брати до уваги багатоагентні системи, то вони мають багато автономних працюючих агентів, які взаємодіючи між собою та з користувачем можуть досягати поставлених цілей.

Приклади таких завдань: управління інформацією, потоками та мережею, управління рухом повітряних суден, задачі пошуку інформації у мережі, електронні бібліотеки, задачі електронної комерції, в освіті тощо.

1.2 Властивості інтелектуальних агентів.

Основні властивості інтелектуальних агентів[2]:

- робота агента без втручання розробника;
- активність;
- автономність;
- вміння організовувати та здійснювати дії;
- комунікабельність;
- взаємодія та спілкування з іншими агентами;
- реактивність;
- адекватне сприйняття стану довкілля та реакція з його зміна.
- цілеспрямованість;
- припущення про наявність власних джерел мотивації;
- мати базові знання про себе, інших агентів та навколишнього середовища;
- переконання є мінливу частину базових знань, що змінюється з часом;
- бажання;
- прагнення певним станам;
- наміри;
- заплановані дії;
- зобов'язання;
- завдання, що виконуються одним агентом на запит та/або від імені інших агентів.

Характеристики агентів також можуть бути класифіковані за рівнем знань агентів (табл. 1.1).

Таблиця 1.1 – Характеристики агентів у мультиагентних системах

Характеристика	Тип			
	звичайний	розуміючий	розумний	інтелектуальний
Автономія	Так		Так	Так
Взаємодія з агентами в системі	Так	Так	Так	Так
Реактивність	Так	Так	Так	Так
Адаптивність		Так	Так	Так
Набуття знань з навколишнього середовища			Так	Так
Адекватна реакція на помилку			Так	

До цього списку іноді додають інші якості, зокрема:

- правдивість;
- неможливість заміни правдивої інформації явно неправдивою;
- дружелюбність;
- готовність співпрацювати з іншими агентами у процесі вирішення своїх завдань;

- пріоритет спільних цілей над особистими;
- мобільність;
- можливість агента мігрувати через мережу в пошуках потрібної інформації.

Крім того, виділяють когнітивних або розумних агентів, та реактивних. У розумних агентів є вбудовані моделі міркування, аналізу дій, а також, зазвичай, доданою базою даних. На рис.1.1 можна побачити структуру інтелектуального агента[2].

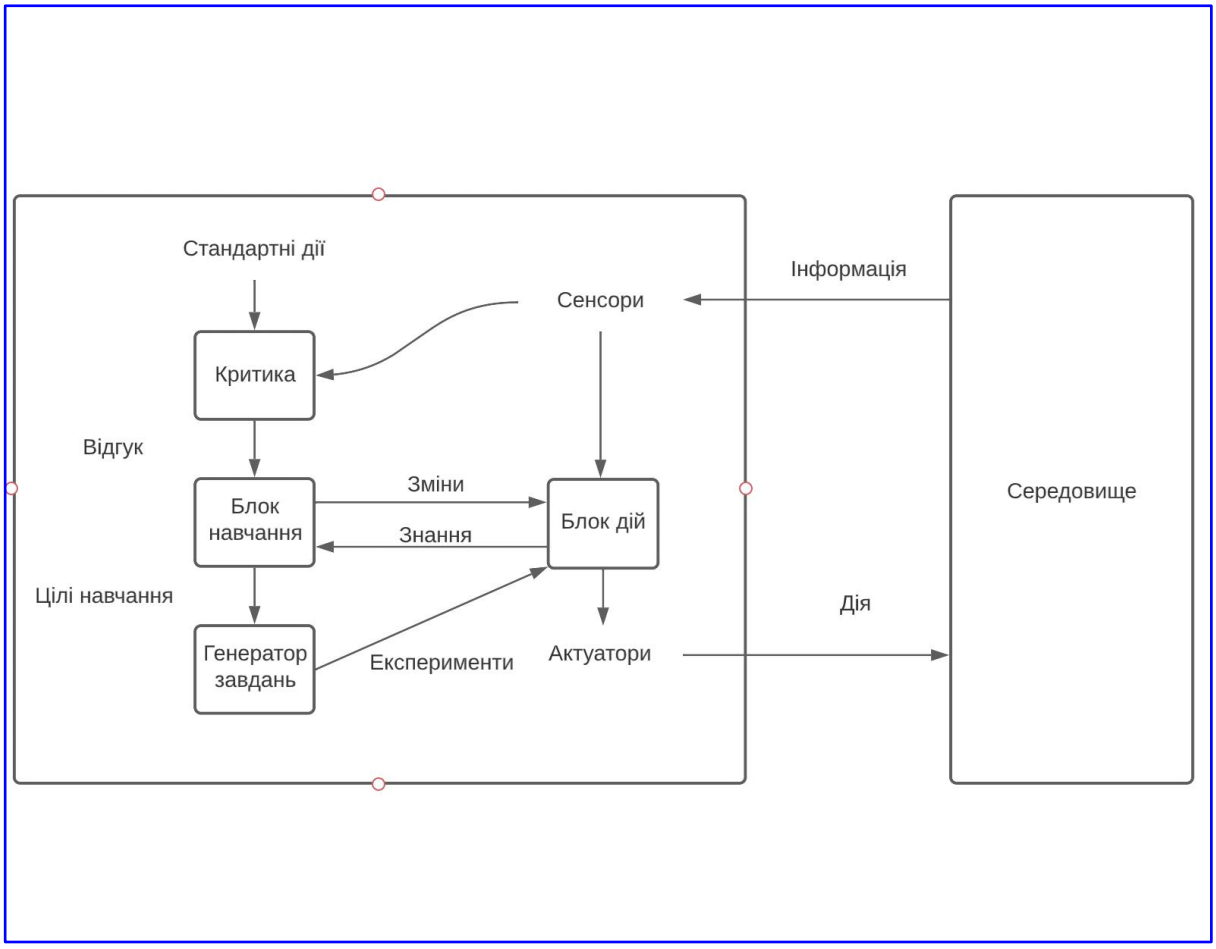

Рисунок 1.1 – Структура інтелектуального агента [2]

Говорячи про реактивних агентів, вони не мають такого ж рівня розуміння зовнішнього середовища, як у інтелектуальних, тому вони не можуть мати та розпоряджатися ресурсами. Їх поведінка зазвичай визначена лиш

простою метою, на яку вони повинні формувати свої відповіді. Планувати свої дії вони не здатні, а тому повністю планують свої дії самостійно.

1.3 Архітектура інтелектуальних мультиагентних систем

Інтелектуальна багатоагентна система може складатися с набору агентів, які розподіляються по середовищу та можуть мігрувати в пошуках потрібних даних, знань, процедур для досягнення поставлених цілей.

Виділяють три основні типи архітектури мультиагентних систем залежно від концепцій розробки [6]:

1) архітектури, що засновуються на роботі зі знаннями;

2) архітектури, що засновуються на роботі з патернами поведінки;

3) гібридні архітектури.

У першому типі архітектур використовуються традиційні моделі, методи та інструменти штучного інтелекту для представлення та обробки знань, а також прийняття рішення приймаються з урахуванням формальних механізмів міркування. У перших системах цього знання використовувалися логіки предикатів першого порядку для уявлення та обробки знань. Подальші дослідження у даному напрямку призвело до логічних обчислень, які орієнтовані на характеристики агентів, таких як: бажання, наміри, обов’язок. Основним недоліком першого типу архітектури є складність побудови повних баз знань, необхідних для створення систем. Зокрема, розумний агент може мати архітектуру типової системи, здатної сприймати інформації з зовнішнього середовища та виконувати певні дії в результат обробки цієї інформації. Основні відмінності програми з агентним підходом та звичайної є те, що у програмі з агентним підходом наявні спеціальні модулі комунікації та вбудовані механізми розуміння цілей. Така архітектура дозволяє розмірковувати, але не досліджувати. Базуючись на системах класифікації Дж.

Холланда, можна реалізувати адаптивну поведінку агента. Найбільш важливі відмінності від звичайних неагентних систем:

1) можливість формування нових правил за допомогою генетичних алгоритм;

2) наявність механізму винагороди.

Реактивні типи архітектур не здатні використовувати традиційні моделі символьного уявлення знань, тому представляють модель поведінки агента зазвичай простим набором правил, що дозволяють агенту обрати конкретні дії під певну ситуацію. Системи такого типу зазвичай мають високий ступінь спеціалізації та жорсткі обмеження за складністю Завдання.

Найбільш перспективними є гібридні інтелектуальні багатоагентні системи, що дозволяють використовувати можливості інтелектуали та реактивні архітектори. Архітектура з ієрархічною базою знань містить структуровану певним чином базу знань, що має модуль управління, виділену пам'ять, зв'язком та людино-машинний інтерфейс [3]. Така архітектура агента здатна розмірковувати та реагувати на навколишнє середовище. Його база знань містить три рівні:

1) предметна галузь;

2) розуміння того, як діяти в умовах невизначеність;

3) контроль знань.

Головною особливістю розумного агента є перш за все здатність приймати рішення, а реактивна (проста) система працює за принципом глобальної дошки оголошень, де агент взаємодіє з користувачем використовуючи спеціальний інтерфейс.

Одним із нових напрямків є використання нейронних мереж, які дозволяють агентам самостійно навчатися та використовувати знання, що потрапляють в систему в процесі навчання.

1.4 Моделі поведінки агентів у мультиагентній системі

Є такі моделі координації поведінки агентів: моделі планування колективної поведінки, теоретико-ігрові моделі, моделі колективної поведінки автоматів, моделі координації поведінки на основі конкуренції, моделі на основі архітектури BDI[6].

Розглядаючи ігрову модель можна сказати, що в теорії ігор завданням є вибір найкращих розв’язків в умовах невизначеності або в умовах конфлікту.

Можливі ситуації вибору поведінки агента та його дій можна класифікувати так:

1) Симетрична кооперація;

2) Симетричний компроміс;

3) Несиметрична кооперація;

4) Конфлікт.

Моделі колективної поведінки автоматів засновані на ідеях повної розподіленості та самоорганізації. Даний тип поведінки може підходити для побудови структури переговорів у завданнях, яким характерно взаємодіяти з великою кількістю характеристик у системі.

Модель планування колективної поведінки може бути розподіленою, централізованою або частково централізованою. В кінцевому випадку агенти самостійно будуть приймати рішення відносно наступних дій.

Також є моделі на основі BDI-архітектур, що застосовують логічної парадигми штучного інтелекту та аксіоматичні методи теорії ігор. Опис здійснюється за рахунок логічних засобів.

Моделі на основі конкуренції використовують таке поняття як аукціон, який виступає повноцінним механізмом координації поведінки агентів.

1.5 Роль інтелектуальних агентів при ліквідації наслідків надзвичайних ситуацій

Екстремальні та надзвичайні ситуації, нажаль, все частіше зустрічаються людьми у світі з різних причин. Найпоширенішими на даний момент є війни та стихійні природні явища, що періодично відбуваються в різних куточках нашої планеті. Наприклад, за даними німецької компанії Munich Re збитки від стихійних лих у 2021 році складають приблизно 280 мільярдів доларів[12]. Страхова компанії зазначає, що близько 145 мільярдів доларів збитків припадає саме на Сполучені Штати Америки, де люди страждають від ураганів, повеней, торнадо та аномальних температур[12]. Десятки тисяч будівель були зруйнованими, тисячі постраждалих. Служби з надзвичайних ситуацій займаються ліквідацією наслідків таких явищ та порятунком людей, що могли бути під завалами. Проблема даного виду ліквідації наслідків в тому, що саме люди та людський фактор є вирішальним, й будь-яка дія може загрожувати життю.

Щоб зберегти життя людей зі спец. служб можуть бути використані інтелектуальні агенти, програмні модулі яких можуть бути імплементовані у пошукових роботів. Головною перевагою в даних роботах є застосування нейронної мережі, яка може не тільки шукати найкращий шлях до об’єктів під завалами, а також збирати дані та прораховувати ризики, оцінювати стан завалів, підраховувати кількість людей під завалами та їх точне місцезнаходження тощо.

Перш за все необхідно розглянути питання як наземні роботи зазвичай використовуються для допомоги рятувальним при ліквідації наслідків надзвичайних ситуацій?

Нинішній стан практики пошуку всередині завалів полягає в тому, щоб використовувати або невеликий гусеничний транспортний засіб, такий як Inkutun VGTV Extreme, який є найбільш часто використовуваним роботом для

таких ситуацій, або змієподібним роботом, таким як Active Scope Camera. розроблений [15].

Наземні роботи, як правило, використовуються для того, щоб зайти в місця, куди пошукачі не можуть поміститися, і пройти далі, ніж можуть пошукові камери. Пошукові камери зазвичай досягають максимуму на 18 футів, тоді як наземні роботи можуть занурюватися на глибину понад 60 футів у уламки. Вони також використовуються для того, щоб зайти в небезпечні порожнечі, куди міг би поміститися рятувальник, але це було б небезпечно і, отже, вимагало б годин роботи, щоб підтримати, перш ніж хтось зможе безпечно увійти в неї.

Теоретично, наземних роботів (рис. 1.2) можна було б також використовувати, щоб медичний персонал міг бачити та розмовляти з людьми, які опинились під завалами, а також носити до них невеликі пакети з водою та ліками[16].

Рисунок 1.2 - Наземний пошуково-рятувальний робот.

Одна зі значних проблем для наземних роботів це бачити руїни. Якщо вдасться загнати робота в уламки, то інженери-конструктори зможуть побачити

внутрішню частину зруйнованої будівлі, щоб зрозуміти як саме проводити рятувальні операції з урахуванням збереження життя та здоров’я [16].

Потрапити всередину руйнувань справді важко, бо масштаб уламкыв може бути дуже великим. Якщо порожнечі мають порядок розміру робота, це складно. Якщо щось піде не так, воно не може повернутися; він повинен їхати назад.

Крім того, важливо відмітити звивистість роботів при виконанні задач, тому для пошуку постраждалих у надзвичайних ситуаціях було розроблено вченими Стенфордського університету змієподібного робота, який може рости як ліана, протискаючись між завалами. Більше того, робот має можливість доставляти води постраждалим через свої канали та має маленьку портативну камеру[16] (рис 1.3).

Рисунок 1.3 - Змієподібний робот.

Щоб протестувати свою розробку, команда зі Стенфорда використала низку перешкод, і робот успішно подолав липучки, клей та цвяхи, та навіть крижану стіну (рис. 1.4).

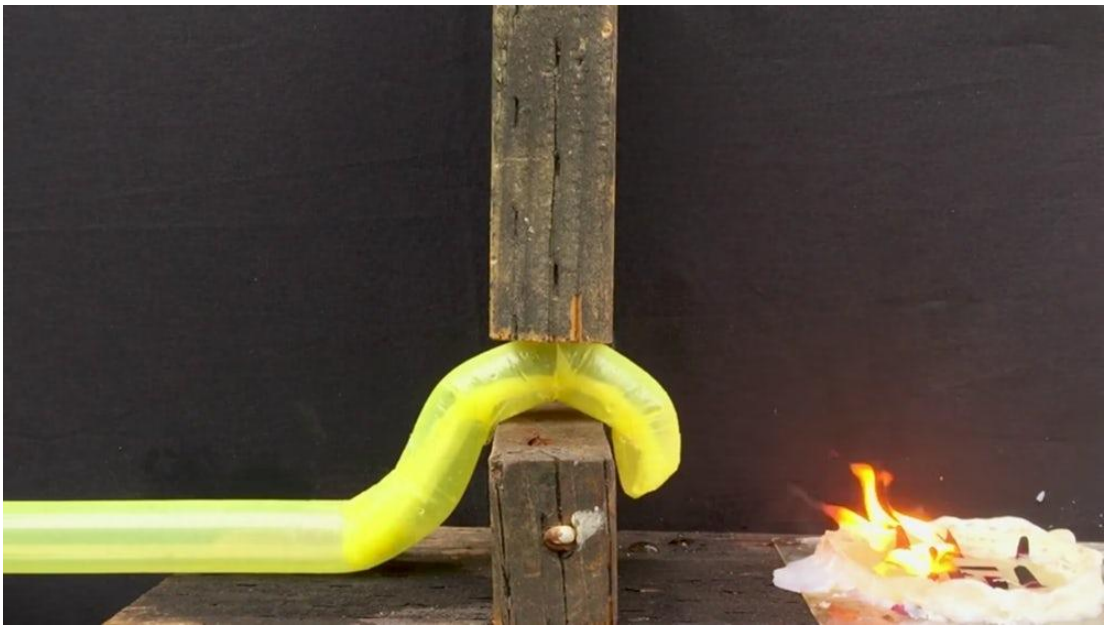

Рисунок 1.4 - Проходження перешкод змієподібним роботом [17].

Як і будь-якому роботу, йому не вдалося уникнути пошкоджень, але завдяки унікальному методу пересування проколи не сильно впливають на функціональність робота.

1.6 Постановка задачі пошуку інтелектуальним агентом об'єкта-цілі

Кожен агент характеризується своїм станом S відносно середовища й має чіткі координати початку роботи (x, y, z) та вектор направлення (v):

$$S = < x, y, z, v > | \tag{1.1}$$

Задача агента в середовищі - навчитись досягати певних об’єктів за найкоротший час.

В результаті отримується узагальнений показник успішності досягнення цілі W (показник ефективності), використовуючи дані про винагороди агента Wi: чим більше винагорода - тим точніше модель.

$$W = \sum_{i=1}^{n} Wi;\ Wi \geq 1.0 \quad (1.2)$$

Модуль управління агента має складатися з декількох важливих блоків:

- блок дій агента;
- блок оцінки діяльності агента.

Блок дій агента має включати з нейрону мережу та алгоритм, що визначає поведінку агента.

Блок оцінки діяльності агента показує покращився чи погіршився показник успішності досягнення цілі порівняно з попередніми результатами.

1.7 Функціональні та системні вимоги до інтелектуального агента

До функціональних вимог відноситься [4]:

- визначення агентом найкоротшого шляху до цільових об’єктів середовища;
- визначати інтелектуальним агентом межі свого впливу на середовище;
- фіксація факту досягнення цілі;
- визначення типу та розміру винагороди за виконані дії.

Для візуалізації роботи інтелектуального агента та дослідження її ефективності необхідно обрати віртуальне середовище, що може відображати агента та тривимірні об’єкти, які будуть ціллю. Дане середовище повинне бути замкнутим, тобто агент має чіткі рамки повноважень для пошукових дій.

До нефункціональних вимог можна віднести[4]:

- доступ користувача до віртуального середовища, де агент веде діяльність;
- підтримка OS Windows 7, Windows 8, Windows 10, MacOS;

- наявність пакетів Python, PyTorch, Pip3, Numpy, ML Agents, що необхідні для запуску навчання нейронної мережі та візуалізації роботи у певному віртуальному середовищі.

До системних вимог при роботі з віртуальним середовищем можна віднести:

- процесор із тактовою частотою не менше ніж 1500 МГц;
- розмір оперативної пам’яті мінімум 2 ГБ;
- розмір відеопам’яті не менше 1 ГБ;
- із зовнішніх пристроїв монітор, клавіатура, миша.

Висновки до розділу 1

Використання інтелектуальних агентів при вирішенні пошуково-рятувальних завдань є надзвичайно актуальним на сьогоднішній день. Правильно навчений інтелектуальний агент може не тільки рятувати життя жертвам надзвичайних ситуацій, убезпечувати працівників служб спасіння, а ще й збирати оперативні дані для проведення подальшого аналізу.

У першому розділі було розглянуто:

- поняття інтелектуального агенту;
- основні проблеми, які вирішує інтелектуальний агент;
- моделі, на основі яких можуть бути побудовані програмні агенти;
- роль розумних програмних агентів при ліквідації наслідків надзвичайних ситуацій;
- розроблено постановку задачі, функціональні та нефункціональні вимоги, системні вимоги.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ ТА МЕТОДІВ НАВЧАННЯ ІНТЕЛЕКТУАЛЬНИМ АГЕНТІВ

2.1 Огляд моделей навчання інтелектуальних агентів

Для розуміння поведінки агентів її можна декомпозувати на ряди або рівні, які виконуються послідовно (рис. 2.1). В даному випадка ієрархія рівнів складається з шару вибору дій, шару прицілювання та пересування. Важливо зазначити, що зазначений розділ на рівні має місце тільки для стратегій руху. Якщо є потреба створити чат-бот, який має за мету дачу певних відповідей, необхідна інша стратегія.

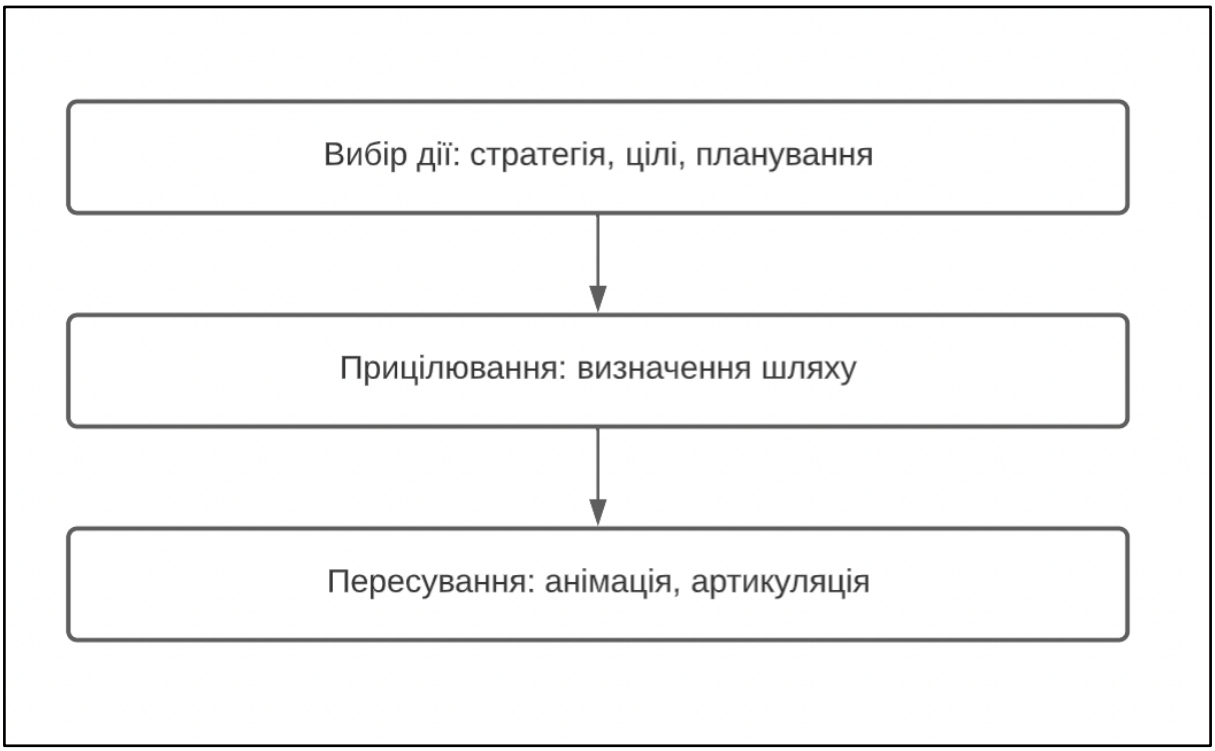

Рисунок 2.1 – Ієрархія стратегій руху з трьох шарів

У більшості інтелектуальних агентів розробка агентів починається з шару вибору дій та визначення стратегій, цілей та планування дій інтелектуального агента в середовищі.

Зафіксувавши дані першого шару, агент у своїх розрахунках має перейти до визначення найкращого та найкоротшого шляху, який виконується на другому шарі.

Третій шар, пересування, є фізичним втіленням агента. В даному випадку цей шар отримує сигнали із попереднього рівня, рівня прицілювання про те, яке завдання перед агентом зафіксовано.

Важливо відмітити, що стратегії прицілювання можуть стати повністю незалежними від особливостей схем руху завдяки сигналам з рівня стратегії.

Агент може бути представлений у вигляді динамічної моделі об'єкта, що може рухатись та взаємодіяти з іншими агентами. В даному випадку використовується звичайна модель руху і адаптивну модель анімації. Така комбінація може бути використана для зв'язування абстрактної моделі переписування і будь-якої конкретної основи, де власне буде рухатись об’єкт.

Модель руху агента базується примітивному методі пересування. Слово пересування може бути примінено до всіх рухомих об'єктів у середовищі, від гужового транспорту, автомобілів, до літаків, дронів тощо.

Розміри об’єкта тут не враховується, у просторі дані об’єкти відображені лиш матеріальними точками. Таким чином можна отримати спрощену фізичну модель. Важливою відміткою тут є те, що будь-який фізичний об’єкт, який має масу, повинен мати не нульовий показник радіуса з певною інерцією у просторі. Будь-який фізичний об'єкт з масою повинен мати не нульовий радіус разом з зафіксованим моментом інерції.

Використання спрощеної моделі в даному випадку обумовлено перш за все зручністю і задачею збереження загальних міркувань.

Матеріальні точки у просторі визначається його координатами й масою. Крім того, примітивні моделі містять у своїй конфігурації параметр швидкості руху та обмеження відносно напрямків. У зв'язку з тим, що ми маємо справу із засобами для пересування агентів, ці сили в загальному сенсі можуть впливати на себе й обов’язково мають бути обмежені.

Для опису примітивної моделі доречно бути використати узагальнене поняття «максимальна сила». Це обмеження виникає в результаті взаємодії прискорення і гальмування.

Більше того, засоби для пересування також характеризуються параметрами орієнтації, взятим з поточного положення об'єкта, вирівняного за системою координат у середовищі, до якої пов'язана геометрична модель транспортного засобу. Примітивна модель агента транспортного засобу:

Масового	(вага – скалярний)
Позиція	(позиція - вектор)
Швидкість	(швидкість – вектор)
Максимальна швидкість	(max_speed – скалярний)
Максимальні зусилля	(max_force – скалярний)
Орієнтації	(N основних векторів)

Тривимірна модель складається з трьох компонентів, а значення параметра орієнтації – з трьох векторів (матриці 3x3).

Основа для примітивної моделі агента заснована на методі Ейлера. На кожному етапі моделювання сили націлювання, що застосовуються до матеріальної точки, визначаються алгоритмами поведінки і обмежуються параметром max_speed max_force.

Також для розрахунку моделі потрібно додати значення швидкості до попередньої позиції об'єкта.

Steering force = truncate (steering direction, max_force);

Acceleration = steering force / mass;

Velocity = truncate (velocity + acceleration, max_speed);

Position = position + velocity

Примітивна модель агента підтримує певне вирівнювання швидкості у заданому просторі, використовуючи динамічні параметри.

Для визначення об’єкта у локальній системі координат, використовуються такі вектори як вектор опису об’єктів у просторі та вектора початку координат.

В загальному вектори повинні визначати напрямок і довжину кроків у системі координат, для кожного з трьох взаємно перпендикулярних напрямків осей. Для простоти розуміння дані осі визначають як: X – пряма, Y – верхня, Z – бічна.

Для підтримки вирівнювання швидкості на всіх етапах моделювання, вектори повинні постійно перенаправляти у новому напрямку. У випадку, якщо вектор швидкості буде дорівнювати нулю, стару орієнтацію необхідно зберегти, але варто враховувати, що простір буде перебудовуватися кожен раз за допомогою наближення та заміни. Для цього використовуються спеціальні вирази векторного добутку для знаходження потрібних базисних векторів.

New_forward = normalize (velocity);

Approximate_up = normalize (approx);

New_side = cross (forward, approx);

New_up = cross(forward, side);

Суть даних виразів у наближенні напрямку *Y* з найближчими перпендикулярними до цільового напрямку *X*. Це можна примінити тому, що орієнтація змінюється від кадру до кадру незначними кроками. Виходячи з цього, вони є перпендикулярні один одному.

Поняття «вирівнювання швидкості» визначається не тільки параметром орієнтації.

Концепція «вирівнювання по швидкості» не може визначатися лише одним параметром орієнтації. В даному випадку необхідно розглянути степінь обмежень для обертання навколо осі X. Створюючи новий простір ми гарантуємо, що величина обертання об'єкта із початку створення моделей має залишатися постійною. Щоб визначити величину обертання потрібно виконати

додаткові обчислення, що застосовуються на конкретній моделі засобу пересування.

Якщо брати до уваги транспорті кораблі як літаки, човни, космічні кораблі тощо, є сенс додати до моделі параметри обертання такі як крен корпуса. Так як в такого роду завдань присутня гравітація, нижній напрямок базису потрібно вирівнювати відносно сум прискорення повороту разом з гравітаційним прискоренням. Виходить, що для реалізації ідеї крену необхідно апроксимувати вісь Y сумою прискорень до цільового об’єкту.

У моделі «автомобіль», що рухається по деякій поверхні у певному середовищі, необхідно розглядати як її позицію на цій поверхні, так і вирівнювання осі Y відносно площини. Причому швидкість рухомого об'єкта повинна бути направлена по дотичній до нашої поверхні. Кожна точка в просторі порівнюється з точками поверхні, які знаходяться в межах середовища системи координат. У примітивній моделі «автомобіль» сигнали, надходять від рівня прицілювання у рівень пересування, описані тільки одним вектором. Якщо розглядати більш реалістичні моделі пересування, то набір сигналів буде більш складним. До прикладу можна віднести автомобіль, що має рульове колесо, педалі газу та гальмів. Ці величини будуть представлені як скаляри, тому стає можливим примінення сили, яка націлює вектор на три компоненти системи. Вісь Z стає величиною прицілювання, вісь X – стає прискоренням, при умові, що величина додатна, а також сигналами гальмування, якщо величина невід’ємна.

Відображення по осі X можуть бути не симетричними в загальному випадку. Наприклад, середньостатистичний автомобіль може гальмувати тільки тоді, коли сила гальмування буде сильніше за силу тяги його двигуна (рис.2.2).

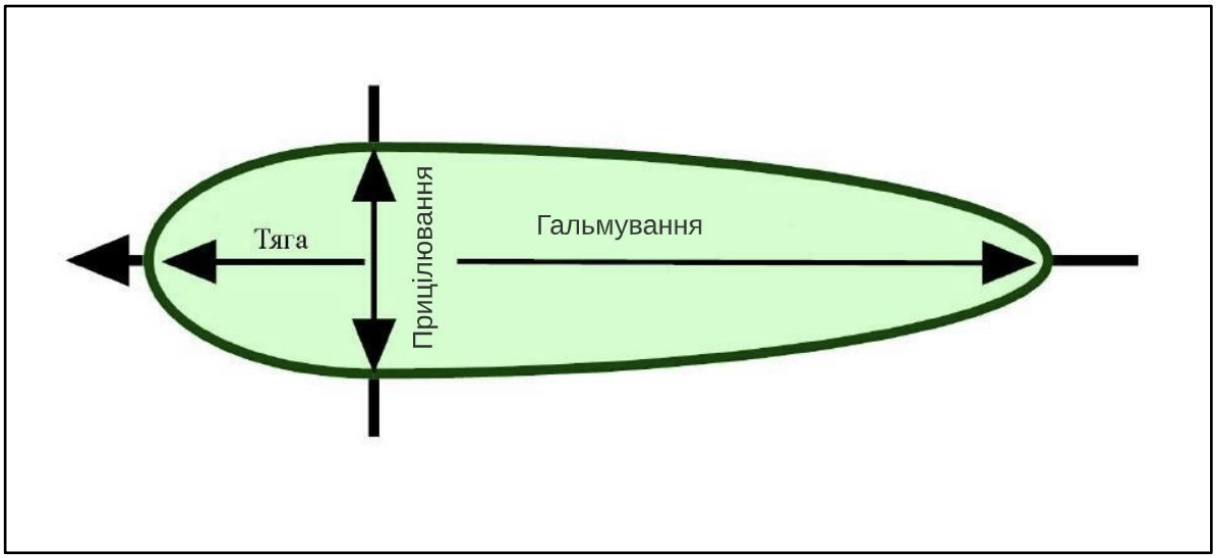

Рисунок 2.2 – Модель гальмування засобу пересування

Розглянемо різні стратегії поведінки примітивної моделі агентів (матеріальної точки) [7].

Стратегії прицілювання. Для матеріальної точки дана стратегія означена в переліку термінів геометричних обчислень над векторами. Головним завданням стратегії прицілювання є визначення шляху інтелектуального агента, що напряму залежить від цілей агента, тобто це може бути ціль знайти певні об’єкти або знайти вихід.

Стратегія «Пошук». Базова стратегія пошуку перш за все націлює агента на конкретну точку у просторі. Особливість у тому, що вектори швидкості та напрямку повинні бути направлені у бік мети. Вектор прицілювання в даному випадку це різниця між цільовою швидкістю і наявною швидкістю агента (рис.2.3.):

Desired_velocity = normalize (position - target) * max_speed;

Steereing = desired_velocity – velocity.

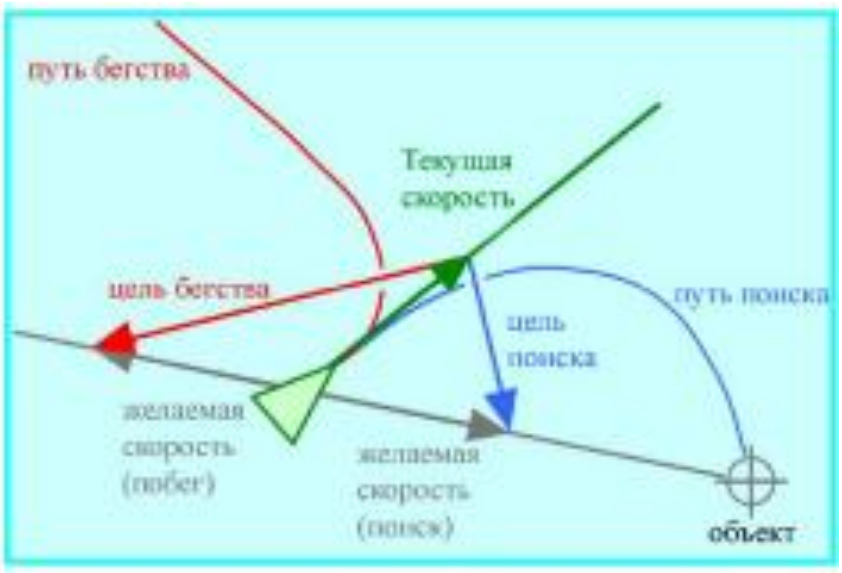

Рисунок 2.3 – Стратегія «пошук»

В ситуації, коли агент досягнув ціль й не зупинив пошук, він повертається на початкову точку й починає пошук наново.

Стратегія «Блукання». Дана стратегія має ознаку випадкового прицілювання. Найпростішим способом реалізаціє є додавання випадково сили прицілювання на кожному кроці роботи моделі, але такий рух буде нестійким. Крім того, зберігання напрямку прицілювання є також одним з цікавих підходів. На кожному кроці достатньо робити невеликі відступи від напрямку прицілювання. Таким чином агент навчається ходити приблизно в одному напрямку. Ідея може бути реалізована різними методами, але безумовно деякі з методів в кінцевому випадку згенерують значно кращі результати, порівнюючи з іншими. Дана стратегія может бути застосована, якщо зафіксувати силу прицілювання на поверхні сфери, яка має бути спроектована на площинах X та Z. Додавши випадковий зсув до попередньо зазначеного значення сили, розробник може отримати силу кроку. Сума обов’язково повинна бути обмежена поверхнею встановленої сфери. Радіус сфери (велике коло на рис.2.4) визначає максимальний розмір значення сили «блукання», а значення зсуву виражається величиною «блукання».

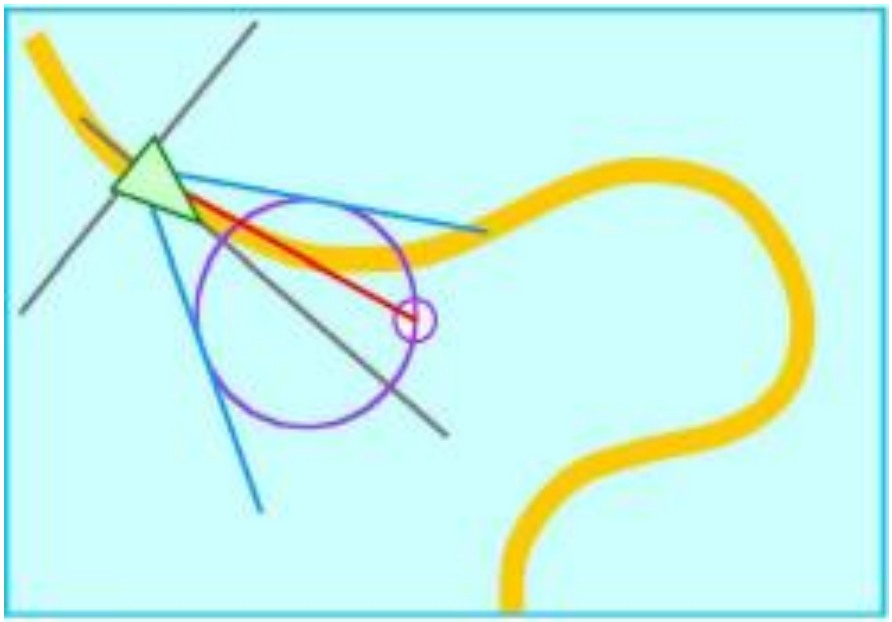

Рисунок 2.4 – Стратегія «блукання»

Стратегія «Проходження по шляху». Поведінка «Проходження по шляху» дозволяє інтелектуальному агентову пересуватись уздовж заздалегідь визначеного шляху, наприклад шосе, тунелі, стежки. Рух, відповідно до стратегії «проходження по шляху», має подібність до на пересування людей по коридору: агент зазвичай проходить паралельно зазначеній центральній лінії коридору, але в цілому об'єкт руху має можливість відхилитися від лінії. При цьому агент повинен залишатися всередині встановленого розробником «коридору». Якщо агент рухається у напрямку виходу з коридору, потім він повинен повернутись на заданих шлях, а лиш наступним кроком може бути продовження виконання своїх завдань.

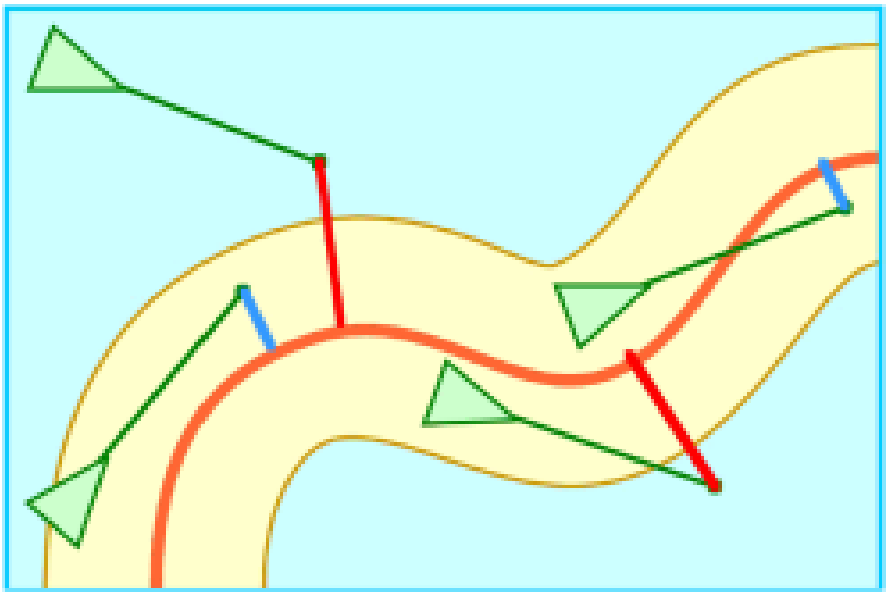

Рисунок 2.5 – Стратегія проходження по «коридору»

Для того, щоб обчислити силу прицілювання у стратегії «проходження шляху», необхідно зазначити майбутнє положення агента, що засноване на швидкості агента. Позиція агента проектується на найближчу опорну точку «коридору» (рис.2.5). Якщо спроектована відстань менше ніж радіусу шляху, можна вважати, що інтелектуальний агент направлений у вірному напрямку і немає необхідності у втручання роботи агента. Інакше, агенту необхідно, використавши стратегію поведінки пошуку, повернутись до «коридору».

2.2 Моделі поведінки агентів з використанням нейронних мереж.

У даній кваліфікаційній роботі створюється інтелектуальний агент, який буде знаходитись у певному тривимірному середовищі та виконувати пошуково-рятувальні задачі.

Як вже було зазначено, середовище буде характеризуватись шириною, висотою та глибиною, а також наявністю агентів та цільових об’єктів для пошуку. Агенти не можуть вийти за межі зазначеного середовища так як воно є замкненим.

Штучна нейронна мережа є спрощеною моделлю людського мозку і являє набір даних та методів для певної взаємодії. Ці елементи називаються нейронами або вузлами. Вони являють собою деякі прості обчислювальні потужності, які обмежуються певним правилом комбінування правила активації та вхідних сигналів, що дозволяють робити обчислення найбільш якісним чисном. Вихідний сигнал відправляється іншим елементам системи по зважених зв'язках, а також кожен з них має ваговий коефіцієнт або вагу.

Залежно від вагового коефіцієнта сигнали або посилюються або послаблюються. Перевага даного напрямку навчання полягає в тому, що агенти можуть самостійно набувати знань в процесі виконання задач, що призводить до більш якісної навченості та можливості виконувати більш складні задачі. Крім того, навченість може стати й недоліком, так як аналізувати навчену нейронну мережу досить складно.

2.3 Навчання нейронної мережі

Ціль навчання ШНМ – досягати створення бажаної моделі виконання певних задач через деяку множину вхідних сигналів, яка називається навчальною вибіркою. Вхідні і вихідні множини сигналів зручно виражати через вектори. Навчання інтелектуального агента здійснюється шляхом послідовного зазначення вхідних векторів з навчальної вибірки та одночасною побудовою мережі з певними параметрами, що відповідають деякому процесу навчання.

Процедура навчання проводиться допоки не буде досягнута цільова реакція штучної нейронної мережі для всієї навчальної вибірки. У математичному змісті навчання штучної нейронної мережі являє собою ітераційний послідовний процес, який спрямований на підстроювання певних параметрів у мережу. Використовуючи такий функціонал необхідно

враховувати функцію помилки, яка несе інформацію про ступінь наближення вхідного вектора до цільового вихідного значення.

Перед формуванням процесу навчання потрібно підготувати модель зовнішнього середовища, в якому інтелектуальний агент повинен навчатись. Таким чином будуть визначені початкові дані, що значно спрощують процес досягнення мети. У рамках певної парадигми навчання далі конструюються правила підстроювання параметрів, іншими словами алгоритм навчання. Існують декілька парадигм навчання: «із учителем», «без учителя» (самонавчання) і змішана. Коли мережа має тільки один єдиний шар, це справедливо назвати алгоритмом навчання “із учителем”, який є достатньо простим, через те, що вихідні стани нейронів цього шару є відомими, і підстроювання синаптичних зв'язків просувається в напрямку мінімізації помилки на виході мережі. За цим принципом будується, наприклад, алгоритм навчання одношарового персептрона.

2.4 Генетичний алгоритм

Генетичний алгоритм, який був розроблений Джоном Холландом, базується на пошуку пошуку потрібного рішення на основі механізмів, що відомі ще з біології. Мова йде про природний відбір та алгоритм «спрямованого» випадкового пошуку. На етапі ініціалізації алгоритму для пошуку потрібного та найкращого рішення потрібно створити популяцію рішень. Виходячи з того, як фактична популяція справляється з пошуком рішення, буде формуватись нове покоління, яке виступає в ролі “вихідного матеріалу” для наступних поколінь. Цикл генетичного алгоритму складається з певних стадій відбору, схрещування і мутацій. Представники, які найкраще виконали завдання – відбираються для створення наступної популяції, тому потрібно припускати, що кожне нове покоління повинно містити кращі рішення, ніж попередні. Популяція у алгоритмі повинна складатись з набору

бінарних рядків, що несуть в собі деякий варіант рішення проблема та має назву “хромосома”. У процесі відбору на першому етапі визначаються рядки, які будуть використані для нового покоління хромосом, тому враховуючи методику організації роботи алгоритму, він має просуватися у самому перспективному напрямку пошуку на кожній ітерації. Схрещування є другим кроком і полягає в тому, що для пари відібраних рядків довжини r кожна повинна вибиратись довільним чином для обміну битами від s+1 до r для утворення нових нащадків . Число s r □{1,..., }. Фінальна стадія – мутація. Під час ініціалізації алгоритму встановлюється певна маленька ймовірність мутації, якій піддаються новоутворені хромосоми (рис. 2.6).

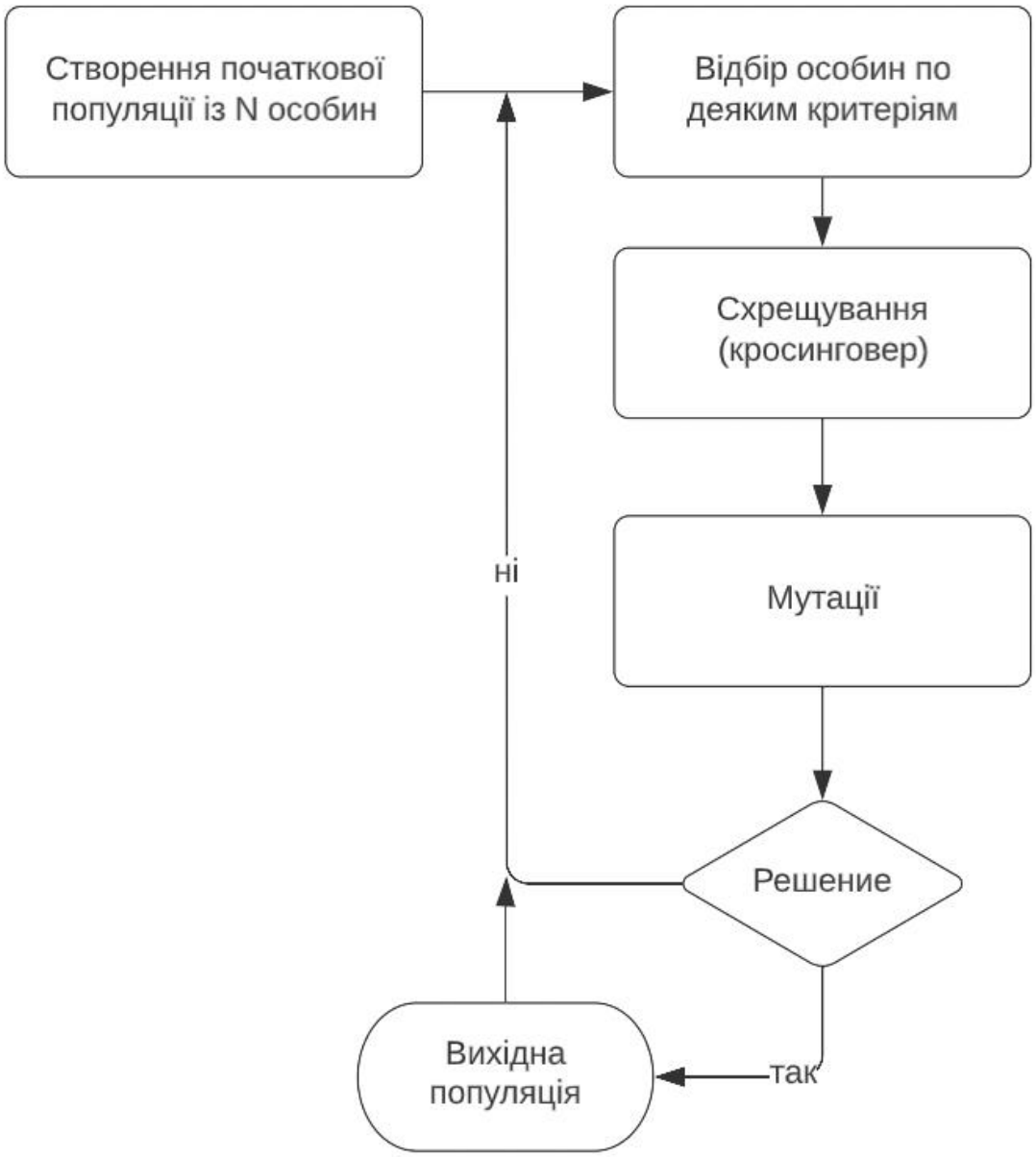

Рисунок 2.6 – Вигляд роботи генетичного алгоритму [10]

Генетичний алгоритм має ітераційний підхід, тому він має наближений розв’язок, який компенсується швидкістю обчислень та області, де застосовується, при обмежених обчислювальних потужностях. Алгоритм зазвичай застосовує підстроювання вихідних шарів та вагових коефіцієнтів у деякому конкретному наборі зв'язків й широко застосовується в задачах навчання нейронних мереж та оптимізації. Звертаючись до математичної моделі, то використовується алгоритм пошуку глобального екстремуму

багатоекстремальної функції, що має за ціль паралельну обробку множин альтернативних розв’язків у системі. При цьому пошук, виходячи з самої суті алгоритму, концентрується на найбільш перспективних із них та відбирає їх на майбутнє. Алгоритм використовує такі визначення:

- ген - є ваговим коефіцієнтом у нейронній мережі;
- хромосома - розв’язки, що складаються з набору генів;
- популяція - велика кількість хромосом, які розглядаються як набори вагових коефіцієнтів для нейронної мережі;
- епоха - це ітерація, яка описує певне покоління хромосом.

Хромосоми є такими сутностями, над якими певним чином в межах конкретної ітерації можуть бути примінені такі операції:

- схрещування;
- мутація;
- пристосування.

Мутація працює над вирішенням проблеми, що має місце у навчанні багатошарової нейронної мережі, використовуючи метод зворотного поширення помилки. Це потрібно для того, щоб популяція вийшла з локального екстремума та впливала на збіжності з результатами попередніх ітерацій. На рис. 2.7 нейронна мережа, вагові коефіцієнти якої складаються з хромосом, що в свою чергу є набором генів[2].

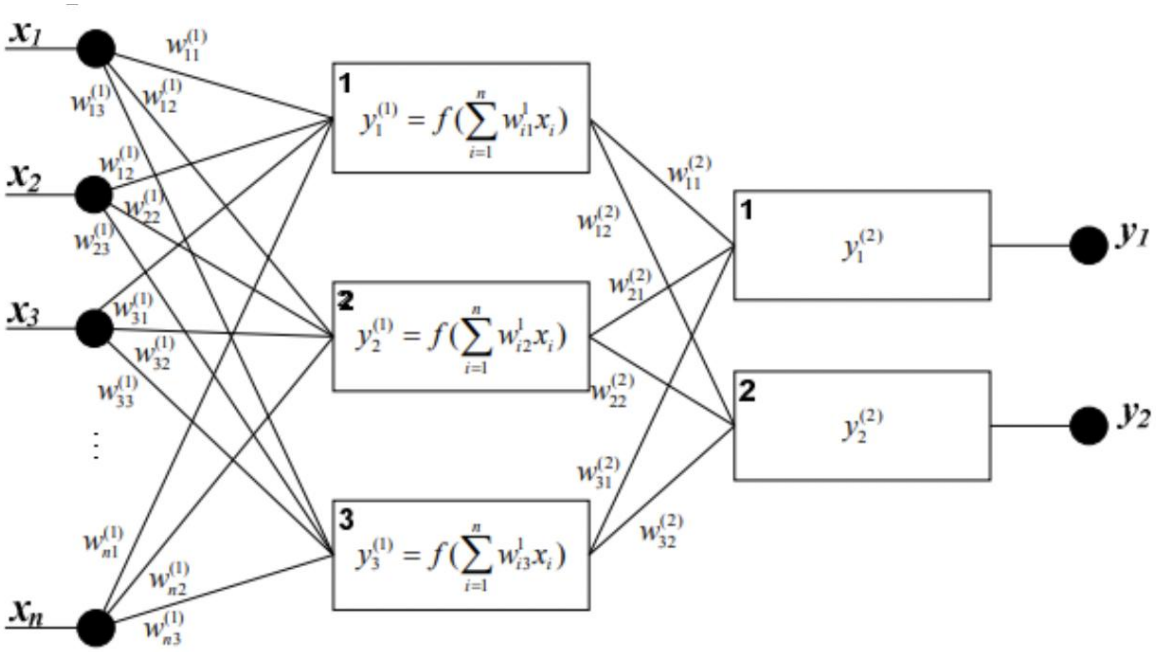

Рисунок 2.7 – Двошарова нейронна мережа[2]

Найважливішим етапом у генетичному алгоритмі є етап відбору батьківських хромосом для нової популяції. У класичному генетичному алгоритмі є два основних генетичних оператора: схрещування і мутації. Імовірність схрещування буде достатньо великою **(зазвичай 0,5 ≤ pc ≤ 1), а ймовірність мутації** має бути встановлена відносно **малою (найчастіше 0 ≤ pm ≤ 0,1).**

Процес навчання нейронної мережі на основі генетичного алгоритму можна побачити на блок-схемі (рис. 2.8).

Основним завданням навчання нейронної мережі зводиться до налаштування вагових коефіцієнтів, де найбільш релевантним можна вважати генетичний алгоритм.

Нижче описана схема роботи алгоритму навчання штучної нейронної мережі генетичним алгоритмом. В загальному:

1) Створюється набір хромосом;

2) Кожна хромосома популяції наповнюється генами з випадковими значеннями, що не суперечать один одному та умові задачі.

3) Перевірити: операції виконано для всіх хромосом у популяції даної ітерації? Якщо ТАК, перейти на крок 8.

4) Якщо НІ, то обрати в популяції пару хромосом з певною ймовірністю та виконати операцію схрещування; нову хромосому додати в нову популяцію для роботи на наступній ітерації.

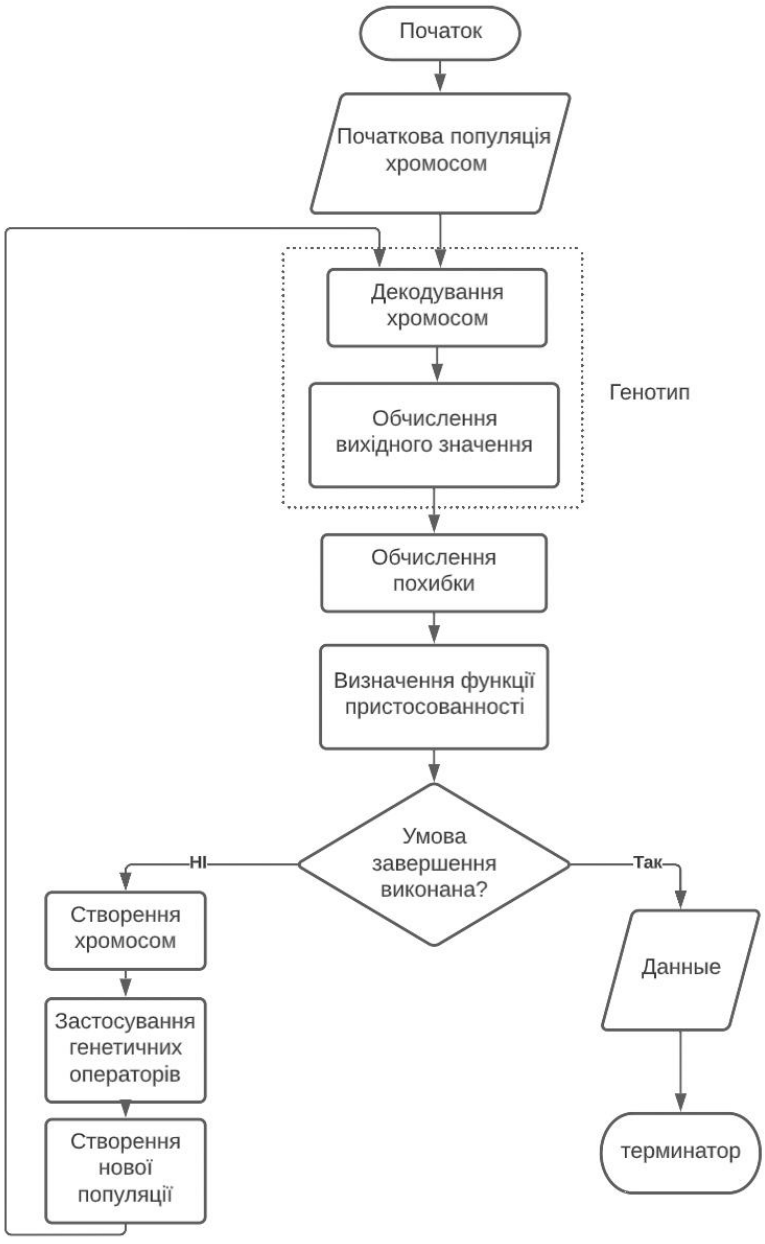

Рисунок 2.8 – Навчання нейронної мережі з використанням генетичного алгоритму [11]

5) Обрати в популяції хромосому з імовірністю pm та виконати операцію мутації генів, після чого додати хромосому в популяцію.

6) Перевірити хромосоми на релевантність розв'язку задачі. Також розташувати хромосоми у порядку відповідності розв’язку задачі.

7) Якщо порядковий номер перевищує первісне число хромосом у фактичній популяції, то видаляємо цю хромосому з популяції. Рухатись до кроку 3.

8) Перевірка: хромосома відповідає розв'язку задачі у популяції? Якщо ТАК, то розв’язок задачі знайдено.

2.5 Q-алгоритм

Q-алгоритм є одним із алгоритмів Reinforcement Learning (навчання з підкріпленням), який навчає модель знаходити оптимальне рішення проблеми шляхом самостійного прийняття рішень. Перевагою даного напрямку є те, що агенти долають проблему отримання практично повністю[13].

Алгоритми Reinforcement Learning складаються з:

- середовища, з яким агент буде взаємодіяти, щоб навчитися досягати мети або виконувати дію;
- винагороди, якщо дія, яку виконує модель, наближає або веде агента до мети. Це робиться для того, щоб навчання моделі рухалось у правильному напрямку.
- негативної винагороди, якщо агент виконує дію, яка не приведе до мети, щоб запобігти навчанню у неправильному напрямку.

Навчання з підкріпленням вимагає моделі машинного навчання, щоб вивчити проблему і самостійно знайти найбільш оптимальне рішення.

Q-Learning – це політика навчання з підкріпленням, яка визначає наступну найкращу дію, враховуючи поточний стан[14]. Він вибирає цю дію

випадково і прагне максимізувати винагороду. На рис. 2.9 можна ознайомитись з ключовими компонентами даного алгоритму.

Рисунок 2.9 – Компоненти Q-алгоритму [14]

Q-навчання — це навчання з підкріпленням без моделі, яке не відповідає правилам, яке знайде найкращий курс дій, враховуючи поточний стан агента. Залежно від того, де агент знаходиться в оточенні, він вирішує наступні дії.

Метою Q-моделі є знайти найкращий курс дій з урахуванням її поточного стану. Для цього він може розробити власні правила або діяти поза політикою, яку йому дано дотримуватися. Це означає, що фактичної потреби в політиці немає, тому ми називаємо її позаполітичною.

Безмодельне означає, що агент використовує прогнози очікуваної реакції середовища, щоб рухатися вперед. Для навчання використовується не система винагород, а метод проб і помилок.

Прикладом Q-learning є система рекомендацій щодо реклами зображена на рис. 2.10. У звичайній системі рекомендацій оголошень реклама, яку отримують користувачі соціальних мереж, базується на попередніх покупках або веб-сайтах, які користувачі могли відвідувати. Якщо користувачі наприклад цікавляться та планують купити або вже здійснили покупку телевізора в

інтернеті, ці ж самі користувачі будуть отримувати рекомендації щодо телевізорів інших брендів [14].

Рисунок 2.10 – Рекомендації на основі Q-алгоритму

Використовуючи Q-learning, з’являється можливість оптимізувати систему рекомендацій оголошень, щоб рекомендувати продукти, які часто купують разом. Винагорода буде, якщо користувач натисне на запропонований товар.

Нижче наведено декілька важливих понять релевантних до алгоритму Q-Learning.

Стани: стан S представляє поточну позицію агента в середовищі.

Дія: Дія A — це крок, який робить агент, коли він знаходиться в певному стані.

Нагороди: за кожну дію агент отримає позитивну або негативну винагороду.

Епізоди: коли агент перебуває в стані завершення і не може виконати нову дію.

Q-Value: Використовується для визначення того, наскільки ефективна дія A, здійснена в певному стані S. Q (A, S)[14].

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

(2.2)

Часова різниця: формула, яка використовується для знаходження Q-значення за допомогою значення поточного стану та дії та попереднього стану та дії. Формула (2.1) зображує процес навчання Q-алгоритму.

2.6 Алгоритм Proximal Policy Optimization

PPO — це метод градієнта політики, який можна використовувати для середовищ з дискретними або безперервними просторами дій.

Методи градієнта політики є основоположними для недавніх проривів у використанні глибоких нейронних мереж для контролю, від відеоігор до тривимірного пересування та Go. Але отримати хороші результати за допомогою методів градієнта політики складно, оскільки вони чутливі до вибору розміру кроку — занадто малий, а прогрес безнадійно повільний; занадто великий, і сигнал перевантажений шумом, або можна побачити катастрофічне падіння продуктивності. Вони також часто мають дуже низьку ефективність вибірки, вимагаючи мільйони (або мільярди) кроків часу для вивчення простих завдань.

Політика оновлюється за допомогою оптимізатора стохастичного градієнтного підйому, тоді як функція значення встановлюється за допомогою певного алгоритму градієнтного спуску. Ця процедура застосовується протягом багатьох епох, поки середовище не буде вирішено[15].

Завдяки навчанню під керівництвом ми можемо легко реалізувати функцію вартості, запустити на ній градієнтний спуск і бути дуже впевненими, що отримаємо чудові результати з відносно невеликим настроюванням гіперпараметрів. Шлях розв’язку задачі у навчанні з підкріпленням

прокладений через роботу з багатьма рухомими частинами, які важко налагоджуються для пошуку найкращих результатів.

PPO в даному випадку балансує між простотою налаштування, простотою впровадження та складністю зразка, тим самим намагаючись обчислити оновлення на кожному кроці роботи, що дає можливість мінімізувати функцію втрат. При цьому відхилення від попередньої політики буде відносно невелике.

У даному алгоритмі впроваджена цільова функція (рис. 2.12), яка в інших алгоритма зустрічається рідко[15]:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\ ] \tag{2.3}$$

$r(\theta)$ – відношення імовірностей;

L^{clip} – функції втрати політики;

E_t – математичне очікування;

A_t – оцінка якості значень від виконаної дії.

Ця ціль реалізує спосіб оновлення Trust Region, сумісного зі стохастичним градієнтним спуском, і спрощує алгоритм, усуваючи потребу вносити адаптивні оновлення.

2.7 Необхідні компоненти для навчання нейронної мережі

Для того, щоб навчати будь-яку мережу, необхідно заздалегідь підготувати деякі ключові елементи[5]:

- віртуальне середовище з підтримкою системи координат та інтеграцією з такими мовами програмування як Python та C#;
- віртуальне середовище, яке може здійснювати візуальне відображення процесу навчання агентів, а також їх створення й наповнення різними параметрами;

- пакет Python 3.8+, який має всі необхідні бібліотеки для моделей на основі нейронних мереж;
- пакетний інсталятор Pip3 для встановлення Python та Numpy
- встановити Numpy - бібліотеку Python, яка надає багатовимірний об’єкт масиву;
- встановити PyTorch - фреймворк машинного навчання.

Для того, щоб відобразити процес навчання моделі, необхідно обрати один з 3D-рушіїв, які є зараз на ринку.

Найкращим прикладом може бути Unity, що дозволяє створювати розумних віртуальних агентів, гравців або ігрових персонажів. Перша інтеграція Unity з Python та нейронними відбулась у 2019 році й наразі стала найбільш поширеною платформою для відображення інтелектуальних агентів в процесі навчання.

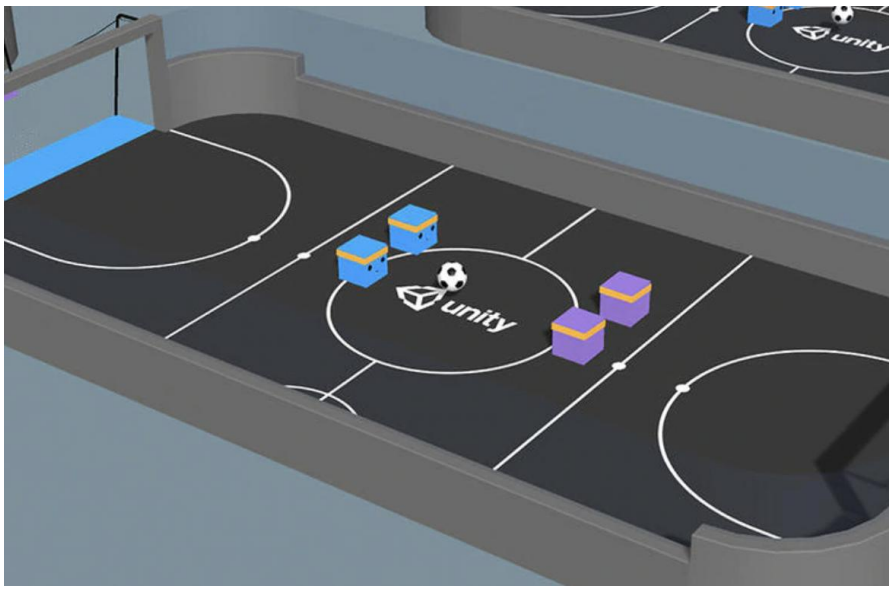

Рисунок 2.13 – Приклад можливого відображення інтелектуальних агентів.

На рис. 2.13 відображений приклад розташування агентів у мультиагентній системі перед їх подальшим навчанням у даному віртуальному середовищі. Цільовим об’єктом для зображених інтелектуальних агентів є футбольний м’яч й основним завданням є забити «гол». В даному випадку це

означає, що середовище Unity є найбільш підготовка до навчання інтелектуальних агентів, як програмно, так і візуально.

2.8 Середовища розробки агентів

На сьогоднішній день існує декілька рушіїв для побудови віртуального середовища та наступної реалізації у них інтелектуальних агентів.

Unreal Engine 5 один із поширених віртуальних середовищ для розробки ігор, що також може бути використаний у даній кваліфікаційній роботі. У середовищі є підтримка навігаційної системи, яка включає в себе різноманітні компоненти та параметри, що змінюють спосіб створення навігаційної сітки, наприклад вартість полігонів. Це, у свою чергу, впливає на те, як агенти переміщуються заданим рівнем.

Навігаційна система включає три режими генерації : статичний , динамічний і лише динамічні модифікатори . Дані режими контролюють спосіб створення навігаційної сітки у проекті та надають різноманітні варіанти, які відповідають вашим потребам. На рис. 2.14 можна побачити віртуальне середовище з декількома агентами.

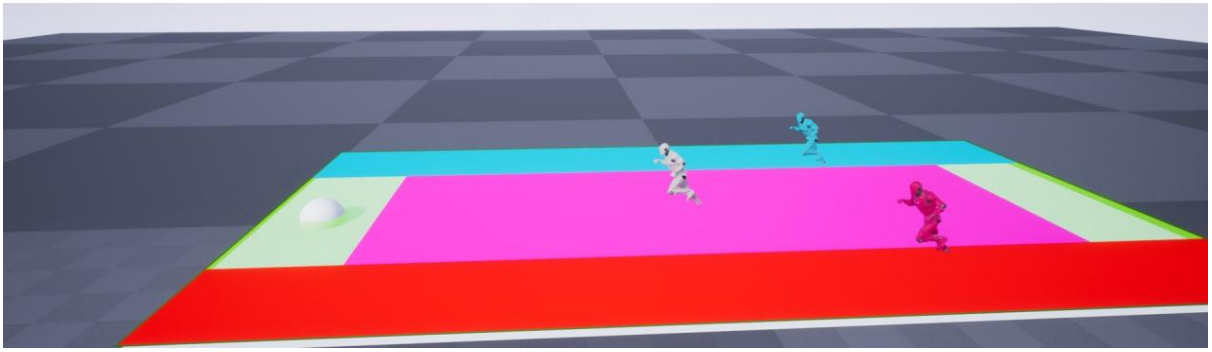

Рисунок 2.14 - Агенти у Unreal Engine 5.

Дане рішення є релевантним до виконання кваліфікаційної роботи. Підтримка у даному середовищі мови C++.

Unity - також один з найпоширеніших рушіїв, які існують на сьогодні. Крім того в середині Unity є спеціально підготовлені програмні рішення, які за допомогою поєднання глибокого навчання з підкріпленням та навчання з імітацією дозволяють достатньо швидко вчити агентів виконувати певні задачі.

Використання ML-Agents дозволяє розробникам створювати більш захоплюючий ігровий процес і покращувати ігровий досвід.

Розвиток досліджень штучного інтелекту (ШІ) залежить від з'ясування складних проблем у існуючих середовищах, використовуючи поточні контрольні показники для навчання моделей ШІ. Однак, коли ці проблеми «вирішуються», виникає потреба в нових середовищах. Але створення таких середовищ часто займає багато часу і вимагає спеціальних знань.

Використовуючи Unity та набір інструментів ML-Agents, можна створювати середовища AI, які є фізично, візуально та когнітивно наповненими. Крім того, у середовищі Unity є можливість виконувати порівняльний аналіз агентів, об'єктів тощо, а також досліджувати нові алгоритми і методи (рис. 2.15).

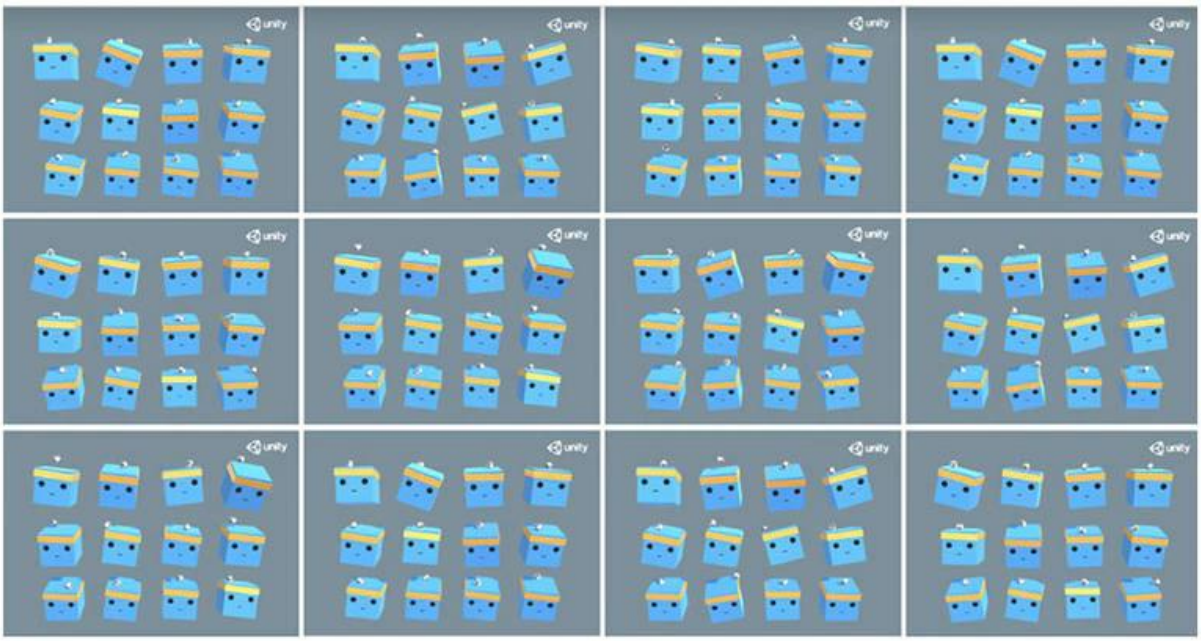

Рисунок 2.15 - Інтелектуальні агенти у Unity.

Виходячи з широкого спектру інструментів, які надає Unity для розробки та управління інтелектуальних агентів, було обрано саме цей тривимірний рушій для подальшої роботи та відображення навчання агентів.

Висновки до 2 розділу

У другому розділі даної кваліфікаційної роботи були описані деякі найбільш поширені поведінки інтелектуальних агентів та стратегії пошуку об’єктів у середовищі, які допомагають досягти цілей даної кваліфікаційної роботи.

Крім того, було розглянуто:

- Q-алгоритм;
- генетичний алгоритм;
- алгоритм Proximal Policy Optimization
- компоненти, які необхідно встановити для навчання нейронної мережі;
- найбільш функціональні 3D-рушії для візуального відображення процесу навчання інтелектуальних агентів, з підтримкою мов програмування C# та Python бібліотек навчання нейронних мереж.

Кожен розробник сам обирає методи розробки відповідно до своїх цілей. Крім того, слід враховувати, що кожне середовище має свої складнощі та швидкість, з якою можна взаємодіяти із середовищем.

Більше того, було обрано алгоритм Proximal Policy Optimization, який є найбільш релевантним з наведених за рахунок легкої реалізації функції вартості, запустити градієнтний спуск та потенційно отримати якісні результати з відносно невеликою кількістю встановлених гіперпараметрів. Усе це спонукає використати даний алгоритм у навчанні інтелектуального агента у третьому розділі кваліфікаційної роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

Як було зазначено у другому розділі, Unity є одним з найпоширеніших рушіїв, які існують. Саме в цьому 3D-рушії будуть проходити подальші налаштування та навчання агента у віртуальному середовищі.

3.1 Базовий опис інтелектуального агента у Unity

Інтелектуальний агент у даній кваліфікаційній роботі виконує пошуково-рятувальні задачі у варіанті, коли немає значних перешкод для роботи. Головна проміжна задача – зрозуміти на скільки ефективно штучний інтелект може виконувати пошук найкоротшого шляху до певних об’єктів та скільки це займає часу. На рисунку 3.1 можна побачити певне програмне середовище, де є зона діяльності, агент та цільовий об’єкт у вигляді сфери.

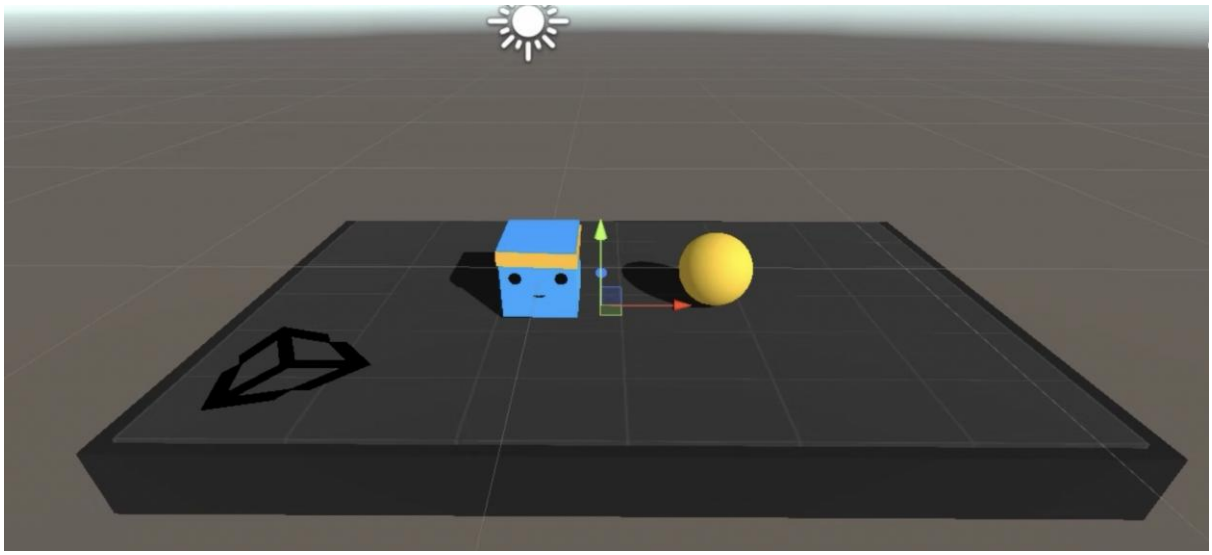

Рисунок 3.1 – Базове відображення тривимірного середовища

Кожен об’єкт у програмному просторі характеризується певними параметрами такими як позиція у просторі за рахунок певних координат X, Y, Z, визначений тип даних, з яким працює середовище та інтелектуальний агент

та базові матеріальні складові на кшталт гравітації та обмеження руху по програмному середовищу.

3.2 Налаштування віртуального середовища Unity для навчання інтелектуального агента.

Для того, щоб навчити інтелектуального агента виконувати певні дії та досягати пошуково-рятувальні цілі, необхідно підготувати робоче середовище та встановити додаткове програмне забезпечення.

В першу чергу потрібно налаштувати візуальну частину роботи, а це:

- створити проект у Unity;
- створити платформу, на якій буде проходити діяльність агента;
- створити самого інтелектуального агента;
- створити сферу, яка буде ціллю.

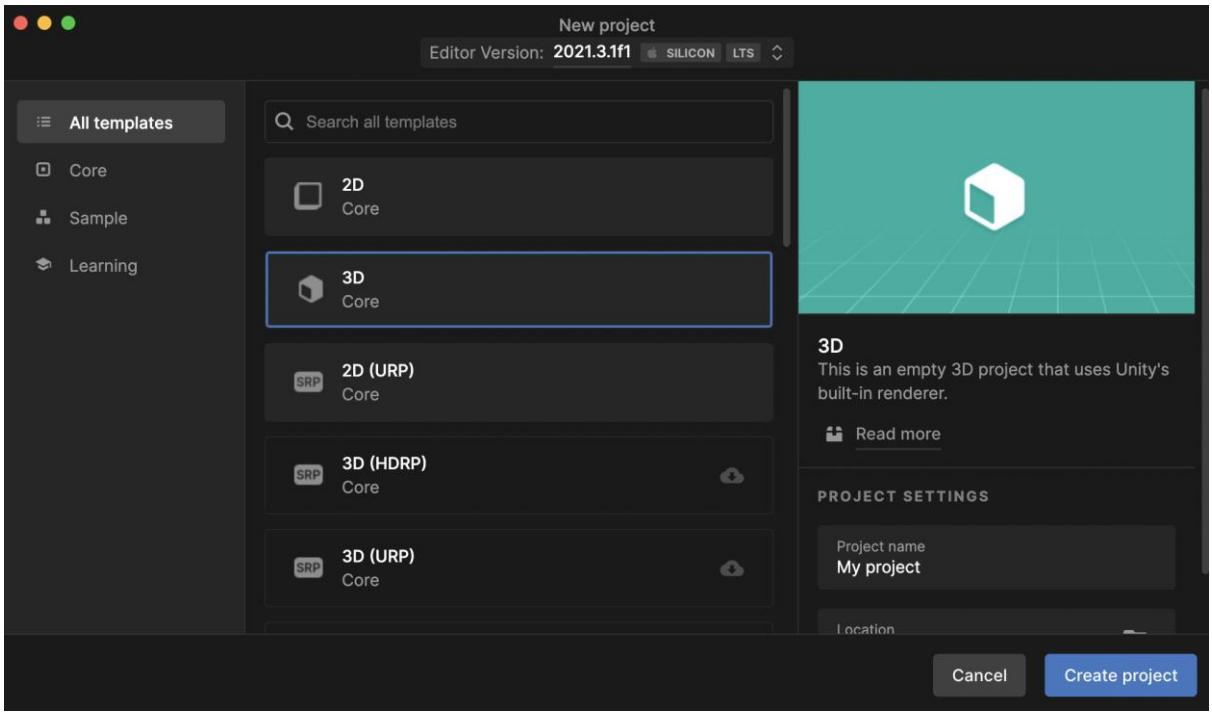

Рисунок 3.2 – Створення проекту в Unity.

На рисунку 3.2 зображений процес створення тривимірного проекту в Unity. Важливо звернути увагу на те, що є різні версії даного комп'ютерного застосунку, під різні платформи – Windows або MacOS. Операційні системи можуть досить сильно відрізнятись, так само як і їх процесори.

Створення оболонки інтелектуального агенту можна здійснити дуже просто. У Assets Store в Unity є багато різних моделей об'єктів, які можна використати у даній системі. Крім того, необхідно завантажити пакет ML Agents у Unity для того, щоб система мала доступи до методів навчання ззовні.

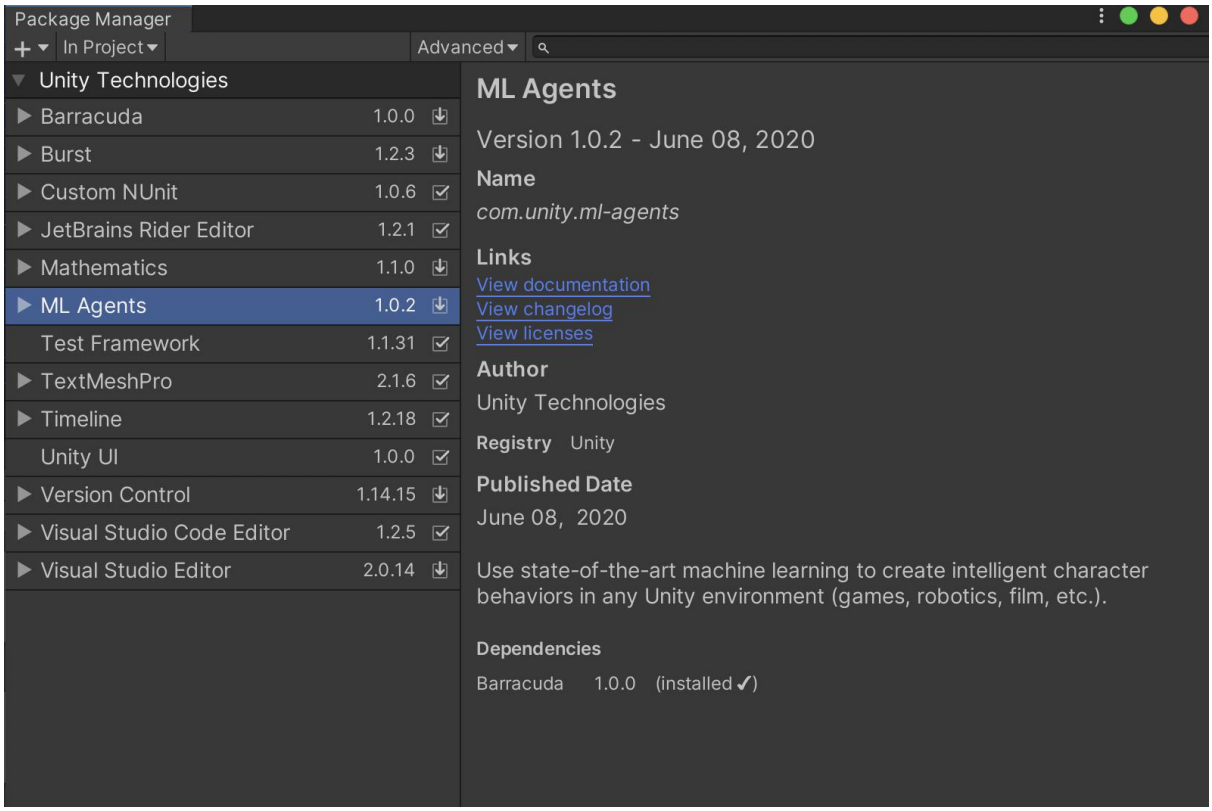

Рисунок 3.3 – Встановлення пакету ML Agents у Unity.

У розділі Package Manager необхідно знайти пакет ML Agents та завантажити його. Даний пакет дозволяє працювати з агентами всередині середовища, моделювати їхню поведінку та підключати агентів до нейронної мережі, яка формується ззовні за рахунок комп'ютерних потужностей. На рисунку 3.3 можна побачити успішно встановлений пакет агентів, що готовий до використання.

Наступним кроком буде розташувати всі необхідні елементи по середовищу та налаштувати певні параметри для послідуючої роботи.

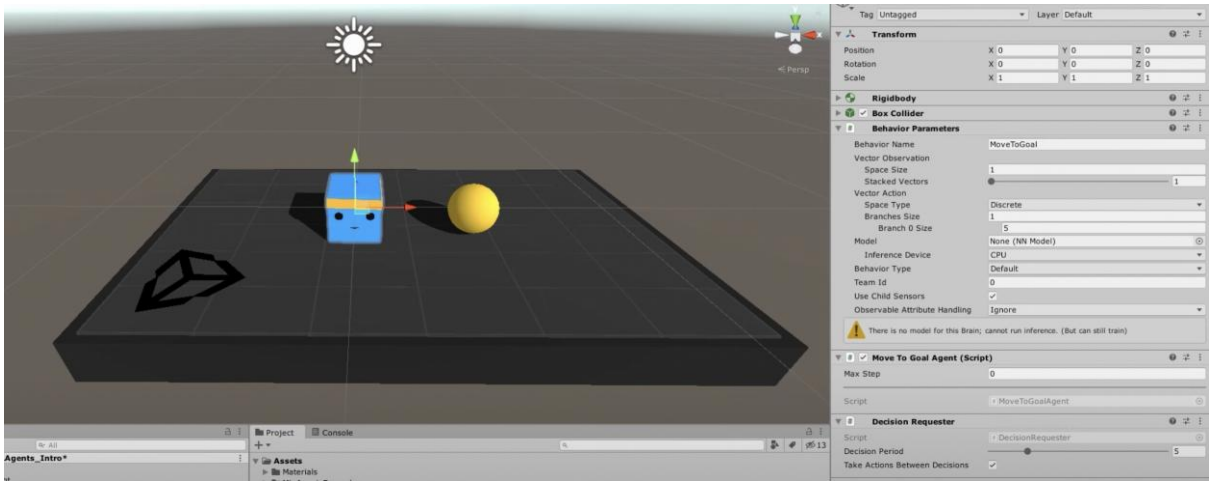

Рисунок 3.4 – встановлення у середовище агента та цільових об’єкт.

На рисунку 3.4 можна побачити не тільки встановлених об’єктів, необхідних для подальшої роботи, а також ряд параметрів та налаштувань доступних у системі. Всі ці параметри можна змінювати у програмному коді.

Більше того, як було зазначено на початку пояснювальної записки, кожен агент повинен мати обмеження. В даному випадку зону діяльності розумного агента треба обмежити до краю платформи. Наближаючись до краю, пошук найкоротшого шляху повинен завершуватись та починатись наново. Для того, щоб встановити всі ці параметри, необхідно звернутись до параметрів Collider та RigidBody, які в свою чергу зупиняють агента від подальших дій та виступають тригером для завершення пошуку. На рисунку 3.5 відображений базовий набір параметрів кожного об’єкту, який рухається у просторі. Серед них:

– налаштування розташування відносно координат X, Y, Z;
– налаштування тригерності при взаємодії з об'єктами;
– налаштування маси об’єктів;
– налаштування формату гравітації;
– налаштування колізії.

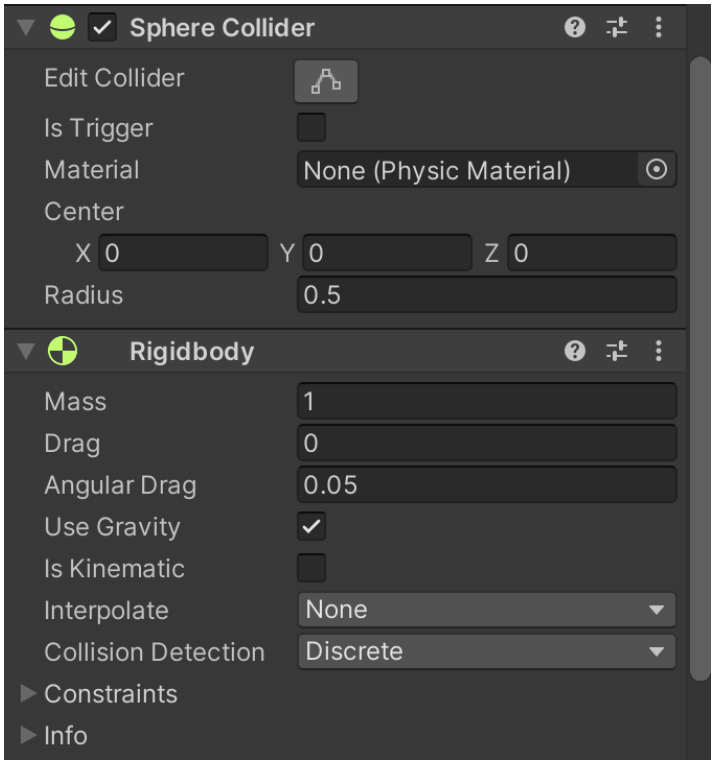

Рисунок 3.5 – налаштування базових параметрів об’єктів у просторі.

Більше того, наступник кроком необхідно звернутися до коду й деяких налаштувань. В нашому випадку нам необхідно переписати деякі функції агентів зі стандартного пакету ML Agents, що мають певні особливості у роботі з середовищем. На рисунку 3.6 зображена функція Heuristic, яка дозволяє нашому агенту переміщатись у програмному середовищі. В даному випадку це вісь X та Z.

```
0 references
public override void Heuristic(float[] actionsOut)
{
    actionsOut[0] = Input.GetAxis("Horizontal"); // x
    actionsOut[1] = Input.GetAxis("Vertical"); // z
}
```

Рисунок 3.6 – Написання коду переміщення у просторі.

Крім того, дуже важливо налаштувати в коді ініціалізацію нашого агента. На рисунку 3.7 відображене програмне налаштування функції Initialize(), що

виконує присвоєння певним змінним у коді значення об’єктів з програмного середовища (RigidBody).

```
0 references
public override void Initialize()
{
    playerRigidbody = GetComponent<Rigidbody>();
    originalPosition = transform.localPosition;
}
```

Рисунок 3.7 – Написання коду ініціалізації агентів у віртуальному середовищі.

Також умісно вказати програмно, в якій саме точці агент починає своє існування під час навчання нейронної мережі.

3.3 Налаштування модулів навчання

Для того, щоб запустити навчання необхідно виконати певні дії відносно таких налаштувань та встановлень, таких як:

- Python 3.8;
- Віртуальне середовище Python;
- PyTorch;
- Baracuda;
- Pip3;
- Numpy;
- ML Agents package.

Перш за все необхідно завантажити Python з офіційної сторінки, яка підтримує навчання нейронної мережі та інтелектуальних агентів. В нашому випадку це повинна бути версія не нижче Python 3.8, яка найбільш стабільно працює під наші цілі. На рисунку 3.8 можна побачити короткий опис можливостей даної версії програмування. Переваги створення Python ідеальним рішенням для машинного навчання та проектів, керованих штучним

інтелектом, включають простоту та послідовність, гнучкість, доступ до потужних бібліотек і фреймворків AI та машинного навчання (ML), незалежність від платформи та великі спільноти. Ці речі підвищують популярність мови.

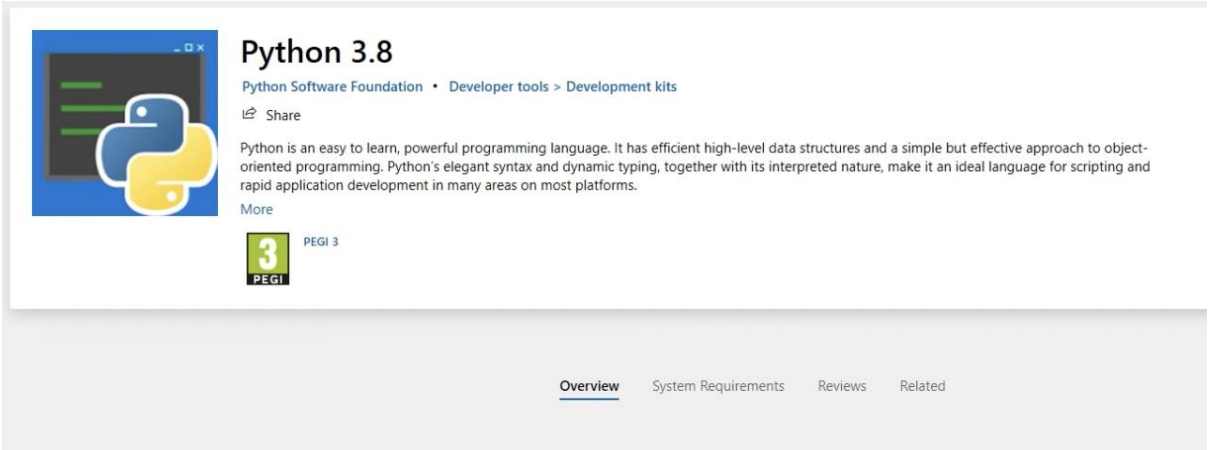

Рисунок 3.8 – Встановлення Python 3.8.

Наступник кроком є створення віртуального середовища Pyhon. Це необхідно для того, щоб будь-які встановлення, зміни тощо відбувались в одному середовищі, а також для роботи програми без переплітання даних та навчальних моделей з інших проектів. Щоб це зробити, необхідно поперше створити віртуальне середовище за допомогою коду на рисунку 3.9 та відкрити його як на рисунку 3.10 у командному рядку.

```
python3 -m venv
```

Рисунок 3.9 – Створення віртуального середовища Python 3.8.

```
(venv) httpoolartem@UA-M-038 environments % source venv/bin/activate
(venv) httpoolartem@UA-M-038 environments %
```

Рисунок 3.10 – Активація віртуального середовища.

Після того, як ми потрапили у віртуальне середовище, починається наступний етап налаштувань, що буде знаходитись тільки у цьому середовищі. Встановлення PyTorch можна побачити на рисунку 3.11, де відображений рядок зі встановленням даного додатку у командному рядку.

Рисунок 3.11 – Встановлення PyTorch.

PyTorch – фреймворк для навчання нейронної мережі, який спрощу взаємодію всіх елементів системи й допомагає будувати модель для інтелектуального агента. Разом з PyTorch також буде встановлено ряд додаткових налаштувань та програмних фреймворків таких як Numpy, Dataclasses та інші (рис. 3.12).

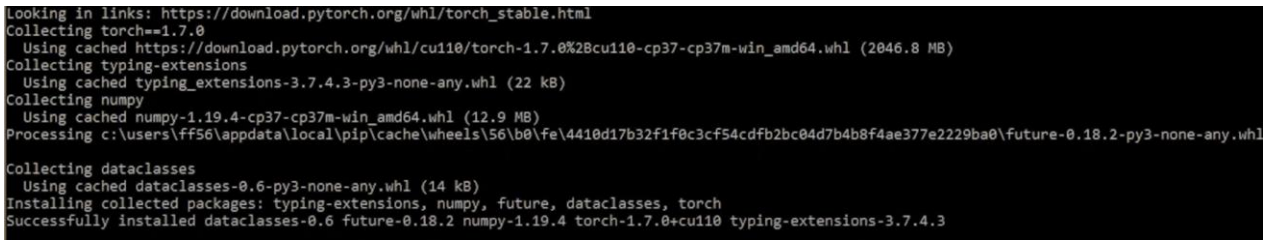

Рисунок 3.12 – Встановлення додаткових налаштувань з PyTorch.

Практично завершальним етапом наших налаштувань є встановлення пакету Python ML Agents, що дозволяє отримувати від агентів з таких середовищ як Unity та саме навчати їх. Для цього використаємо простий код пакету pip3.

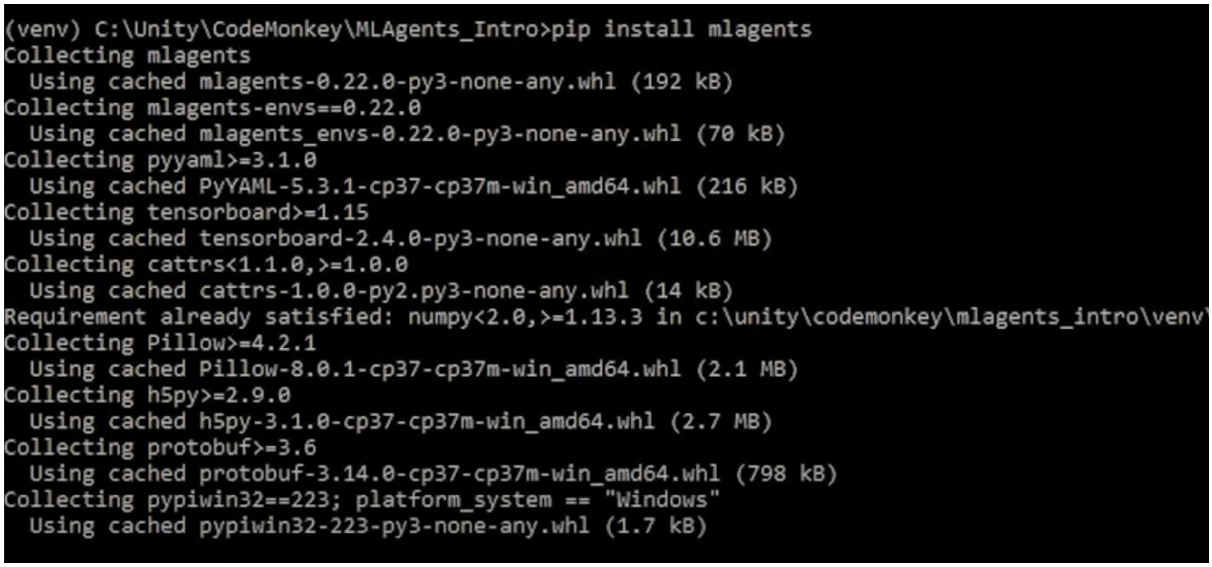

Рисунок 3.12 – встановлення пакету mlagents.

MLagents-learn є основною навчальною утилітою, що надається ML-Agents Toolkit. Агент отримує ряд параметрів CLI на додаток до файлу конфігурації YAML, який містить усі конфігурації та гіперпараметри, які будуть використовуватися під час навчання. Набір конфігурацій і гіперпараметрів, які потрібно включити в цей файл, залежить від агентів у середовищі та конкретного методу навчання, який буде використовується. Значення гіперпараметрів можуть мати великий вплив на продуктивність навчання (тобто на здатність агента вивчати політику, яка вирішує завдання). Далі будуть розглянуті гіперпараметри для всіх методів навчання та надамо рекомендації та поради щодо їх значень.

Тож якщо ми правильно все встановили, отримаємо таке повідомлення від системи (рис. 3.13).

Рисунок 3.13 – Вікно, яке побачить користувач після встановлення mlagents.

Якщо звернути увагу на останній рядок, можна побачити «Listening on port 5004», що означає пошук певної інформації з віртуального середовища Unity через порт 5004.

3.4 Навчання інтелектуального агенту

Виходячи з того, що всі налаштування підготовані, тепер інтелектуальний агент може бути навчений шукати певні об’єкти у віртуальному середовищі. Перш за все потрібно звернути увагу на вхідні дані, які отримує нейронна мережа для навчання розумного агента.

3.4.1 Вхідні дані для нейронної мережі

Для навчання інтелектуального агента було обрано алгоритм Proximal Policy Optimization. На вхід до нейронної мережі необхідно подати ряд параметрів поведінки для навчання інтелектуального агента як:

– тип тренування;

– номер партії;
– розмір буферу;
– епсілон;
– кількість епох;
– швидкість навчання;
– нормалізація даних;
– кількість шарів;
– кількість та якість винагород;
– максимальна кількість кроків;
– час на виконання.

На рисунку 3.14 можна ознайомитись з параметрами, які були введені для навчання нейронну мережу інтелектуального агента.

```
behaviors:
  PlayerMaze:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 500000
    time_horizon: 64
    summary_freq: 5000;
```

Рисунок 3.14 – Вхідні параметри для навчання агента

Всі зазначені параметри являють собою невід’ємну частину навчання інтелектуальних агентів у віртуальному середовищі Unity.

3.4.2 Початок навчання інтелектуального агента

Для того, щоб почати навчання інтелектуального агента, необхідно відкрити одночасно середовище Unity та командний рядок, де потрібно звертатись до засобів навчання нейронної мережі. Рис. 3.15 відображає готовність середовища до навчання.

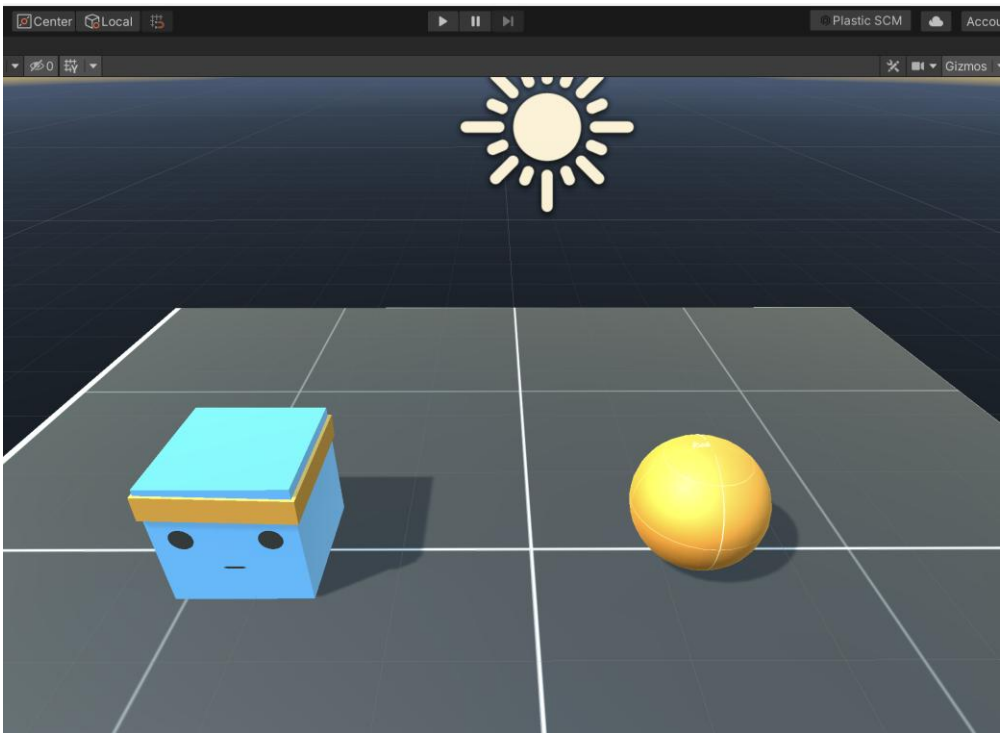

Рисунок 3.15 – Підготовка середовища для навчання агента.

На рис. 3.16 можна побачити командний рядок з інформацією про готовність отримувати інформацію про середовище Unity для навчання агента через порт 5004.

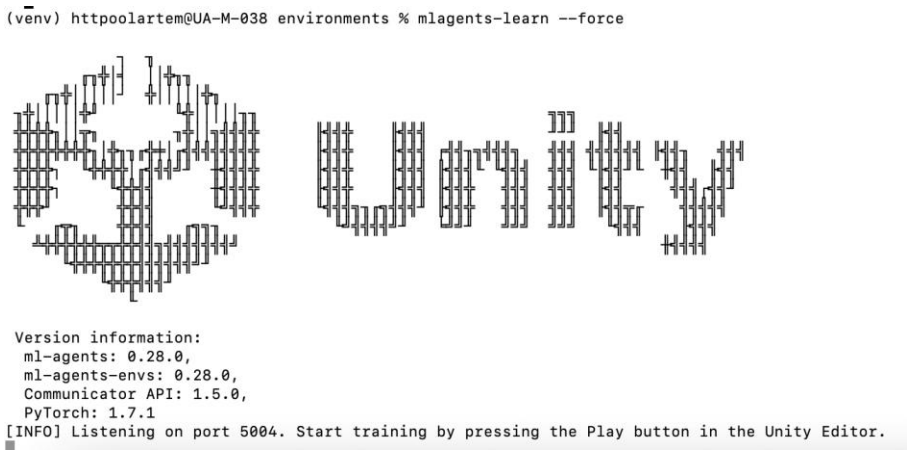

Рисунок 3.16 – Запуск процесу навчання у командному рядку.

На даному етапі процес навчання інтелектуального агенту розпочато у віртуальному середовищі Unity. Агент розпочав пошук найкоротшого шляху до об’єкту, який є ціллю. На рис. 3.17 можна як агент для початку намагається зрозуміти, який саме об’єкт є тою самою ціллю, за що середовище дає винагороду.

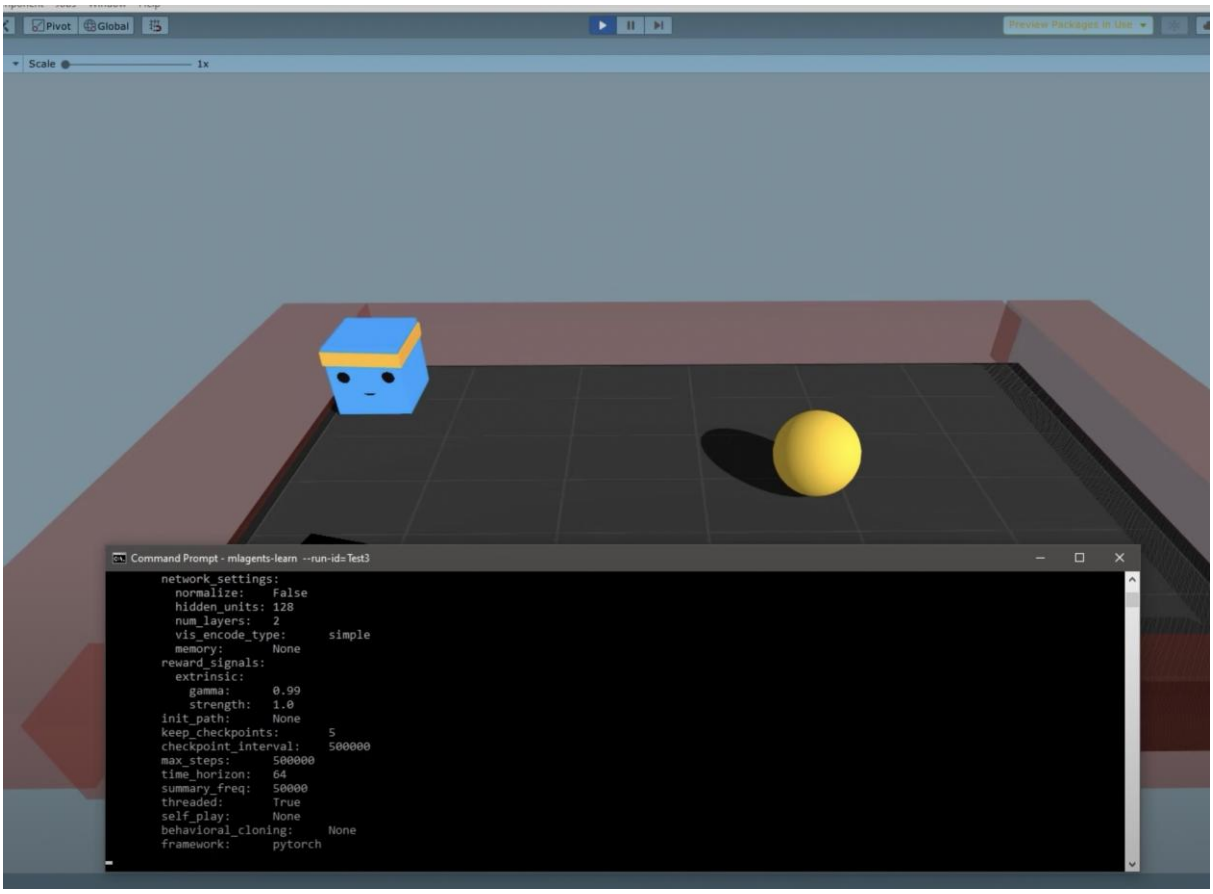

Рисунок 3.17 – Процес пошуку шляху до цільового об’єкту.

Одним із важливих налаштувань агента є встановлення максимальної кількості його кроків. Може бути такий випадок, коли агент не може знайти цільовий об’єкт та починає вчитися рухатись за хибним шляхом. Тому потрібно внести в параметр максимальної кількості кроків.

3.5 Результати навчання інтелектуального агента.

Рис. 3.18 відображає успішне досягнення агентом його цілі за найкоротший час. В даному випадку зеленим платформа, на якій пересувається агент, має зелений колір тому, що ціль була досягнута.

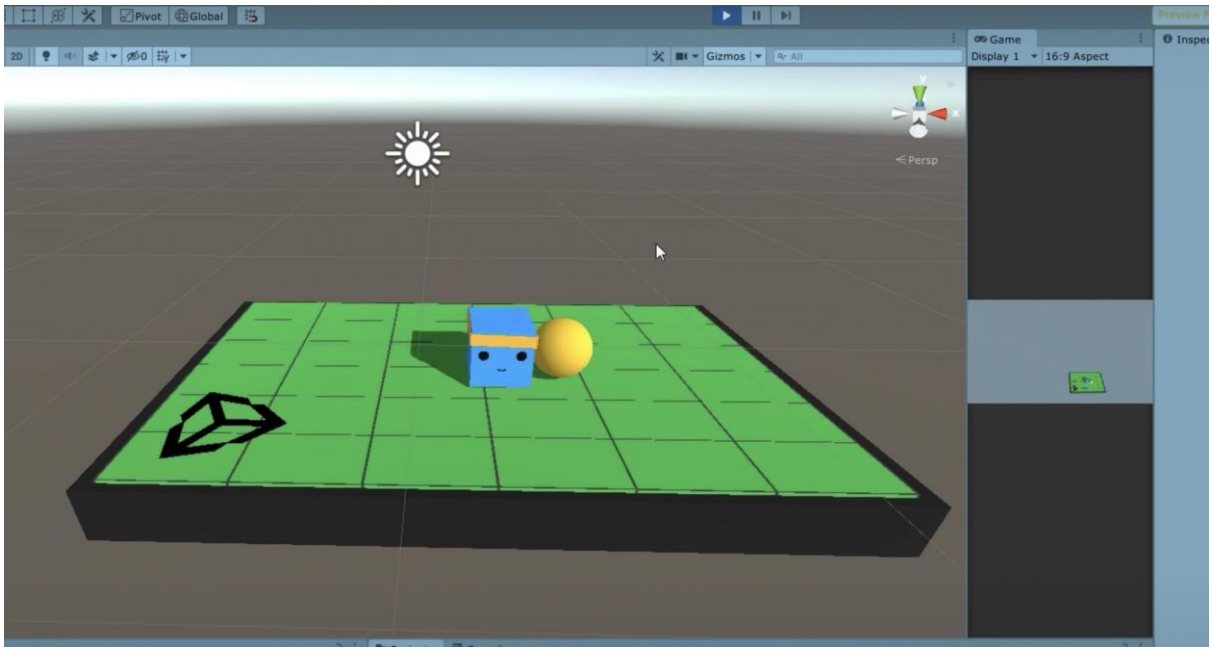

Рис. 3.18. – успішне досягнення агентом об’єкту у середовищі Unity.

Якщо агент хибить, то отримує негативну винагороду й червоний колір платформи.

3.5.1 Аналіз точності моделі

На рисунку 3.19 зображений витяг з командного рядку, де можна побачити кількість кроків, зроблених агентом, час, рівень його винагороди та стандартне відхилення від винагороди. Для навчання було зроблено 70 000 кроків.

```
Starting training from step 0 and saving to results\MoveToGoal2\MoveToGoal.
 Step: 10000. Time Elapsed: 35.482 s. Mean Reward: 0.377. Std of Reward: 0.926. Training.
 Step: 20000. Time Elapsed: 58.636 s. Mean Reward: 0.959. Std of Reward: 0.284. Training.
 Step: 30000. Time Elapsed: 81.177 s. Mean Reward: 0.997. Std of Reward: 0.076. Training.
 Step: 40000. Time Elapsed: 104.175 s. Mean Reward: 0.999. Std of Reward: 0.037. Training.
 Step: 50000. Time Elapsed: 127.559 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
 Step: 60000. Time Elapsed: 151.227 s. Mean Reward: 0.997. Std of Reward: 0.072. Training.
 Step: 70000. Time Elapsed: 174.822 s. Mean Reward: 0.992. Std of Reward: 0.125. Training.
```

Рисунок 3.19 – Процес навчання інтелектуального агента.

Для відображення результатів зручно буде використати застосунок Tensorboard, який має широкі можливості для аналізу навченої моделі нейронних мереж. На рис. 3.20 та 3.21 можна побачити інформацію про покращення моделі нейронної мережі відносно кроків, а також час, який витрачається на виконання задачі в процесі навчання.

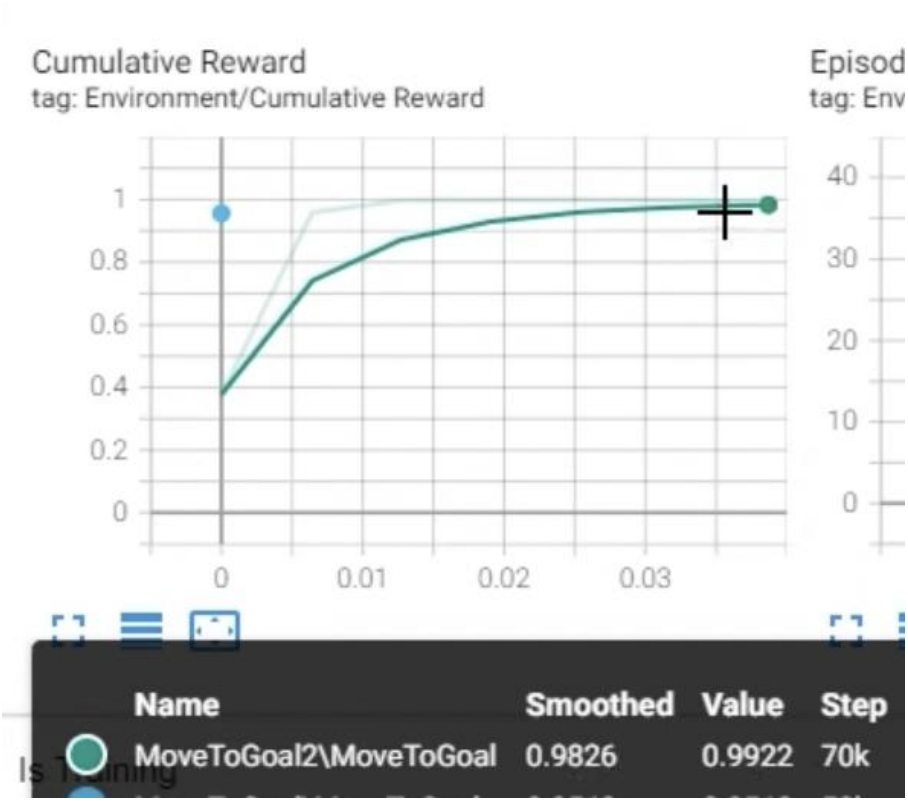

Рисунок 3.20 – Графік покращення моделі нейронної мережі на відстані 70 000 кроків.

Рисунок 3.21 – Графік витраченого часу на виконання задачі за 70 000 кроків.

За час навчання інтелектуального агента можна побачити, що на 70000 кроці інтелектуальний агент практично без помилок досягає поставленої мети та використовує на це мінімальну кількість часу.

ВИСНОВОК

У даній кваліфікаційній роботі було створено інтелектуального агента, який за допомогою нейронної мережі навчився найкоротшим шляхом досягати цільового об’єкта.

Також було проведно:

- аналіз стратегій пошуку інтелектуальним агентом шляху до заданого об'єкта - цілі;
- аналіз алгоритмів пошуку найкращого шляху за заданими критеріями;
- аналіз алгоритмів навчання нейронної мережі для розв'язку задачі пошуку найкращого шляху;
- вибір віртуального середовища для візуального відображення та проведення експериментальних досліджень;
- дослідження ефективності роботи обраного алгоритму шляхом емуляції роботи інтелектуального пошукового агента у віртуальному середовищі.

Оцінюючи результати роботи інтелектуального агента, можна зробити висновок, що при послідовному навчанні агент може виконувати задачі значно швидше та ефективніше, а найголовніше – без прямого втручання людини у пошуково-рятувальний процес.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Яловець А.Л. Мультиагентне моделювання переслідування на площині: від теорії до програмної реалізації / Яловець А.Л., 2019. – 168 с.
2. Пітер Н. Штучний інтелект: сучасний підхід / Н. Пітер, Р. Стюарт., 2009. – 1408 с.
3. Нестеренко О.В. Ковтунець О.В. Інтелектуальні системи і технології / Нестеренко О.В., 2017. – 90 с.
4. Рассел С. Сумісний з людиною. Штучний інтелект і проблема контролю / Рассел С., 2020. – 416 с.
5. Пратик Д. Штучний інтелект з прикладами на Python / Пратик Д., 2020. – 448 с.
6. Нарушев Є., Хорошевський В. AgSDK: інструментарій розробки мультиагентних систем / Нарушев Є., Хорошевський В., 2000. – 798 с.
7. Marr B., Ward M. Artificial Intelligence in Practice: How 50 Successful Companies Used AI and Machine Learning to Solve Problems / Marr B., Ward M., 2019. – 352 с.
8. Alpaydin. E. Machine Learning : The New AI / Alpaydin. E., 2016. – 224 с.
9. Kelleher J. Deep Learning / Kelleher J., 2020. – 296 с.
10. Denning P. Computational Thinking / Denning P., 2020. – 264 с.
11. Sheppard C. Genetic Algorithms with Python / Sheppard C. – 2016. – 264 с.
12. Демидова О. Збитки від стихійних лих у 2021 році у світі склали 280 млрд доларів [Електронний ресурс] / Демидова О. – 2022. – Режим доступу до ресурсу: https://www.dw.com/ru/ushherb-ot-stihijnyh-bedstvij- v-2021-godu-v-mire-sostavil-280-mlrd-dollarov/a-60381352.
13. Amrani A. Q-Learning Algorithm: From Explanation to Implementation [Електронний ресурс] / Amrani A. – 2020. – Режим доступу до ресурсу: https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187.

14. Simplilearn What Is Q-Learning: The Best Guide To Understand Q-Learning [Електронний ресурс] / Simplilearn – 2020. – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning
15. Schulman J., Wolski F. Proximal Policy Optimization [Електронний ресурс] / Schulman J., Wolski F. – 2017. – Режим доступу до ресурсу: https://openai.com/blog/openai-baselines-ppo/.
16. Murphy R. An expert on search and rescue robots explains the technologies used in disasters like the Florida condo collapse [Електронний ресурс] / Murphy R. – 2021. – Режим доступу до ресурсу: https://gcn.com/emerging-tech/2021/07/an-expert-on-search-and-rescue-robots-explains-the-technologies-used-in-disasters-like-the-florida/315596/
17. Шульц Н. Создан робот для поиска людей под завалами [Електронний ресурс] / Шульц Н. – 2017. – Режим доступу до ресурсу: https://fainaidea.com/technologii/roboty/sozdan-robot-dlya-poiska-lyudej-pod-zavalami-129670.html
18. Choudhary A. A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python [Електронний ресурс] / Choudhary A. – 2019. – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python.
19. Honari M. The Unity Machine Learning Agents Toolkit [Електронний ресурс] / Honari M. – 2022. – Режим доступу до ресурсу: https://github.com/Unity-Technologies/ml-agents.
20. Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game 1st Edition / Manning J., Nugent T., – 2019. – 406 с.

ДОДАТКИ

Код програми

```
using System;
using UnityEditor;
using UnityEngine;
using UnityEditor.Build.Reporting;

namespace Unity.MLAgents
{
    public class StandaloneBuildTest
    {
        const string k_OutputCommandLineFlag = "--mlagents-build-output-path";
        const string k_SceneCommandLineFlag = "--mlagents-build-scene-path";

        public static void BuildStandalonePlayerOSX()
        {
            // Read commandline arguments for options
            var outputPath = "testPlayer";
            var scenePath = "Assets/ML-Agents/Examples/3DBall/Scenes/3DBall.unity";

            var args = Environment.GetCommandLineArgs();
            for (var i = 0; i < args.Length - 1; i++)
            {
                if (args[i] == k_OutputCommandLineFlag)
                {
                    outputPath = args[i + 1];
                    Debug.Log($"Overriding output path to {outputPath}");
                }
                else if (args[i] == k_SceneCommandLineFlag)
                {
                    scenePath = args[i + 1];
                }
            }

            string[] scenes = { scenePath };
            var buildResult = BuildPipeline.BuildPlayer(
                scenes,
                outputPath,
                BuildTarget.StandaloneOSX,
                BuildOptions.None
            );
            var isOk = buildResult.summary.result == BuildResult.Succeeded;
            var error = "";
            foreach (var stepInfo in buildResult.steps)
            {
                foreach (var msg in stepInfo.messages)
                {
                    if (msg.type != LogType.Log && msg.type != LogType.Warning)
                    {
                        error += msg.content + "\n";
                    }
                }
            }
            if (isOk)
            {
                EditorApplication.Exit(0);
            }
            else
            {
                Console.Error.WriteLine(error);
                EditorApplication.Exit(1);
```

```
public class ArtAgent : Agent
{
    [Header("Specific to Ball3DHard")]
    public GameObject ball;
    Rigidbody m_BallRb;
    EnvironmentParameters m_ResetParams;

    public override void Initialize()
    {
        m_BallRb = ball.GetComponent<Rigidbody>();
        m_ResetParams = Academy.Instance.EnvironmentParameters;
        SetResetParameters();
    }

    [Observable(numStackedObservations: 9)]
    Vector2 Rotation
    {
        get
        {
            return new Vector2(gameObject.transform.rotation.z, gameObject.transform.rotation.x);
        }
    }

    [Observable(numStackedObservations: 9)]
    Vector3 PositionDelta
    {
        get
        {
            return ball.transform.position - gameObject.transform.position;
        }
    }

    public override void OnActionReceived(ActionBuffers actionBuffers)

    {
        var continuousActions = actionBuffers.ContinuousActions;
        var actionZ = 2f * Mathf.Clamp(continuousActions[0], -1f, 1f);
        var actionX = 2f * Mathf.Clamp(continuousActions[1], -1f, 1f);

        if ((gameObject.transform.rotation.z < 0.25f && actionZ > 0f) ||
            (gameObject.transform.rotation.z > -0.25f && actionZ < 0f))
        {
            gameObject.transform.Rotate(new Vector3(0, 0, 1), actionZ);
        }

        if ((gameObject.transform.rotation.x < 0.25f && actionX > 0f) ||
            (gameObject.transform.rotation.x > -0.25f && actionX < 0f))
        {
            gameObject.transform.Rotate(new Vector3(1, 0, 0), actionX);
        }
        if ((ball.transform.position.y - gameObject.transform.position.y) < -2f ||
            Mathf.Abs(ball.transform.position.x - gameObject.transform.position.x) > 3f ||
            Mathf.Abs(ball.transform.position.z - gameObject.transform.position.z) > 3f)
        {
```

```
        {
            SetReward(-1f);
            EndEpisode();
        }
        else
        {
            SetReward(0.1f);
        }
    }

    public override void OnEpisodeBegin()
    {
        gameObject.transform.rotation = new Quaternion(0f, 0f, 0f, 0f);
        gameObject.transform.Rotate(new Vector3(1, 0, 0), Random.Range(-10f, 10f));
        gameObject.transform.Rotate(new Vector3(0, 0, 1), Random.Range(-10f, 10f));
        m_BallRb.velocity = new Vector3(0f, 0f, 0f);
        ball.transform.position = new Vector3(Random.Range(-1.5f, 1.5f), 4f, Random.Range(-1.5f, 1.5f))
            + gameObject.transform.position;
    }

    public void SetBall()
    {
        //Set the attributes of the ball by fetching the information from the academy
        m_BallRb.mass = m_ResetParams.GetWithDefault("mass", 1.0f);
        var scale = m_ResetParams.GetWithDefault("scale", 1.0f);
        ball.transform.localScale = new Vector3(scale, scale, scale);
    }

    public void SetResetParameters()
    {
        SetBall();
    }
}
```

```
namespace Unity.MLAgentsExamples
{
    /// <summary>
    /// Utility class to allow the NNModel file for an agent to be overriden during inference.
    /// This is used internally to validate the file after training is done.
    /// The behavior name to override and file path are specified on the commandline, e.g.
    /// player.exe --mlagents-override-model-directory /path/to/models
    ///
    /// Additionally, a number of episodes to run can be specified; after this, the application will quit.
    /// Note this will only work with example scenes that have 1:1 Agent:Behaviors. More complicated scenes like WallJump
    /// probably won't override correctly.
    /// </summary>
    public class ModelOverrider : MonoBehaviour
    {
        HashSet<string> k_SupportedExtensions = new HashSet<string> { "nn", "onnx" };
        const string k_CommandLineModelOverrideDirectoryFlag = "--mlagents-override-model-directory";
        const string k_CommandLineModelOverrideExtensionFlag = "--mlagents-override-model-extension";
        const string k_CommandLineQuitAfterEpisodesFlag = "--mlagents-quit-after-episodes";
        const string k_CommandLineQuitAfterSeconds = "--mlagents-quit-after-seconds";
        const string k_CommandLineQuitOnLoadFailure = "--mlagents-quit-on-load-failure";

        // The attached Agent
        Agent m_Agent;

        // Whether or not the commandline args have already been processed.
        // Used to make sure that HasOverrides doesn't spam the logs if it's called multiple times.
        private bool m_HaveProcessedCommandLine;

        string m_BehaviorNameOverrideDirectory;

        private List<string> m_OverrideExtensions = new List<string>();

        // Cached loaded NNModels, with the behavior name as the key.
        Dictionary<string, NNModel> m_CachedModels = new Dictionary<string, NNModel>();


        // Max episodes to run. Only used if > 0
        // Will default to 1 if override models are specified, otherwise 0.
        int m_MaxEpisodes;

        // Deadline - exit if the time exceeds this
        DateTime m_Deadline = DateTime.MaxValue;

        int m_NumSteps;
        int m_PreviousNumSteps;
        int m_PreviousAgentCompletedEpisodes;
```

```
bool m_QuitOnLoadFailure;
[Tooltip("Debug values to be used in place of the command line for overriding models.")]
public string debugCommandLineOverride;

// Static values to keep track of completed episodes and steps across resets
// These are updated in OnDisable.
static int s_PreviousAgentCompletedEpisodes;
static int s_PreviousNumSteps;

int TotalCompletedEpisodes
{
    get { return m_PreviousAgentCompletedEpisodes + (m_Agent == null ? 0 : m_Agent.CompletedEpisodes); }
}

int TotalNumSteps
{
    get { return m_PreviousNumSteps + m_NumSteps; }
}

public bool HasOverrides
{
    get
    {
        GetAssetPathFromCommandLine();
        return !string.IsNullOrEmpty(m_BehaviorNameOverrideDirectory);
    }
}

public static string GetOverrideBehaviorName(string originalBehaviorName)
{
    return $"Override_{originalBehaviorName}";
}
```