Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

МОДЕЛЮВАННЯ ДІЙ РУХОМИХ ОБ'ЄКТІВ У КОМП'ЮТЕРНІЙ ГРІ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-1

Кузьменко П. М.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Гороховатський В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(код і повна назва)
Тип програми освітньо-професійна
Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ
Зав. кафедри _____
(підпис)
«___»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кузьменкові Пилипу Михайловичу
(прізвище, ім'я, по батькові)
1. Тема роботи Моделювання дій рухомих об'єктів у комп'ютерній грі

затверджена наказом університету від 20 травня 2024 року № 464 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23 травня 2024 р.
3. Вихідні дані до роботи науково-методична та науково-технічна літератури, дані інтернет-мережі, ігровий двигун Unity, пакет ML Agents.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд ігрового рушія Unity.
2. Огляд алгоритму навчання з підкріпленням MA-POCA.
3. Механізм Self-play та ELO рейтинг.
4. Програмна реалізація та аналіз результатів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Приклад ігрового об'єкта з компонентами Transform, Mesh Filter, Mesh Renderer, Box Collider та Rigidbody; приклад скрипту AreaReplicator; компонент AreaReplicator на основі скрипту AreaReplicator; полігональна сітка ігрового персонажа та вигляд персонажа відображеного з матеріалами у вікні Unity; параметри імпорту тривимірної моделі; візуальна ілюстрація рівняння Фонга; приклад структури нейронної мережі прямого поширення; UML діаграма класів; компоненти об'єкту Canvas; префаби користувацького інтерфейсу; процес навчання; передбачені результати матчу між двома гравцями; ELO рейтинг моделі в залежності від кількості кроків симуляції; головне меню гри; режим спостерігача; режим прямого керування; ігрове меню

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по-батькові)	Позначка консультатна про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.24	
2	Аналіз завдання, підбір літератури	08.04.24	
3	Аналіз літератури з досліджуваної проблеми	08.04.24-15.04.24	
4	Аналіз технічних засобів	16.04.24-18.04.24	
5	Програмна реалізація	08.04.24-25.04.24	
6	Оформлення пояснювальної записки	08.04.24-21.05.24	
7	Перевірка на плагіат	27.05.24	
8	Рецензування	28.05.2024	
9	Підготовка презентації та доповіді	13.05.24–02.06.24	
10	Занесення роботи в електронний архів	03.06.24	
11	Попередній захист кваліфікаційної роботи	03.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Гороховатський В. О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи 68 с., 3 табл., 17 рис., 51 джерело.

КОМП'ЮТЕРНА ГРА, ШТУЧНІ НЕЙРОННІ МЕРЕЖІ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, ML AGENTS, MA-POCA, SELF-PLAY.

Об'єктом роботи є програмне моделювання дій рухомих об'єктів при створенні комп'ютерної гри у футбол. Для здійснення моделювання дій агентів використана штучна нейронна мережа прямого поширення. Гравець може як спостерігати хід гри в автономному режимі, так і приймати безпосередню участь у грі, беручи прямий контроль над довільним агентом гри.

Метою роботи є розробка комп'ютерної гри з використанням агентів штучного інтелекту, що використовують штучні нейронні мережі прямого поширення та навчаються з використанням алгоритму навчання з підкріпленням MA-POCA та механізму Self-play.

Для розробки були використані Unity версії 2022.3, пакет ML Agents, Visual Studio Code та середовище Anaconda з пакетом mlagents.

COMPUTER GAME, ARTIFICIAL NEURAL NETWORKS, REINFORCEMENT LEARNING, ML AGENTS, MA-POCA, SELF-PLAY.

The object of the work is the program modeling of actions of the moving objects when creating a football computer game. An artificial neural network of direct propagation was used to model the actions of agents. The player can either watch the game progress passively or take direct part in the game by taking direct control of an arbitrary agent of the game.

The aim of the work is to develop a computer game using artificial intelligence agents that use forward propagation artificial neural networks and learn using the MA-POCA reinforcement learning algorithm and the Self-play mechanism.

Unity version 2022.3, ML Agents package, Visual Studio Code and Anaconda environment with mlagents package were used in the development.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

ПК – персональний комп’ютер

GLSL – OpenGL Shading Language

HLSL – High-Level Shading Language

GPU – Graphics Processing Unit (графічний процесор)

PBR – Physically Based Rendering

NPC – Non-Player Character (неігровий персонаж)

PPO – Proximal Policy Optimization

MA-POCA – Multi-Agent Posthumous Credit Assignment

MARL – Multi-Agent Reinforcement Learning (мультиагентне навчання з підкріпленням)

POMDP – partially observable Markov decision process (частково спостережуваний марковський процес прийняття рішень)

IACC – Independent Actor with Centralized Critic (незалежний актор із централізованим критиком)

IAC – Independent Actor-Critic (незалежний актор-критик)

JAC – Joint Actor-Critic (спільний актор-критик)

UI – User Interface (користувацький інтерфейс)

GUI – Graphical User Interface (графічний користувацький інтерфейс)

ВСТУП

За останні роки відбувся значний прогрес у дослідженні глибокого навчання з підкріпленням і розробки алгоритмів [1-4]. Суттєвим фактором цього швидкого розвитку стала наявність складних і масштабованих платформ моделювання, таких як Arcade Learning Environment, VizDoom, MuJoCo, та багато інших [5]. Навчальне середовище Arcade Learning Environment (ALE), наприклад, мало важливе значення для забезпечення засобів порівняльного аналізу підходу керування з пікселів Deep Q-Network. Подібним чином інші середовища та платформи допомогли мотивувати дослідження ефективніших і потужніших алгоритмів [6, 7]. Симуляційне середовище є основним способом, за допомогою якого спільнота навчання з підкріпленням перевіряє свої ідеї та алгоритми. Таким чином, якість навколишнього середовища має вирішальне значення. Дивно, але загальна дискусія навколо цього інтегрального компонента недостатньо розвинена порівняно з його алгоритмічним аналогом.

Багато сучасних дослідницьких платформ базуються на популярних відеоіграх або ігрових рушіях, таких як Atari 2600, Quake III, Doom і Minecraft. Це частина довгострокової тенденції, в якій ігри слугували платформою для досліджень штучного інтелекту (ШІ). Цю тенденцію можна простежити до найдавніших робіт у сфері штучного інтелекту щодо таких ігор, як шахи та шашки, або пізніших робіт із застосування навчання з підкріпленням до гри в нарди. Необхідний пошук, прийняття рішень і планування, які роблять відеоігри цікавим викликом для людей, також є тим самим викликом, які цікавлять дослідників ШІ. Це розуміння спонукало до широкого спектру досліджень перетину відеоігор і штучного інтелекту з різних точок зору гри, моделювання гравців і створення контенту [8].

У міру того, як алгоритми навчання з глибоким підкріпленням стають складнішими, існуючі середовища та тести на їх основі стають менш інформативними. Наприклад, більшість середовищ в ALE було вирішено до рівня, що перевищує людський, що робить подальше використання еталонного тесту менш цінним [9]. Додатковим моментом, створеним цим станом алгоритмічного прогресу, є те, що існує доброчесне коло, у якому розробка нових середовищ стимулює розробку нових алгоритмів. Ми можемо очікувати, що дослідницьке співтовариство й надалі надаватиме високоякісні алгоритми. Однак незрозуміло, звідки дослідникам слід очікувати високоякісних середовищ, оскільки створення таких середовищ часто займає багато часу та вимагає спеціальних знань у галузі. Ця постійна потреба в нових середовищах вимагає простої у використанні, гнучкої та універсальної платформи для необмеженого створення середовища.

Змодельоване середовище обмежене обмеженнями самих симуляторів. Симулятори не є рівноцінними у своїй здатності забезпечувати значущі виклики системам навчання. Крім того, іноді неочевидно, які властивості середовища роблять його вагомим еталоном для дослідження. Складність фізичного світу є основним кандидатом на те, щоб кинути виклик поточним алгоритмам, а також алгоритмам, які планується розробити. Саме у фізичному світі розвинувся інтелект ссавців і, точніше, людський інтелект, і саме цей тип інтелекту дослідники часто зацікавлені у відтворенні [10].

Сучасні ігрові рушії – це потужні інструменти для моделювання візуально реалістичних світів зі складною фізикою та складною взаємодією між агентами з різними можливостями. Крім того, рушії, розроблені для розробки ігор, забезпечують інтерфейси користувача, які спеціально розроблені, щоб бути інтуїтивно зрозумілими, простими у використанні, інтерактивними та доступними на багатьох платформах. Таким чином, у цій кваліфікаційній роботі ми стверджуємо, що ігрові рушії ідеально готові дати необхідні виклики для осяжного майбутнього досліджень ШІ. Для спільноти

це дало б можливість тестувати алгоритми в доменах з такою ж глибиною та різноманітністю, як і сучасні відеоігри [11].

Нейронна мережа забезпечує рух та обертання агентів, ціллю яких є забиття голу у ворота супротивника в тому числі шляхом командної взаємодії. Спостереження агентів формується шляхом трасування декількох променів (цей процес відомий як рейкастинг) та введення такої інформації як дистанція та тег об’єкту, на який натрапив промінь. Тег вказує на інформацію про об’єкт: пурпурний агент (purpleAgent), синій агент (blueAgent), м’яч (ball), стіна (wall), сині ворота (blueGoal), пурпурні ворота (purpleGoal).

Результати моделювання показали, що штучна нейронна мережа прямого поширення ефективно змоделювала дії агентів у комп’ютерній грі у футбол. Використовуючи цю штучну нейронну мережу прямого поширення, рух та обертання агентів точно контролювалися, щоб досягти мети забити у ворота суперника гол за допомогою індивідуальних дій або командної взаємодії. Спостереження агентів були згенеровані за допомогою рейкастингу, де кілька променів відстежувалися для збору інформації, такої як відстань і теги об’єктів. Ці теги, включно з тегами пурпурний агент (purpleAgent), синій агент (blueAgent), м’яч (ball), стіна (wall), сині ворота (blueGoal), пурпурні ворота (purpleGoal), надавали ключові дані для прийняття агентами зважених рішень під час гри.

Кваліфікаційну роботу було структуровано наступним чином: ми починаємо із обґрунтування вибору та огляду ігрового рушія Unity, після чого робимо огляд нейронних мереж та пакету ML Agents та закінчуємо кваліфікаційну роботу на практичній частині, де міститься інформація щодо того яким чином було реалізовано програмний засіб кваліфікаційної роботи.

1 ОБҐРУНТУВАННЯ ВИБОРУ ТА АНАЛІЗ ФУНКЦІОНАЛУ ІГРОВОГО РУШІЯ UNITY

1.1 Аналіз середовища програмної реалізації

Існуючі ігрові рушії є корисними для розробки власної комп’ютерної гри завдяки своїм надійним функціям та інструментам, які спрощують процес розробки гри. Ці рушії забезпечують широкий спектр функціональних можливостей, таких як візуалізація графіки, симуляція фізики, керування аудіо та сценаріями (скриптами), що дозволяє розробникам зосередитися на дизайні гри та створенні вмісту, замість того щоб створювати ці компоненти з нуля. Крім того, ігрові рушії часто мають вбудовану підтримку багатоплатформеності, що полегшує розгортання гри на різних пристроях. Загалом, використання вже існуючого ігрового рушія може заощадити час, ресурси та зусилля, дозволяючи розробникам ефективніше втілювати свої ігрові ідеї.

Серед вже існуючих ігрових рушіїв можна виділити насттупні рушії: Unity, Unreal Engine та Godot. Усі три рушія є безкоштовними, втім для Unity безкоштовною є лише базова версія продукту.

Розглянемо переваги та недоліки кожного з перелічених рушіїв.

Переваги рушія Unity:

– кросплатформність: Unity дозволяє розробникам розгортати свої ігри на кількох платформах, включаючи ПК, мобільні пристрої та консолі, що може допомогти охопити ширшу аудиторію;

– Asset Store: Unity Asset Store пропонує величезну бібліотеку ресурсів, включаючи тривімірні моделі, шейдери, плагіни та скрипти, які

можуть прискорити розробку та покращити візуальні та функціональні аспекти гри;

– підтримка спільноти: Unity має велику й акртивну спільноту розробників і користувачів, які можуть надати допомогу, поділитися знаннями та запропонувати ресурси, щоб допомогти подолати труднощі під час розробки;

– наявність пакету для машинного навчання: Unity дає доступ до пакету машинного навчання під назвою ML Agents, що дозволяє швидко то легко створювати та навчати інтелектуальних агентів для потреб гри.

Недоліки рушія Unity:

– умовна безкоштовність: ігровий рушій Unity є безкоштовним лише у випадку, якщо прибуток за останні 12 місяців не перевищує 100 000 доларів США, окрім цього, як приклад, у безкоштовній версії рушія неможливо прибрати логотип Unity під час запуску програмного продукту;

– низька швидкодія: у випадку, якщо програмний продукт використовує складні та інтенсивні обчислення, швидкодії рушія Unity може виявитися недостатньо, особливо на пристроях нижчого класу.

Переваги рушія Unreal Engine:

– кросплатформність: Unreal Engine дозволяє розробникам розгортати свої ігри на кількох платформах, включаючи ПК, мобільні пристрої та консолі, що може допомогти охопити ширшу аудиторію;

– висока швидкодія: Unreal Engine використовує мову програмування C++ для скриптингу, що забезпечує достатньо високий рівень швидкодії та ефективності у використанні ресурсів пристрою;

– наявність плагіну для машинного навчання: Unreal Engine дає доступ до плагіну машинного навчання під назвою Learning Agents, що дозволяє швидко то легко створювати та навчати інтелектуальних агентів для потреб гри.

Недоліки рушія Unreal Engine:

– умовна безкоштовність: у випадку, якщо прибуток розробників гри за квартал перевищує 10 000$, розробники забов’язані сплатити 5% від прибутку гри. Насправді, модель відрахувань є дещо складнішою за наведену;

– складність: рушій Unreal Engine є більш складним в освоєнні у порівнянні з іншим ігровими рушіями. Таким чином, це може негативно вплинути на швидкість та кількість проблем при розробці ігор на цьому рушії.

Переваги рушія Godot:

– кросплатформність: Godot дозволяє розробникам розгортати свої ігри на кількох платформах, включаючи ПК, мобільні пристрої та консолі, що може допомогти охопити ширшу аудиторію;

– повна безкоштовність: рушій Godot не має платних версій та не потребує жодних відрахувань з прибутку програмного продукту, таким чином Godot є повністю безкоштовним;

– відкритий програмний код: код ігрового рушія Godot є відкритим та знаходиться під ліцензією MIT;

– наявність пакету для машинного навчання: Godot дає доступ до пакету машинного навчання під назвою Godot RL Agents, що дозволяє швидко то легко створювати та навчати інтелектуальних агентів для потреб гри.

Недоліки рушія Godot:

– низька швидкодія: у випадку, якщо програмний продукт використовує складні та інтенсивні обчислення, швидкодії рушія Godot може виявитися недостатньо, особливо на пристроях нижчого класу.

Таким чином обраним ігровим рушієм є Unity. Такий вибір насамперед обумовлюється наявністю досвіду роботи із цим рушієм та наявністю пакету для машинного навчання ML Agents.

1.2 Аналіз функціоналу Unity

GameObject (ігровий об’єкт) – це фундаментальний об’єкт в ігровому рушії Unity, який представляє персонажів, декорації, рослинність тощо. Кожен об’єкт у створеній грі є GameObject (рис. 1.1).

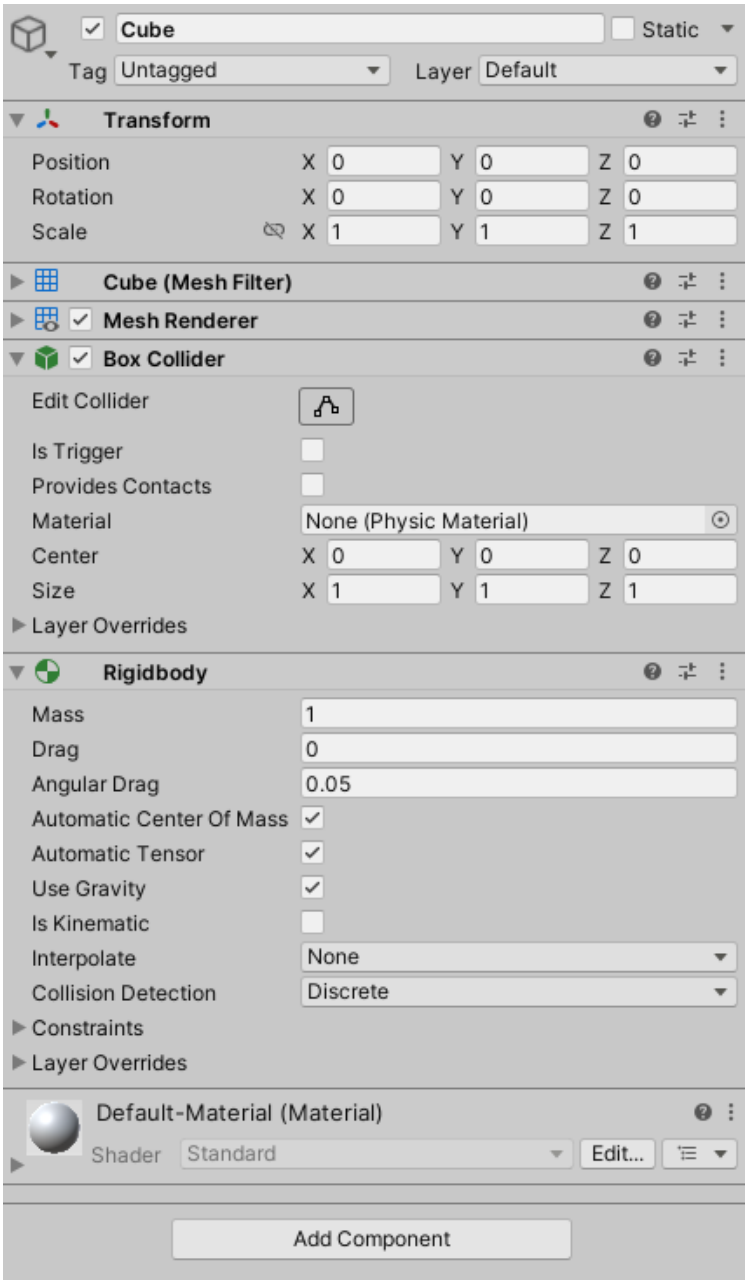

Рисунок 1.1 – Приклад ігрового об'єкта з компонентами Transform, Mesh Filter, Mesh Renderer, Box Collider та Rigidbody

Ігрові об’єкти живуть у тривимірних середовищах, які називаються сценами. Можна уявити сцену як рівень гри, але вона також може являти собою меню, титри в кінці гри або щось зовсім інше.

Поведінка ігрових об’єктів визначається функціональними блоками, які називаються компонентами. Можливо прикріпити декілька компонентів до одного об’єкту [12].

Розглянемо фундаментальні компоненти для ігрових об’єктів:

– Transform: компонент Transform визначає позицію, обертання та масштаб кожного ігрового об’єкта на сцені. Кожен ігровий об’єкт обов’язково має компонент Transform.

– Mesh Filter: цей компонент визначає форму тривимірного ігрового об’єкта.

– Mesh Renderer: цей компонент визначає те, як виглядає ігровий об’єкт, що має форму визначену компонентом Mesh Filter.

– Camera: спеціально налаштовані ігрові об’єкти, які відображають ігровий світ гравцеві.

– Light: визначає ігровий об’єкт, який є джерелом світла у ігровій сцені. Існує декілька типів освітлення: Spot (конічний), Directional (направлений), Point (точковий), Area (площинний).

– Rigidbody: Rigidbody дозволяє ігровим об’єктам взаємодіяти із системою фізики, включно із гравітацією та зіткненнями.

– Box Collider, Capsule Collider, Sphere Collider, Terrain Collider, Wheel Collider тощо: ці компоненти визначають форму тривимірного ігрового об’єкта, що використовується для прорахунку зіткнень.

1.3 Скриптинг у ігровому рушії Unity

Скрипт – це програма, яка використовується для керування, налаштування та автоматизації засобів існуючої системи (рис. 1.2).

Ігровий рушій Unity дозволяє створювати власні компоненти за допомогою скриптингу (рис. 1.3). Скрипти дозволяють запускати події у грі, змінювати властивості компонентів з плином часу та реагувати на натиснення кнопок користувачем. C# є єдиною мовою програмування, яка підтримується ігровим рушієм Unity для потреб скриптингу [12]. Більшість скриптів в ігровому рушії Unity успадковуються від класу MonoBehaviour, що дозволяє використовувати такі функції як Awake, Start, Update, FixedUpdate тощо.

Розглянемо кілька прикладів того, як можна використовувати скрипти:

– отримувати вхідні дані від гравця і переміщувати ігровий об’єкт та діяти на основі цих даних;

– встановити стани виграшу та програшу, які відкривають відповідні сцени виграшу та програшу;

– впливати на компоненти ігрового об’єкту, такі як Transform, Animation чи Renderer на основі інших змінних;

– завантажувати та зберігати ігровий прогрес гравця, наприклад кількість ігрової валюти та стан навколишнього ігрового світу.

Assets > Scripts > AreaReplicator.cs > ...

```
1   using UnityEngine;
2
    0 references
3   public class AreaReplicator : MonoBehaviour
4   {
        3 references
5       public GameObject baseArea;
        1 reference
6       public int count = 10;
        1 reference
7       public Vector3 offset = Vector3.down * 10;
8
        0 references
9       void Awake()
10      {
11          Vector3 position = baseArea.transform.position;
12          for (int i = 0; i < count; i++)
13          {
14              position += offset;
15              Instantiate(baseArea, position, baseArea.transform.rotation);
16          }
17      }
18  }
19
```

Рисунок 1.2 – Приклад скрипту AreaReplicator

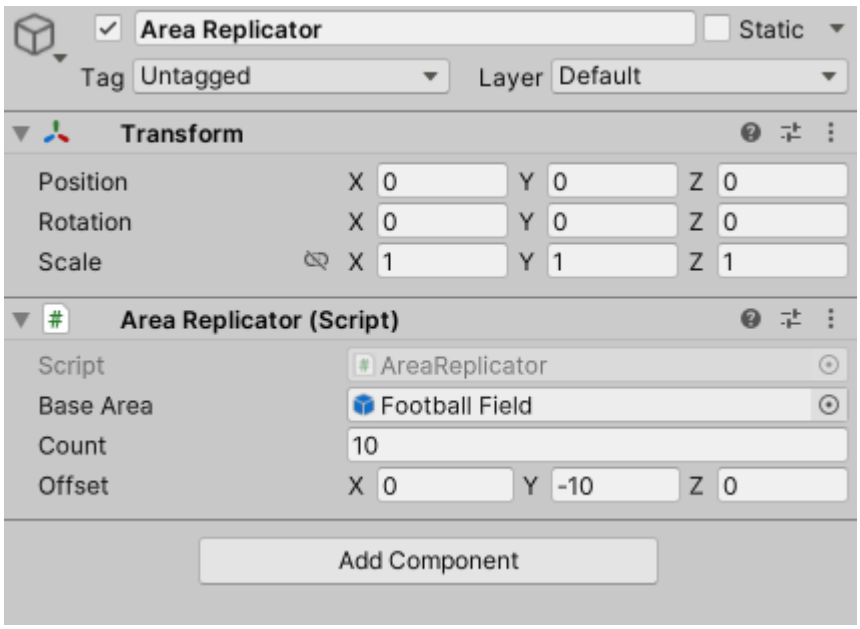

Рисунок 1.3 – Компонент AreaReplicator на основі скрипту AreaReplicator

1.4 Реалізація комп’ютерної графіки

Моделі є тривимірними репрезентаціями об’єктів. Більшість візуальних елементів для тривимірних ігор складається з моделей, таких як персонажі, інтерактивні об’єкти та світ навколо гравця (рис. 1.4, рис. 1.5).

Ігровий рушій Unity підтримує такі інструменти, як Probuilder, що дозволяють створювати тривимірні моделі прямо у вікні редактора. Втім, цей інструмент найкраще працює для прототипування, а не для кінцевого продукту.

Для тривимірних об’єктів також використовуються такі файли як текстури та матеріали – вони визначають як саме буде виглядати тривимірний об’єкт.

Тривимірні моделі зазвичай створюються у програмному забезпеченні такому як Blender, Maya чи 3ds Max. Unity використовує формат моделі .fbx, інші формати моделей також підтримуються, але Unity перетворює їх у формат .fbx після імпорту [12].

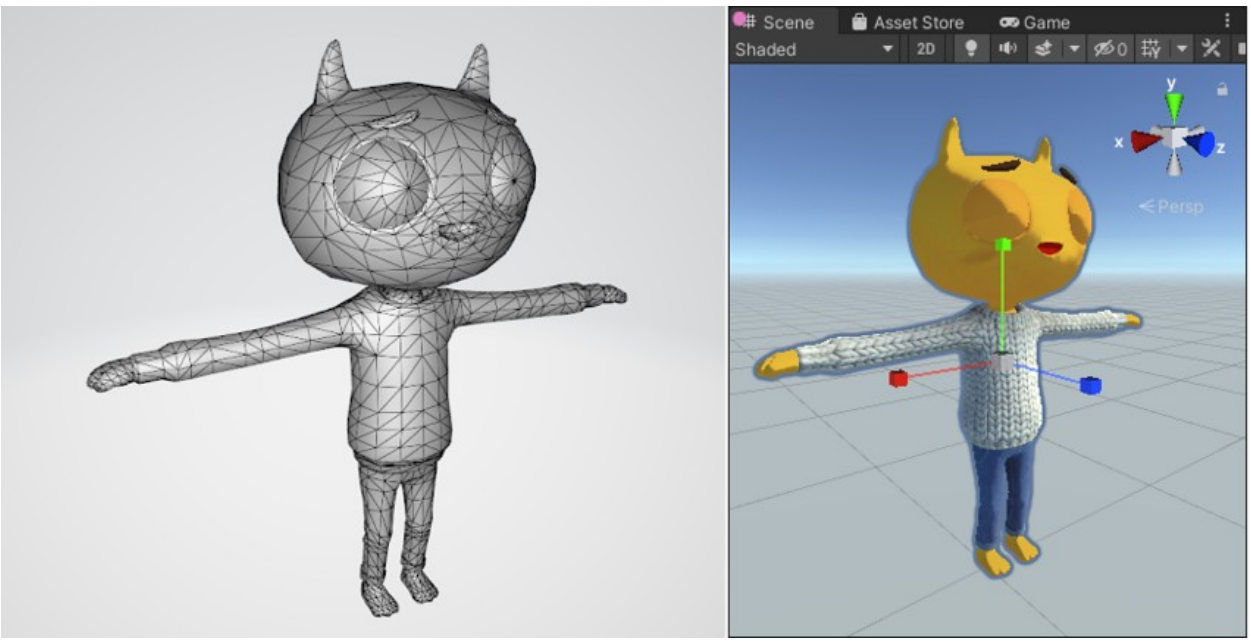

Рисунок 1.4 – Полігональна сітка ігрового персонажа та вигляд персонажа відображеного з матеріалами у вікні Unity

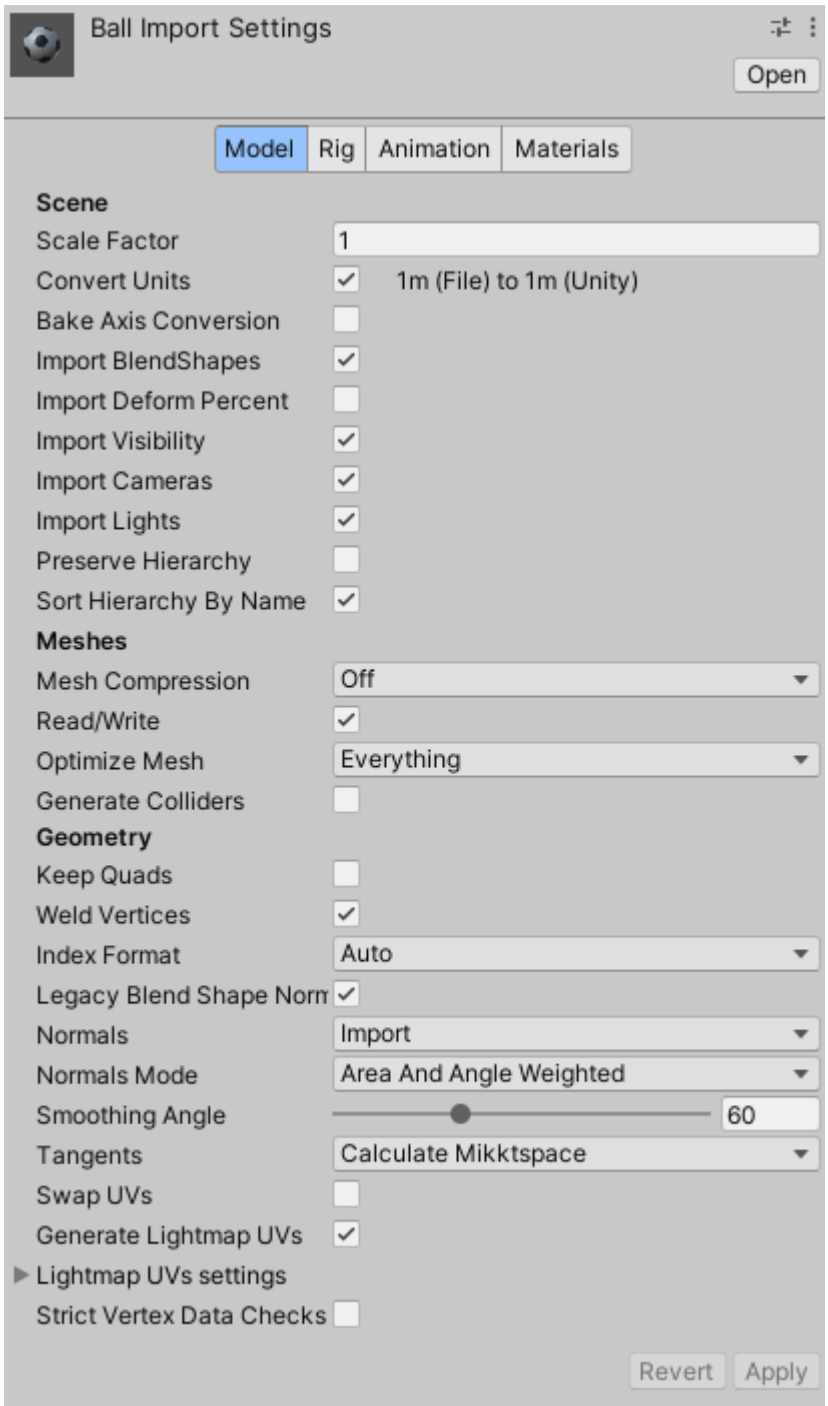

Рисунок 1.5 – Параметри імпорту тривимірної моделі

Вершинні шейдери використовуються для керування положенням і атрибутами вершин у 3D-просторі, тоді як фрагментні шейдери визначають

остаточний колір кожного пікселя на екрані. Геометричні шейдери можуть створювати нову геометрію на основі вхідних вершин, а compute шейдери використовуються для паралельних обчислювальних завдань загального призначення.

Шейдери написані на спеціалізованих мовах шейдерів, таких як GLSL (OpenGL Shading Language) або HLSL (High-Level Shading Language) і виконуються на графічному процесорі (GPU) для ефективної обробки графічних даних. Маніпулюючи шейдерами, розробники можуть досягати різних візуальних ефектів, таких як реалістичне освітлення, тіні, відображення та складні текстури, покращуючи загальну візуальну якість створених комп'ютером зображень та анімації.

Розглянемо моделі освітлення, що часто використовуються в шейдерах, тобто у комп'ютерній графіці.

Першою моделлю є ламбертове відбивання, воно розраховується за наступною формулою [13]

$$C_D = CC_L(N \cdot L\, I_L),$$

де C_D – колір дифузно відбитого світла;

C – колір текстури об'єкта;

C_L – колір світла, що падає;

L – вектор напрямку світла;

N – вектор нормалі поверхні об'єкта;

I_L – інтенсивність світла, що падає.

Зазначимо, що при $V_1\ (r_1;\ g_1;\ b_1)$, $V_2\ (r_2;\ g_2;\ b_2)$, де $r, g, b \in [0;1]$, під виразом $V = V_1 V_2$ мається на увазі $V = (r_1 r_2; g_1 g_2; b_1 b_2)$, в той час як під $V = V_1 \cdot V_2$ мається на увазі $V = |V_1||V_2|\cos(\alpha)$.

Наступною розглянемо модель відзеркалення Фонга (рис. 1.6)

$$C_D = CC_L max(0, N \cdot L) I_L + C_L C_S max(0, \hat{R} \cdot V)^P,$$

$$\hat{R}=2(N\cdot L)N-L,$$

де C_S – колір відблиску;

V – вектор напрямку до спостерігача;

$\hat{R}$ – нормований напрямок, який з цієї точки поверхні буде мати абсолютно відбитий промінь світла;

P – вираженість відблиску.

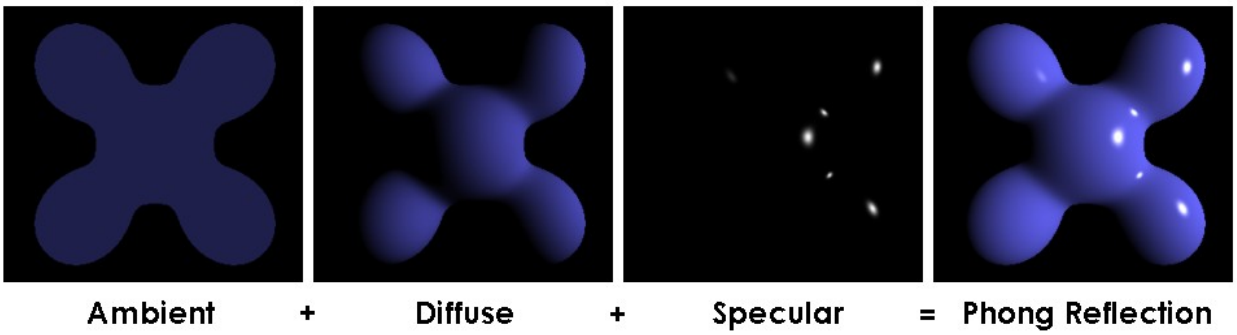

Рисунок 1.6 – Візуальна ілюстрація рівняння Фонга

Наступною розглянемо модель Блінна-Фонга, яка є більш обчислювально ефективною у порівнянні з моделлю відзеркалення Фонга

$$C_D=CC_L max(0,N\cdot L)+(C_L C_S^2 max(0,N\cdot \hat{H})^P)I_L,$$

$$\hat{H}=L+V,$$

де $\hat{H}$ – нормований напіввектор.

Починаючи з версії Unity 5.0 шейдер матеріалу за замовчуванням базується на фізичному рендерингу (PBR). Це означає, що шейдер розроблено для точної імітації взаємодії світла з матеріалами в реальному світі, що забезпечує більш реалістичну та візуально привабливу графіку. PBR шейдери в Unity дозволяють розробникам створювати матеріали, які реалістично реагують на умови освітлення, полегшуючи досягнення високоякісних візуальних ефектів у своїх проектах.

1.5 Виконання подій MonoBehaviour

Функції подій – це набір вбудованих подій, на які можуть підписуватися скрипти MonoBehaviour шляхом реалізації відповідних методів, які часто називають зворотніми викликами (callback). Зворотні виклики відповідають подіям в підсистемах ігрового рушія Unity, таким як фізика, візуалізація, введення користувача або етапам життєвого циклу скрипту [14].

На таблиці 1.1 наведено функції зворотнього виклику для скриптів MonoBehaviour.

Таблиця 1.1 – Таблиця функцій зворотнього виклику

Функція	**Опис**
1	**2**
Перше завантаження сцени	
Awake	Викликається при створенні нового екземпляру об’єкта. Завжди викликається до функції Start. Якщо об’єкт є неактивним, функція Awake не викликається до того моменту, як об’єкт стає активним.
OnEnable	Викликається коли об’єкт стає активним, викликається завжди після Awake (на тому самому об’єкті) та до функції Start.
Редактор	
Reset	Викликається для ініціалізації властивостей скрипта під час його першого приєднання до об’єкта, а також під час виконання команди Reset.
OnValidate	Викликається щоразу, коли встановлено властивості скрипта.
До першого оновлення	

Продовження таблиці 1.1

1	2
Start	Викликається до першого оновлення, тільки якщо скрипт є активним.
Оновлення	
FixedUpdate	FixedUpdate викликається з фіксованим інтервалом ігрового часу замість того, щоб викликатись кожного кадру. Оскільки ці оновлення викликаються з фіксованим інтервалом, а частота зміни кадрів є змінною – функція може бути викликана жодного разу, коли частота кадрів висока, або декілька разів, коли частота зміни кадрів низька. Усі фізичні обчислення виконуються одразу після FixedUpdate. Інтервал, через який відбуваються оновлення, визначається параметром Time.fixedDeltaTime, який можна встановити безпосередньо в скриптах або за допомогою властивості Fixed Timestep у налаштуваннях редактора.
Update	Update викликається один раз на кадр і є основною функцією для логіки що викликається один раз на кадр
LateUpdate	LateUpdate викликається один раз на кадр після завершення Update. Будь-які обчислення, виконані в Update, будуть виконані коли почнеться LateUpdate. Прикладом сценарію, в якому функція LateUpdate буде корисною, є камера від третьої особи. Якщо персонаж рухається та обертається всередині Update, можна виконувати всі обчислення руху та обертання камери в LateUpdate. Це гарантує, що персонаж повністю перемістився до того, як камера змінить своє положення.

Продовження таблиці 1.1

1	2
Візуалізація	
OnPreCull	Виклакається до того, як камера проведе процес кулінгу (culling). Кулінг визначає об’єкти, які є видимими для камери для того щоб заощадити ресурси необхідні на відображення об’єктів.
OnBecameVisible/OnBecameInvisible	Викликається коли об’єкт стає видимим/невидимим для будь-якої камери.
OnWillRenderObject	Викликається один раз для кожної камери, якщо об’єкт є видимим.
OnPreRender	Викликається до того, як камера почне відображення сцени.
OnRenderObject	Викликається коли звичайне відображення сцени є завершеним. Цю функцію можна використовувати для відображення власної геометрії.
OnPostRender	Викликається після завершення візуалізації сцени.
OnRenderImage	Викликається після завершення віізуалізації та призначено для пост-обробки (post-processing) зображення.
OnGUI	Викликається кілька разів за кадр у відповідь на події GUI.
OnDrawGizmos	Використовується для відображення Gizmos у вікні сцени реадктора.
При знищенні об’єкта	
OnDestroy	Функція OnDestroy викликається після всіх оновлень для останнього кадру існування об’єкту.
Під час виходу	

Продовження таблиці 1.1

1	2
OnApplicationQuit	Функція викликається для всіх об’єктів перед виходом із програми. У редакторі вона викликається, коли користувач зупиняє режим відтворення.
OnDisable	Функція викликається коли об’єкт чи скрипт стає вимкненим чи неактивним.

1.6 Постановка задачі

Таким чином, розроблення комп’ютерних ігор з використанням рухомих тривимірних об’єктів є сьогодні актуальним завданням у ігровій індустрії.

Об’єктом роботи є програмне моделювання дій рухомих об’єктів при створенні комп’ютерної гри у футбол.

Метою роботи є розробка комп’ютерної гри з використанням агентів штучного інтелекту, що використовують штучні нейронні мережі прямого поширення та навчаються з використанням алгоритму навчання з підкріпленням MA-POCA та механізму Self-play.

Необхідно розробити комп’ютерну гру, що імітує середовище футбольного матчу. У грі будуть представлені агенти, керовані штучними нейронними мережами прямого поширення, які будуть приймати рішення та діяти в грі. Необхідно створити реалістичний і захоплюючий ігровий процес, де агенти навчаються та покращують свою продуктивність за допомогою методів машинного навчання.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз предметної області машинного навчання;

– здійснити аналіз існуючих алгоритмів навчання з підкріпленням;

– вивчити можливості пакету ML Agents;

– розробити комп’ютерну гру у футбол із застосуванням інтелектуальних агентів;

– провести інженерію винагород для алгоритму навчання з підкріпленням;

– провести підбір гіперпараметрів для навчання моделі штучної нейронної мережі;

– провести навчання (тренування) агентів із застосуванням алгоритму навчання з підкріпленням MA-POCA та механізму Self-play;

– здійснити тестування комп’ютерної гри;

– оцінити продуктивність та ефективність керованих штучним інтелектом агентів у середовищі футбольної гри.

2 АНАЛІЗ ЗАСТОСУВАНЬ НЕЙРОННИХ МЕРЕЖ ТА ПАКЕТУ ML AGENTS

2.1 Основні поняття машинного навчання

Машинне навчання є галуззю штучного інтелекту, що зосереджується на вивченні шаблонів зі складу даних. Три основні класи алгоритмів машинного навчання включають: неконтрольоване навчання (навчання без вчителя), контрольоване навчання (навчання з вчителем) та навчання з підкріпленням. Кожен клас алгоритму навчається на різних типах даних. Розглянемо кожний із цих типів машинного навчання, а також їх приклади [15].

Неконтрольоване навчання. Метою неконтрольованого навчання є групування подібних елементів у наборі даних. Наприклад, розглянемо гравців гри. Можливо, необхідно згрупувати гравців залежно від того, наскільки вони зацікавлені в грі. Це дасть нам змогу націлюватися на різні групи (наприклад, для дуже зацікавлених гравців ми можемо запросити їх стати бета-тестерами нових функцій, тоді як для звичайних гравців ми можемо надіслати їм корисні посібники електронною поштою). Припустимо, що ми хочемо розділити наших гравців на дві групи. Спочатку ми визначимо основні атрибути гравців, такі як кількість зіграних годин, загальна кількість грошей, витрачених на покупки в програмі, та кількість пройдених рівнів. Потім можемо передати цей набір даних (три атрибути для кожного гравця) алгоритму неконтрольованого навчання, де вказується кількість груп, яка дорівнює двом. Тоді алгоритм розділить набір даних про гравців на дві групи, де гравці в кожній групі будуть схожі один на одного. Враховуючи атрибути, які були використані для опису кожного гравця, у цьому випадку результатом

буде поділ усіх гравців на дві групи, де одна група семантично представлятиме залучених гравців, а друга група семантично представлятиме незалучених гравців.

Тут не було надано конкретних прикладів того, які гравці вважаються залученими, а які – незалученими. Ми просто визначили відповідні атрибути та покладалися на алгоритм, щоб розкрити дві групи самостійно. Цей тип набору даних зазвичай називають набором даних без міток, оскільки в ньому відсутні прямі мітки. Отже, неконтрольоване навчання може бути корисним у ситуаціях, коли ці мітки можуть бути дорогими або складними для визначення.

Контрольоване навчання. Під час контрольованого навчання подібні предмети не просто групуються, а безпосередньо вивчається відображення кожного елемента в групі (або класі), до якого він належить. Повертаючись до попереднього прикладу кластеризації гравців, скажімо, тепер необхідно передбачити, хто з гравців збирається закинути гру (тобто припинити грати в гру на наступні 30 днів). Тут можна переглянути історичні записи та створити набір даних, який містить атрибути гравців на додаток до мітки, яка вказує на те, закинули вони гру чи ні. Зауважимо, що атрибути гравців, які було використано для цього завдання прогнозування відтоку, можуть відрізнятися від тих, які було використано для попереднього завдання кластеризації. Потім можна передати цей набір даних (атрибути та мітку для кожного гравця) в контрольований алгоритм навчання, який вивчатиме зіставлення атрибутів гравця з міткою, яка вказуватиме, чи буде цей гравець закидати гру чи ні. Інтуїція полягає в тому, що контрольований алгоритм навчання дізнається, які значення цих атрибутів зазвичай відповідають гравцям, які закинули гру, а які не закинули (наприклад, він може дізнатися, що гравці, які витрачають дуже мало і грають протягом дуже короткого часу, швидше за все закинуть гру). Тепер, маючи цю вивчену модель, можна надати їй атрибути нового гравця (того, який нещодавно почав грати в гру), і вона виведе передбачувану мітку

для цього гравця. Це передбачення – це очікування алгоритмів щодо того, чи закине гравець гру, чи ні. Тепер можна використовувати ці прогнози, щоб націлитися на гравців, які, як очікується, закинуть гру, і спонукати їх продовжувати грати в гру.

Як для неконтрольованого навчання, так і контрольованого, необхідно виконати два завдання: вибір атрибутів і вибір моделі. Вибір атрибутів стосується вибору того, як ми хочемо представити зацікавлену сутність, у даному випадку гравця. Вибір моделі, з іншого боку, стосується вибору алгоритму (і його параметрів), який добре виконує завдання. Обидва ці завдання є активними областями дослідження машинного навчання і, на практиці, вимагають кількох ітерацій для досягнення хороших результатів.

Навчання з підкріпленням. Навчання з підкріпленням можна розглядати як форму навчання для послідовного прийняття рішень, яка зазвичай асоціюється з керуванням роботами (але насправді є набагато загальнішою). Розглянемо автономного робота-пожежника, якому доручено переміщатися в територію, знаходити вогонь і нейтралізувати його. У будь-який момент робот сприймає навколишнє середовище за допомогою своїх датчиків (наприклад, камери, тепла, дотику), обробляє цю інформацію та виконує дію (наприклад, рух ліворуч, обертання шланга з водою, увімкнення води). Іншими словами, він постійно приймає рішення про те, як взаємодіяти в цьому середовищі, враховуючи його погляд на світ (тобто дані з датчиків) і мету (тобто нейтралізацію вогню). Навчити робота бути успішною пожежною машиною – це саме те, для чого покликане навчання з підкріпленням.

Мета навчання з підкріпленням полягає в тому, щоб вивчити політику, яка, по суті, є відображенням спостережень до дій. Спостереження – це те, що робот може виміряти з навколишнього середовища (у даному випадку, усі його сенсорні дані), а дія, у своїй найбільш грубій формі, – це зміна конфігурації робота (наприклад, його положення, положення його водяного шлангу і чи увімкнений водяний шланг чи вимкнений).

Останнім фрагментом навчального завдання з підкріпленням є сигнал винагороди. Робот навчений вивчати політику, яка максимізує його загальну винагороду. Навчаючи робота бути вогнегасником, ми надаємо йому винагороди (позитивні та негативні), які вказують на те, наскільки добре він виконує завдання. Робот не вміє гасити пожежі до того, як його навчать. Він вивчає завдання, оскільки отримує велику позитивну винагороду, коли гасить вогонь, і невелику негативну винагороду кожну секунду. Той факт, що винагороди є рідкісними (тобто можуть надаватися не на кожному кроці, а лише тоді, коли робот досягає успіху чи невдачі), є визначальною характеристикою навчання з підкріпленням і саме тому вивчення хорошої політики може бути складним та трудомістким для складних середовищ.

Вивчення політики зазвичай вимагає багатьох випробувань і ітераційних оновлень політики. Робот потрапляє в кілька ситуацій пожежі і з часом вивчає оптимальну політику, яка дозволяє йому ефективніше гасити пожежі. Ми не можемо розраховувати на повторне навчання робота в реальному світі, особливо коли йдеться про пожежі. Саме тому використання Unity як симулятора є ідеальним навчальним майданчиком для вивчення такої поведінки. Фактично, у багатьох відношеннях можна розглядати неігрового персонажа (NPC) як віртуального робота з власними спостереженнями за навколишнім середовищем, власним набором дій і конкретною метою. Таким чином, природно досліджувати те, як ми можемо тренувати поведінку агента в Unity за допомогою навчання з підкріпленням.

Подібно до неконтрольованого та контрольованого навчання, навчання з підкріпленням також включає два завдання: вибір атрибутів і вибір моделі [27-35]. Вибір атрибутів визначає набір спостережень для агента, які найкраще допомагають йому досягти мети, тоді як вибір моделі визначає форму політики (перетворення спостережень на дії) та її параметрів. На практиці навчання поведінці є ітеративним процесом, який може вимагати зміни атрибутів і вибору моделі [15].

2.2 Використання штучних нейронних мереж для навчання агентів

Нейронні мережі є фундаментальною концепцією штучного інтелекту та машинного навчання, натхненні структурою та функціями людського мозку, що складаються з взаємопов’язаних вузлів або нейронів, які працюють разом, щоб обробляти та аналізувати дані. Штучні нейронні мережі складаються з взаємопов’язаних вузлів або нейронів, організованих шарами (рис. 2.1). Вхідний рівень отримує дані, які потім обробляються через приховані шари з використанням вагових коефіцієнтів і зміщень, і, нарешті, виробляє вихідні дані у вигляді прогнозів або класифікацій.

Основним будівельним блоком нейронної мережі є нейрон, який приймає вхідні сигнали, застосовує до них вагові коефіцієнти, підсумовує їх зі зміщеннями та пропускає результат через функцію активації для отримання виходу. Цей процес повторюється на кількох рівнях, при цьому кожен рівень вивчає дедалі складніші характеристики даних.

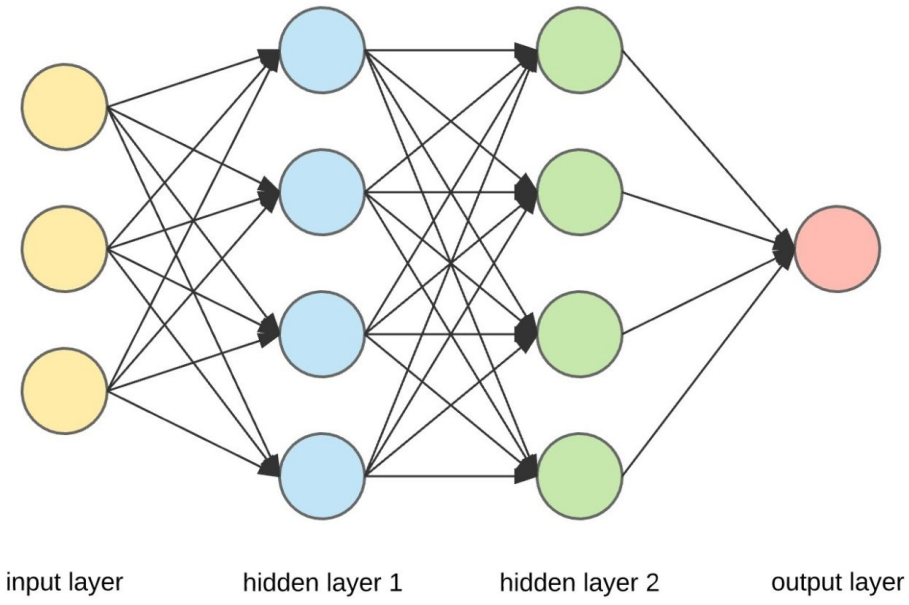

Рисунок 2.1 – Приклад структури нейронної мережі прямого поширення

Найпростішим прикладом штучної нейронної мережі є лінійний класифікатор. Наведемо функцію обчислення значень нейронів на виході, до обробки цих значень функцією активації. Наведена формула є справедливою для обчислення кожного наступного шару в нейронній мережі прямого поширення

$$f(x, W) = Wx + b,$$

де W – матриця вагових коефіцієнтів (параметрів) розмірністю $j \times i$ (i – кількість нейронів на вході, j – кількість нейронів на виході);

x – вектор значень вхідних нейронів розмірністю $i \times 1$;

b – вектор зміщення розмірністю $j \times 1$.

Одним із ключових компонентів нейронних мереж є функція активації, яка визначає вихід нейрона на основі його вхідних даних. Функції активації вводять нелінійність у мережу, дозволяючи їй вивчати складні моделі та зв’язки в даних. Загальні функції активації включають сигмоїдну функцію, функцію tanh, функцію ReLU (Rectified Linear Unit) і функцію softmax.

Сигмоїдна функція, також відома як логістична функція, здавлює вхідні значення в діапазон від 0 до 1, що робить її придатною для завдань двійкової класифікації. Однак ця функція страждає від проблеми зникнення градієнта, коли градієнти стають дуже малими для екстремальних вхідних значень, що призводить до повільного навчання

$$g(x) = \frac{1}{1 + e^{-(Wx+b)}}.$$

Функція tanh подібна до сигмоїдної функції, але зміщує вхідні значення в діапазоні від -1 до 1. Їй часто надають перевагу над сигмоїдною функцією, оскільки вона має вихід з центром на нуль, що сприяє швидшій конвергенції під час навчання

$$g(x)=\tanh(Wx+b).$$

ReLU (Rectified Linear Unit) – популярна функція активації, яка встановлює всі від'ємні вхідні значення на нуль і передає додатні значення без змін. Функція ReLU є обчислювально ефективною і допомагає пом'якшити проблему зникнення градієнта, що призводить до швидшого навчання глибоких нейронних мереж

$$g(x)=max(0,Wx+b).$$

Функція softmax зазвичай використовується на вихідному шарі нейронної мережі для завдань багатокласової класифікації. Вона нормалізує вихідні значення у розподіл ймовірностей, де кожне значення представляє ймовірність належності вхідних даних до певного класу

$$g(x)_i=\frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}.$$

Таким чином, функції активації відіграють вирішальну роль у нейронних мережах, вводячи нелінійність і дозволяючи мережі вивчати складні моделі в даних. Вибір правильної функції активації має важливе значення для продуктивності та ефективності нейронної мережі в різних завданнях, таких як класифікація, регресія та розпізнавання зображень [36-51].

Нейронні мережі навчаються за допомогою процесу, який називається зворотним розповсюдженням, коли модель коригує свої ваги та зміщення на основі помилки між прогнозованим результатом і фактичним результатом. Цей ітеративний процес допомагає нейронній мережі навчатися на основі даних і з часом покращувати свою продуктивність.

Існують різні типи нейронних мереж, наприклад нейронні мережі прямого поширення, рекурентні нейронні мережі та згорткові нейронні мережі, кожна з яких підходить для конкретних завдань. Нейронні мережі прямого поширення є найпростішою формою, де дані протікають в одному напрямку від входу до виходу. Рекурентні нейронні мережі мають зв’язки, які утворюють петлі, що дозволяє їм отримувати послідовну інформацію в даних. Згорткові нейронні мережі призначені для обробки сітчастих даних, наприклад зображень, шляхом застосування згорткових фільтрів для виділення ознак [30-42].

Нейронні мережі продемонстрували надзвичайний успіх у різних сферах, включаючи розпізнавання зображень і мови, обробку природної мови та автономне водіння. Вони мають здатність вивчати складні закономірності та взаємозв’язки в даних, що робить їх потужними інструментами для вирішення широкого кола проблем у різних сферах.

Незважаючи на свою ефективність, нейронні мережі також мають проблеми, такі як перенавчання, зникнення градієнтів і складність обчислень. Дослідники постійно працюють над розробкою нових методів і архітектур для вирішення цих проблем і підвищення продуктивності нейронних мереж у реальних програмах.

Підсумовуючи, нейронні мережі є основоположною концепцією штучного інтелекту та машинного навчання, яка імітує поведінку людського мозку для обробки та аналізу даних. Вони зробили революцію в галузі штучного інтелекту та продовжують розвивати технології та інновації.

2.3 Алгоритм MA-POCA

MA-POCA (Multi-Agent Posthumous Credit Assignment) є алгоритмом машинного навчання з підкріпленням. Цей алгоритм створений підлаштовувати процес тренування під змінну кількість агентів у сцені під час тренувального епізоду, тобто під створення та видалення агентів, використовуючи для цього механізм уваги, що вигідним чином виділяє його на фоні інших, більш старих, MARL (Multi-Agent Reinforcement Learning) алгоритмів [16].

Налаштуванням, яке ми розглядаємо, є децентралізованим POMDP (Partially observable Markov decision process) [17], яке визначається (N, S, O, A, P, r, γ), де $N \geq 1$ – кількість агентів, а S – простір станів середовища. O – спільний простір спостереження всіх агентів $O = O^1 \times \ldots \times O^N$, де O^i – простір спостереження агента з номером i. У момент часу t середовище перебуває в стані $s_t \in S$, a $o^i_t \in O^i$ є локальним спостереженням агента i, яке корелює з s_t. Стан середовища може містити інформацію, недоступну локально жодному агенту, наприклад загальну кількість агентів, які діють наразі. A – простір спільних дій усіх агентів $A = A^1 \times \ldots \times A^N$, де A^i – простір дій агента i. Зауважимо, простори спостереження та дії для різних агентів не обов'язково повинні бути рівними. Крім того, ми використовуємо жирні вектори для представлення спільних величин над агентами, наприклад, спільна дія $\boldsymbol{a} = (a^1, \ldots, a^N)$ або спільне спостереження $\boldsymbol{o} = (o^1, \ldots, o^N)$, де $\boldsymbol{a} \in A$ і $\boldsymbol{o} \in O$.

$P : S \times A \times S \rightarrow [0, 1]$ – функція переходу, де $P(s'|s, \boldsymbol{a})$ – ймовірність того, що середовище переходить у стан s', враховуючи поточний стан s і спільну дію $\boldsymbol{a} \in A$. $r : S \times A \rightarrow R$ – спільна функція винагороди, де $r(s, \boldsymbol{a})$ – винагорода, отримана всіма агентами, коли виконується спільна дія $\boldsymbol{a} \in A$, а середовище перебуває в стані $s \in S$.

Централізоване навчання, децентралізоване виконання. Розглянемо структуру навчання незалежного актора з централізованим критиком (IACC) [18], де критик, навчений спільному інформуванню, використовується для оновлення набору незалежних акторів в архітектурі актор-критик. Незалежний актор-критик (IAC), який навчає незалежного критика та політику для кожного агента, використовуючи лише локальну інформацію, і спільний актор-критик (JAC), який навчає одну спільну політику та спільного критика, є конкуруючими підходами. Загалом IAC погано виконує завдання, які вимагають значної координації через часткову спостережуваність при використанні лише локальних спостережень. Крім того, JAC непрактичний у сценаріях реального світу, оскільки спільній політиці потрібен доступ до всіх спостережень агента одночасно для генерування дій, по суті припускаючи ідеальний зв'язок між агентами та вузлом політики.

Нехай π_i, $1 \leq i \leq N$ представляє політику кожного незалежного агента. Враховуючи стан середовища s_t і відповідне спільне спостереження $\boldsymbol{o_t}$ та дію $\boldsymbol{a_t}$, спільну політику $\boldsymbol{\pi(a_t|o_t)}$ можна розкласти як $\boldsymbol{\pi}(\boldsymbol{a_t}|\boldsymbol{o_t})=\prod_i \pi_i(a_t^i|o_t^i)$, оскільки агенти діють незалежно від локальних спостережень.

Централізована функція цінності стану для стану s_t визначається як

$$V^{\pi}(s_t)=E_{\boldsymbol{\pi}}[\sum_{l=0}^{\infty}\gamma^l r(s_{t+l},\boldsymbol{a_{t+l}})],$$

і централізована функція цінності стану-дій визначається як

$$Q^{\pi}(s_t,\boldsymbol{a_t})=r(s_t,\boldsymbol{a_t})+E_{\pi}[\sum_{l=0}^{\infty}\gamma^l r(s_{t+l},\boldsymbol{a_{t+l}})].$$

Функція цінності MA-POCA. Обговоримо як оцінити очікуваний дисконтований прибуток, наведений для групи агентів, де деякі агенти можуть припинити роботу раніше. Нагадаємо, що в наших умовах кількість активних агентів залежить від t. Таким чином, нехай k_t позначає кількість

активних агентів на кроці часу t так, що $1 \leq k_t \leq N$, де N є максимальною кількістю агентів, які можуть бути живими в будь-який момент.

У літературі про MARL, як правило, існує два способи, як централізований стан або функції цінності стану дії залежать від стану:

– існує окремий вектор, що містить глобальну інформацію (тобто координати всіх агентів), яка не спостерігається жодним окремим агентом [19, 20];

– використовується спільне спостереження агентів $o_t \in O$, оскільки це розумне наближення до глобального стану [21, 22].

Також можливе використання гібрида. Ми розглядаємо лише спільне спостереження активних агентами, хоча перше все одно вимагатиме поглинаючих станів або значень у певній якості та може розглядатися засобами, подібними до того, що наведено нижче.

Щоб обробляти різну кількість агентів за крок, спочатку кодуються спостереження всіх активних агентів $g_i(o_t^i)_{1 \leq i \leq k_t}$, а потім пропускається кодування через RSA. Блок RSA, який використовується, архітектурно подібний до тих, що використовуються в архітектурі звичайного трансформера [23], але без позиційного кодування [24]. Тоді функція значення централізованого стану, параметризована ϕ, має вигляд

$$V_\phi(RSA(g_i(o_t^i)_{1 \leq i \leq k_t})),$$

і навчається з *TD(λ)*

$$J(\phi) = (V_\phi(RSA(g_i(o_t^i)_{1 \leq i \leq k_t})) - y^{(\lambda)})^2,$$

де

$$y^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)},$$

$$G_t^{(n)} = \sum_{l=1}^{n} \gamma^{l-1} r_{t+l} + \gamma^n V_\phi\left(RSA\left(g\left(o_{t+n}^i\right)_{1 \le i \le k_{t+n}}\right)\right),$$

де k_{t+n} — кількість агентів, активних у момент часу $t + n$.

Зауважимо, що k_{t+n} може бути більшим або меншим за k_t, оскільки будь-яка кількість агентів могла припинити роботу раніше або бути породжена на кроці часу t. Саме таким чином очікуване значення від часу $t + n$ може поширюватися до агента, який припинив роботу в момент часу t.

3 РЕЗУЛЬТАТИ ПРОГРАМНОГО МОДЕЛЮВАННЯ

3.1 Вибір програмного забезпечення

Для розробки програмного засобу кваліфікаційної роботи було використано ігровий рушій Unity версії 2022.3 з пакетом ML Agents версії 2.0.1, середовище програмування Visual Studio Code.

Для тренування моделей нейронних мереж використовується середовище Anaconda з пакетом mlagents.

Особливості рушія Unity для даної роботи полягають у наступному:

– реалістичне моделювання фізики: рушій Unity надає просунуті можливості моделювання фізики, що може точно відтворювати рухи та взаємодію гравців, м’яча та інших елементів у грі;

– інтеграція штучного інтелекту: за допомогою пакета ML Agents агенти у грі можуть бути оснащені штучними нейронними мережами, щоб приймати розумні рішення на основі вхідних даних та вчитися на власному досвіді під час навчання;

– підтримка багатоплатформеності: рушій Unity підтримує декілька платформ, що дозволяє розгортати гру на різних пристроях, включаючи ПК, мобільні пристрої та консолі;

– легка інтеграція ресурсів: рушій Unity забезпечує зручний інтерфейс для імпорту ресурсів, таких як моделі гравців, анімація та звукові ефекти, що полегшує створення візально привабливого та захплюючого ігрового середовища;

– простота використання: рушій Unity відомий своєю простотою використання , що робить його популярним вибором серед розробників. Зручний інтерфейс та інтуїтивно зрозумілий дизайн Unity дозволяють

розробникам швидко створювати прототипи ігор без потреби в наявності глибоких знань.

В цілому, поєднання рушія Unity та пакета ML Agents пропонує потужний набір інструментів для створення реалістичної та захоплюючої гри у футбол за допомогою інтелектуальних агентів.

3.2 Особливості проєктування

Класи в розробленому програмному проєкті кваліфікаційної роботи було вирішено розділити на 4 простори імен: Game, Game.Difficulty, Game.Player, Game.UI. До простору імен Game.Difficulty було віднесено класи, пов’язані із складністю гри; до простору імен Game.Player було віднесено класи, пов’язані із гравцем та його діями; до простору імен Game.UI було віднесено класи, пов’язані із користувацьким інтерфейсом гри; усі інші, більш базові, класи було віднесено до простору імен Game.

Проєкт було спроєктовано у такий спосіб, що дозволяє проводити симуляцію футбольного матчу без функціоналу, пов’язаного із діями гравця – це дозволяє проводити навчання агентів без можливості втручання з боку гравця у середовищі майже ідентичному ігровому.

Наведемо і проаналізуємо UML діаграму класів проєкту (рис. 3.1).

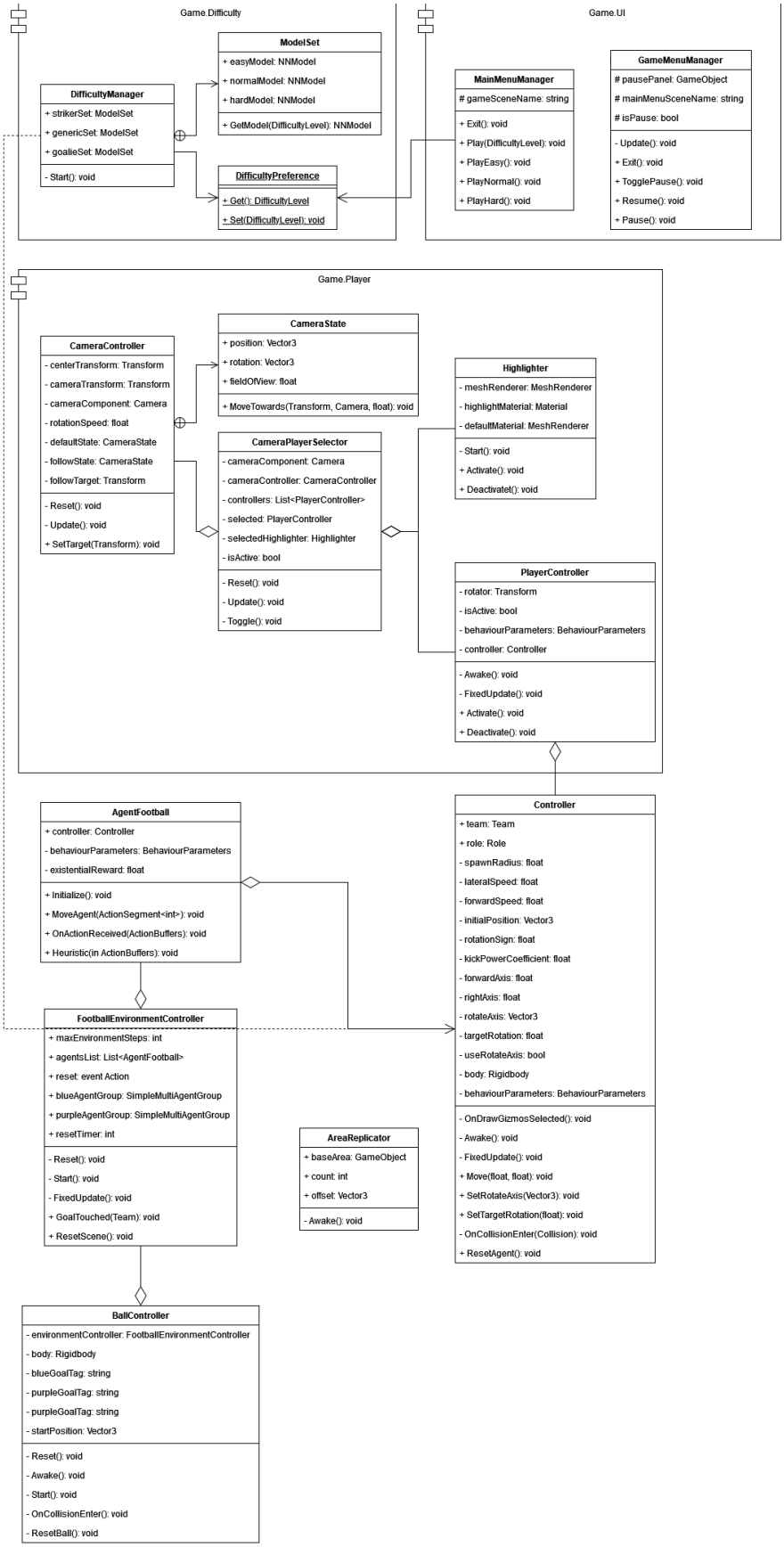

Рисунок 3.1 – UML діаграма класів

Зазначимо, що класи DifficultyManager, CameraController, CameraPlayerSelector, Highlighter, PlayerController, GameMenuManager, MainMenuManager, AreaReplicator, BallController, Controller, FootballEnvironmentController унаслідуються від класу MonoBehaviour, тобто є повноцінними компонентами ігрових об’єктів; клас AgentFootball унаслідується від класу Agent, що в свою чергу унаслідується від MonoBehaviour. Наслідування від цих класів не було позначено на діаграмі у зв’язку зі спрощенням діаграми.

Наведемо таблицю 3.1 опису методів та полів класів.

Таблиця 3.1 – Опис методів та полів класів

Назва	**Опис**
1	**2**
ModelSet	
public NNModel easyModel;	Модель нейронної мережі для легкого рівня складності
public NNModel normalModel;	Модель нейронної мережі для нормального рівня складності
public NNModel hardModel;	Модель нейронної мережі для високого рівня складності
public NNModel GetModel(DifficultyLevel difficulty)	Повертає модель нейронної мережі для даного рівня складності
DifficultyManager	
public ModelSet strikerSet;	Набір моделей різного рівня складності для ролі нападаючого
public ModelSet genericSet;	Набір моделей різного рівня складності для звичайної ролі
public ModelSet goalieSet;	Набір моделей різного рівня складності для ролі воротаря

Продовження таблиці 3.1

1	2
void Start()	Викликається до першого оновлення та встановлює кожному агенту модель нейронної мережі відповідно до збереженого рівня складності
DifficultyPreference	
public static DifficultyLevel Get()	Повертає збережений рівень складності
public static void Set(DifficultyLevel difficulty)	Зберігає даний рівень складності<
CameraState	
public Vector3 position;	Позиція стану
public Vector3 rotation;	Обертання стану
public float fieldOfView;	Поле зору стану
public void MoveTowards(Transform transform, Camera camera, float deltaTime)	Наближає стан даних компонентів до стану, визначеного у цьому об'єкті, згідно даного часу deltaTime
CameraController	
private Transform centerTransform;	Компонент Transform центру обертання камери
private Transform cameraTransform;	Компонент Transform камери
private Camera cameraComponent;	Компонент Camera камери
private float rotationSpeed;	Швидкість обертання
private CameraState defaultState;	Звичайний стан камери
private CameraState followState;	Стан стеження за гравцем
private Transform followTarget;	Ціль для стеження
void Reset()	Викликається для ініціалізації властивостей у редакторі
void Update()	Викликається кожного кадру та оновлює стан компонентів об'єкта відповідно до наявності чи відсутності цілі для стеження
public void SetTarget(Transform target)	Задає ціль для стеження

Продовження таблиці 3.1

1	2
CameraPlayerSelector	
private Camera cameraComponent;	Компонент Camera
private CameraController cameraController;	Компонент CameraController
private List<PlayerController> controllers;	Список об'єктів із компонентом PlayerController
private PlayerController selected;	Найближчий до курсору PlayerController
private Highlighter selectedHighlighter;	Компонент Highlighter найближчого до курсору PlayerController
private bool isActive;	Визначає активність режиму прямого керування
void Reset()	Викликається для ініціалізації властивостей у редакторі
void Update()	Виклакається кожного кадру, підсвічує найближчий до курсору PlayerController та переключає режим прямого керування за необхідності
void Toggle()	Переключає режим прямого керування
Highlighter	
private MeshRenderer meshRenderer;	Компонент MeshRenderer
private Material highlightMaterial;	Матеріал, що використовується при підсвічуванні
private Material defaultMaterial;	Звичайний матеріал об'єкта
void Start()	Викликається до першого оновлення та встановлює звичайний матеріал
public void Activate()	Активує підсвічування
public void Deactivate()	Деактивує підсвічування
PlayerController	
private Transform rotator;	Компонент Transform об'єкта, що визначає обертання агенту під контролем гравця

Продовження таблиці 3.1

1	2
private bool isActive;	Визначає чи є активним режим прямого керування над агентом
private BehaviorParameters behaviorParameters;	Компонент BehaviorParameters цього об'єкта
private Controller controller;	Компонент Controller цього об'єкта
void Awake()	Викликається при створенні об'єкта та встановлює компоненти
void FixedUpdate()	Викликається при оновленні фізики та передає введення користувача, якщо режим прямого керування є активним
public void Activate()	Активує режим прямого керування
public void Deactivate()	Деактивує режим прямого керування
GameMenuManager	
protected GameObject pausePanel;	Панель паузи
protected string mainMenuSceneName;	Назва сцени головного меню
protected bool isPause;	Визначає, чи активована в даний момент пауза
void Update()	Викликається кожного кадру та переключає режим паузи за необхідності
public void Exit()	Виходить з поточної сцени до сцени головного меню
public void TogglePause()	Переключає режим паузи
public void Resume()	Відновлює гру
public void Pause()	Призупиняє гру
MainMenuManager	
protected string gameSceneName;	Назва ігрової сцени
public void Exit()	Вихід з гри
public void Play(DifficultyLevel difficulty)	Розпочинає гру з даним рівнем складності

Продовження таблиці 3.1

1	2
public void PlayEasy()	Розпочинає гру на легкому рівні складності
public void PlayNormal()	Розпочинає гру на звичайному рівні складності
public void PlayHard()	Розпочинає гру на високому рівні складності
AgentFootball	
public Controller controller;	Компонент Controller цього об'єкта
private BehaviorParameters behaviorParameters;	Компонент BehaviorParameters цього об'єкта
private float existentialReward;	Розмір екзистенційної виногороди за один крок симуляції
public override void Initialize()	Викликається для ініціалізації класу
public void MoveAgent(ActionSegment<int> actions)	Переміщує та обертає агента відповідно до сегменту дій
public override void OnActionReceived(ActionBuffers actionBuffers)	Викликається при отриманні дії від нейронної мережі
ublic override void Heuristic(in ActionBuffers actionsOut)	Визначає дії в Heuristic режимі виконання
AreaReplicator	
public GameObject baseArea;	Об'єкт для реплікації
public int count;	Кількість об'єктів для створення
public Vector3 offset;	Зміщення при створенні об'єкта
void Awake()	Викликається при створенні об'єкта та реплікує обраний об'єкт
BallController	
private FootballEnvironmentController environmentController;	Компонент FootballEnvironmentController іншого об'єкта
private Rigidbody body;	Компонент Rigidbody цього об'єкта
private string blueGoalTag;	Тег воріт першої команди

Продовження таблиці 3.1

1	2
private string purpleGoalTag;	Тег воріт другої команди
private Vector3 startPosition;	Початкова позиція м'яча
void Reset()	Викликається для ініціалізації властивостей у редакторі
void Awake()	Викликається при створенні об'єкта та підписує його на подію початку нового раунду
void Start()	Викликається до першого оновлення
void OnCollisionEnter(Collision collision)	Викликається під час початку зіткнення із іншим об'єктом та реагує на зіткнення з воротами
void ResetBall()	Переініціалізує м'яч
Controller	
public Team team;	Команда агента
public Role role;	Роль агента
private float spawnRadius;	Радіус появи
private float lateralSpeed;	Швидкість руху по сторонам
private float forwardSpeed;	Швидкість руху
private Vector3 initialPosition;	Початкова позиція
private float rotationSign;	Визначає напрямок обертання: звичайний чи протилежний
private float kickPowerCoefficient;	Сила удару по м'ячу
private float forwardAxis;	Визначає напрямок руху
private float rightAxis;	Визначає напрямок руху по сторонам
private Vector3 rotateAxis;	Визначає вісь обертання
private float targetRotation;	Визначає бажаний кут обертання
private bool useRotateAxis;	Визначає метод, що використовується для обертання агента
private Rigidbody body;	Компонент Rigidbody цього об'єкта
private BehaviorParameters behaviorParameters;	Компонент BehaviorParameters цього об'єкта

Продовження таблиці 3.1

1	2
void OnDrawGizmosSelected()	Зображує сферу з радіусом появи агента у вікні сцени редактору, якщо виділено цей об'єкт
void Awake()	Викликається при створенні об'єкта та ініціалізує цей клас
void FixedUpdate()	Викликається при оновленні фізика, переміщує та обертає агента
public void Move(float forward, float right)	Задає напрямок руху
public void SetRotateAxis(Vector3 value)	Задає вісь обертання
public void SetTargetRotation(float value)	Задає бажаний кут обертання
void OnCollisionEnter(Collision collision)	Викликається під час зіткнення із іншим об'єктом та містить логіку взаємодії із м'ячем
public void ResetAgent()	Переініціалізує агента
FootballEnvironmentController	
public int maxEnvironmentSteps;	Максимальна кількість кроків симуляції
public List<AgentFootball> agentsList;	Список агентів
public event Action reset;	Подія перезапуску раунду
private SimpleMultiAgentGroup blueAgentGroup;	Група агентів першої команди
private SimpleMultiAgentGroup purpleAgentGroup;	Група агентів другої команди
private int resetTimer;	Лічильник кроків симуляції
void Reset()	Викликається для ініціалізації властивостей у редакторі
void Start()	Викликається до першого оновлення та ініціалізації цього класу

Продовження таблиці 3.1

1	2
void FixedUpdate()	Викликається під час оновлення фізики та перезапускає раунд при перевищенні максимального числа кроків симуляції
public void GoalTouched(Team scoredTeam)	Викликається при забитті голу та нараховує відповідні винагороди командам під час навчання
public void ResetScene()	Перезапускає раунд

3.3 Розробка користувацького інтерфейсу

Для розробки UI було використано інструменти системи Unity UI, що також відома як uGUI [25]. Усі елементи користувацького інтерфейсу для обох сцен (головне меню та ігрова сцена) було розміщено усередині об’єкту із компонентом Canvas (рис. 3.2). Компонент Canvas було налаштовано під роздільну здатність екрану шириною 1920 та висотою 1080 пікселів, такий вибір пояснюється тим, що роздільна здатність 1920x1080 пікселів є найбільш популярною на час створення кваліфікаційної роботи – в свою чергу це допомагає легше підтримувати оптимальний вигляд користувацького інтерфейсу.

Для можливості швидкої зміни оформлення інтерфейсу (UI Theming) було використано систему префабів. У контексті ігрового рушія Unity префабом є попередньо створений об’єкт або група об’єктів, які можна легко повторно використовувати в ігровому проекті. Префаби в Unity дозволяють розробникам створювати шаблон ігрового об’єкта з вже налаштованими його компонентами та скриптами, що полегшує створення екземплярів одного об’єкта в різних частинах гри. Це допомагає оптимізувати розробку ігор,

дозволяючи розробникам швидко й ефективно створювати ігрові об’єкти та керувати ними, не створюючи їх з нуля кожного разу, коли вони потрібні.

Рисунок 3.2 – Компоненти об'єкту Canvas

Так було створено префаби для фону (Background); другого фону (Background2), що є варіантом префабу Background, тобто зберігає більшість змін у властивостях батьківського префабу; кнопки (Button) та тексту (Text). Таким чином було досягнуто можливості швидко змінювати кольори та шрифти у користувацькому інтерфейсі (рис. 3.3).

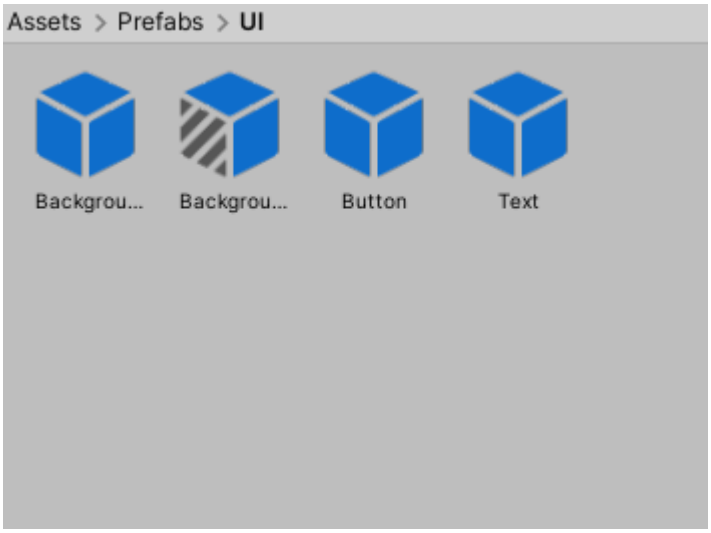

Рисунок 3.3 – Префаби користувацького інтерфейсу

3.4 Аналіз результатів тестування

Тест-кейси містять наступні поля: ID, опис, передумови, кроки тестування, очікуваний результат та фактичний результат. ID є унікальним ідентифікатором, призначеним кожному тест-кейсу для цілей відстеження та посилання. Опис визначає ціль або завдання тест-кейсу, описуючи, який конкретний аспект програмного забезпечення чи системи тестується. Передумови стосуються необхідних умов, які повинні бути виконані перед виконанням тесту, наприклад налаштування середовища або вхідних даних.

Кроки тестування – це детальні інструкції або дії, які необхідно здійснити для виконання тесту; це включає в себе конкретні вхідні дані, взаємодії або операції, які повинен виконати тестувальник. Очікуваний результат описує потрібний наслідок або поведінку, яка очікується після успішного виконання тесту. Фактичний результат, з іншого боку – це фактичний результат або поведінка, які спостерігаються під час виконання тесту; це порівнюється з очікуваним результатом, щоб визначити, чи пройдено тести чи ні.

Наведемо таблицю 3.2 зі списком розроблених тест-кейсів. Перевірка проходження кожного тест-кейсу відбувається вручну. Оскільки в даному випадку фактичний результат збігається з очікуваним результатом для усіх тест-кейсів, фактичний результат не було наведено у таблиці, щоб не дублювати текст та задля спрощення таблиці.

Таблиця 3.2 – Список тест-кейсів

ID	**Опис**	**Передумови**	**Кроки**	**Очікуваний результат**	**Статус**
1	**2**	**3**	**4**	**5**	**6**
TC_01	Гра на легкому рівні складності	Гравець знаходиться у сцені головного меню	1) Натиснути кнопку меню «Easy»	Задається легкий рівень складності; перехід до ігрової сцени	Пройдено
TC_02	Гра на нормальному рівні складності	Гравець знаходиться у сцені головного меню	1) Натиснути кнопку меню «Normal»	Задається нормальний рівень складності; перехід до ігрової сцени	Пройдено

Продовження таблиці 3.2

1	2	3	4	5	6
ТС_03	Гра на високому рівні складності	Гравець знаходиться у сцені головного меню	1) Натиснути кнопку меню «Hard»	Задається високий рівень складності; перехід до ігрової сцени	Пройдено
ТС_04	Вибір агента для прямого керування	1) Гравець знаходиться у ігровій сцені 2) Режим прямого керування не активовано	1) Перемістити курсор на довільну позицію	Найближчий до курсору агент виділяється кольором, відмінним від решти агентів	Пройдено
ТС_05	Перехід у режим прямого керування	1) Гравець знаходиться у ігровій сцені 2) Режим прямого керування не активовано	1) Натистути клавішу Y	Камера переміщується в позиції позаду агента; агент починає реагувати на введення користувача	Пройдено
ТС_06	Перехід у режим спостерігача	1) Гравець знаходиться у ігровій сцені 2) Режим прямого керування активовано	1) Натиснути клавішу Y	Камера переміщується в початкову позицію; жоден агент не реагує на введення користувача	Пройдено

Продовження таблиці 3.2

1	2	3	4	5	6
ТС_07	Відбиття м’яча	1) Гравець знаходиться у ігровій сцені 2) Агент рухається вперед	1) Зіштовхнутись із м’ячем	М’яч відбивається із заданої силою удару	Пройдено
ТС_08	Відбиття м’яча у ролі воротаря	1) Гравець знаходиться у ігровій сцені 2) Агент не рухається вперед	1) Зіштовхнутись із м’ячем	М’яч відбивається із заданою силою удару	Пройдено
ТС_09	Забиття м’яча у ворота першої команди	1) Гравець знаходиться у ігровій сцені	1) Забити м’яч у ворота першої команди	Раунд починається спочатку	Пройдено
ТС_10	Забиття м’яча у ворота другої команди	1) Гравець знаходиться у ігровій сцені	1) Забити м’яч у ворота другої команди	Раунд починається спочатку	Пройдено

3.5 Навчання агентів

Перед навчанням агентів необхідно провести інженерію винагород. Інженерія винагород в навчанні з підкріпленням відноситься до процесу розробки та визначення функції винагороди, яка керує процесом навчання агента ШІ. Функція винагороди є ключовим компонентом у навчанні з підкріпленням, оскільки вона визначає поведінку агента, надаючи зворотний

зв’язок щодо вжитих агентом дій. Інженерія винагород передбачає ретельне розроблення функції винагороди, щоб стимулювати агента досягати бажаних цілей або завдань, уникаючи непередбачуваних наслідків або небажаної поведінки. Важливо знайти баланс між наданням чіткого та значущого сигналу про винагороду для спрямування процесу навчання та забезпечення того, щоб функція винагороди відповідала загальним цілям завдання чи проблеми, що вирішується. Інженерія винагород відіграє ключову роль у формуванні поведінки агентів штучного інтелекту в навчанні з підкріпленням і може значно вплинути на продуктивність і ефективність процесу навчання.

Маємо наступні параметри агентів і середовища, та визначення функції винагород:

– середовище, в якому шість агентів змагаються у грі в футбол командами три проти трьох (один воротар та два агента без ролі в кожній з команд);

– задача: забити м’яч у ворота противника, при цьому запобігаючи потраплянню м’яча у власні ворота;

– екзистенційна винагорода у розмірі (1 / макисмальна кількість кроків симуляції), що зараховується кожного кроку симуляції для агентів з роллю воротар;

– екзистенційне покарання у розмірі (1 / макисмальна кількість кроків симуляції), що зараховується кожного кроку симуляції для агентів з роллю нападаючий;

– групова винагорода у розмірі 1 при забитті м’яча у ворота супротивника та групове покарання у розмірі -1 при забитті м’яча у власні ворота;

– векторний простір спостережень: 336 значень розподілених між 11 променями вперед, розподілених на 120 градусів, і 3 променями назад, розподілених на 90 градусів, кожен з яких виявляє 6 можливих типів об’єктів;

– дії: 3 дискретні дії, що відповідають руху вперед-назад, руху вбік, а також обертанню.

Наступним етапом є створення конфігураційного файлу для навчання нейронної мережі. Він містить таку інформацію як назва алгоритму навчання, який застосовуватиметься під час навчання штучної нейронної мережі, гіперпараметри алгоритму навчання та налаштування різного роду механізмів по типу Self-play.

В лістингу 3.1 наведемо частину вмісту конфігураційного файлу config.yaml, що використовувався для навчання штучної нейронної мережі програмного проєкту даної кваліфікаційної роботи.

Лістинг 3.1 Частина вмісту конфігураційного файлу config.yaml:

```
behaviors:
  Generic:
    trainer_type: poca
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
...
```

Після створення конфігураційного файлу та налаштування віртуального середовища Anaconda, в якому слід встановити пакет mlagents, можна приступити до процесу навчання штучної нейронної мережі. Для цього слід запустити консоль Anaconda Prompt і виконати наступні команди: «conda activate *назва_віртуального_середовища*» для активації віртуального середовища, в якому встановлено пакет mlagents; «cd *шлях_до_папки_з_конфігураційним_файлом*» для переміщення у папку з конфігураційним файлом; «mlagents-learn config.yaml —run-

id=*назва_навчальної_сессії*» для початку сеансу навчання штучної нейронної мережі. Після цього необхідно відкрити вікно редактору Unity та запустити навчальну сцену, після чого розпочнеться навчання штучної нейронної мережі (рис 3.4).

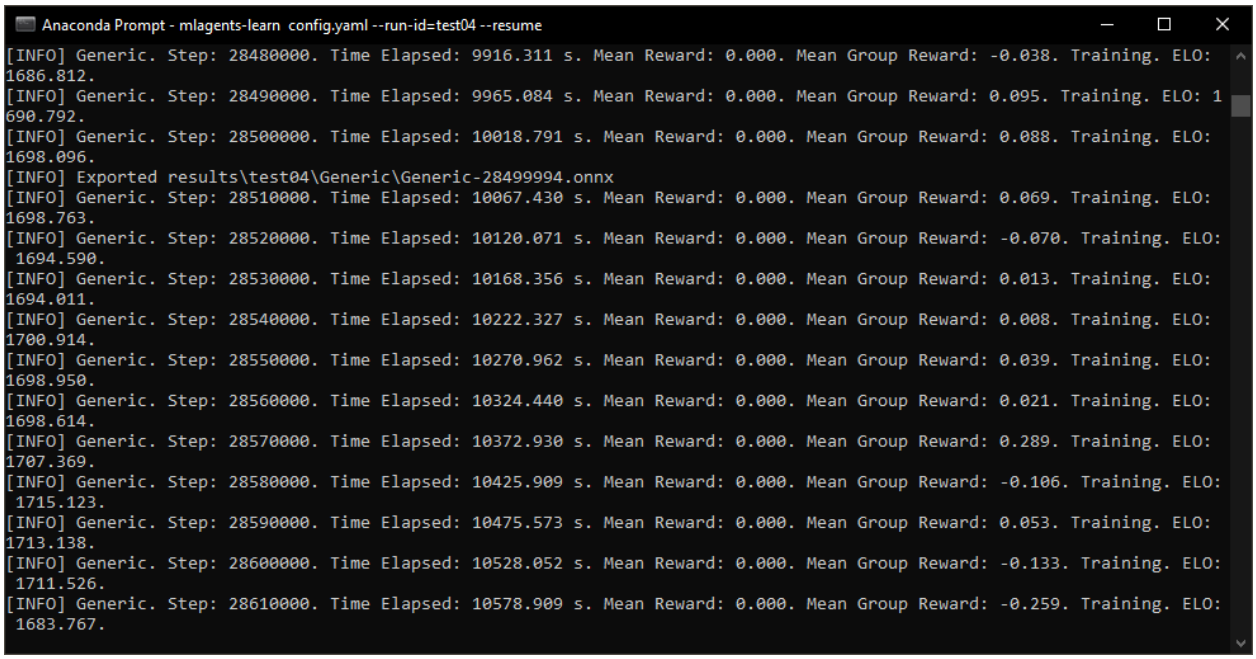

Рисунок 3.4 – Процес навчання

Після проходження навчання вдалося досягти результат у 1500 – 1700 одиниць ELO рейтингу, з початкового ELO рейтингу у 1200 одиниць, що є помірним результатом.

Рейтингова система ELO являє собою метод розрахунку відносного рівня майстерності двох гравців у грі з нульовою сумою. Гра з нульовою сумою – це гра, де виграш або втрата корисності кожного гравця точно врівноважується виграшем або втратою корисності суперника. Ми стикаємося з грою з нульовою сумою, коли один агент отримує +1,0, а його опонент отримує винагороду -1,0. Наприклад, теніс – це гра з нульовою сумою: якщо ви виграєте, ви отримуєте винагороду +1,0, а ваш суперник отримує винагороду -1,0.

Різниця в рейтингу між двома гравцями служить провісником результатів матчу. Наприклад, якщо гравець А має рейтинг ELO у 2100 одиниць, а гравець Б має рейтинг ELO у 1800 одиниць, шанс на перемогу гравця А становить 85% проти 15% у гравця Б. Ілюстрація цього наведена на рисунку 3.5 [26].

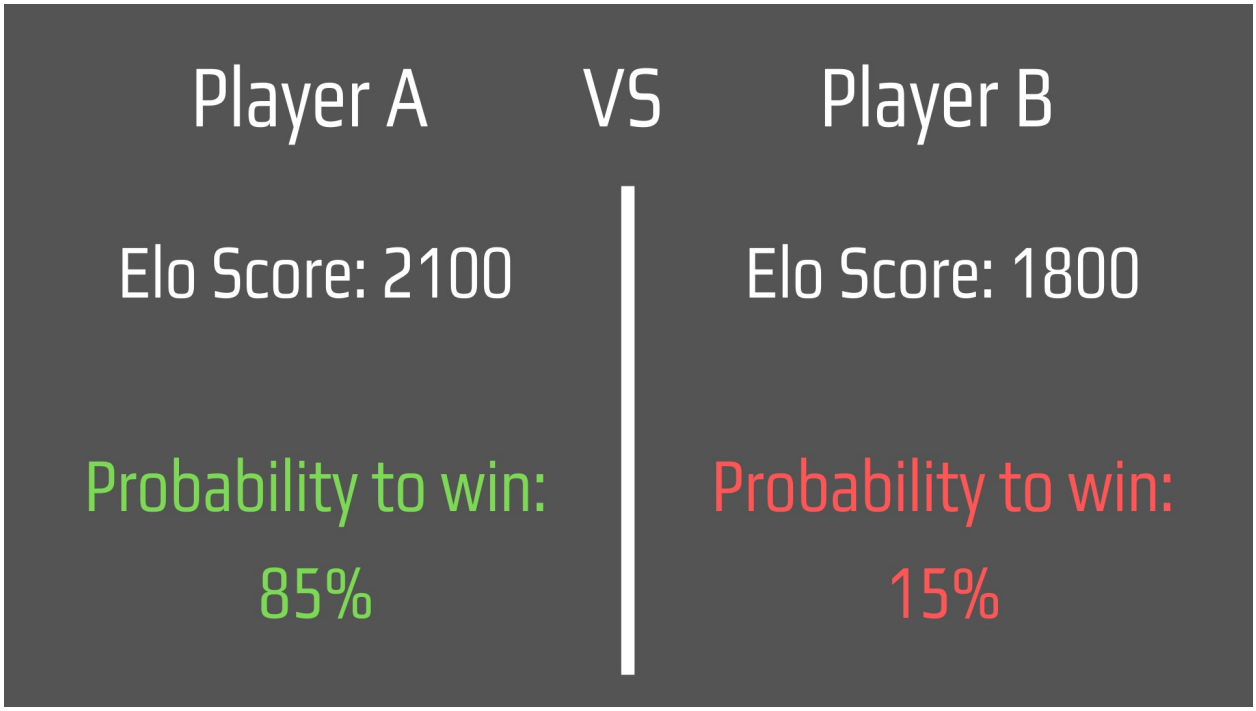

Рисунок 3.5 – Передбачені результати матчу між двома гравцями

Очікуваний рахунок кожного гравця розраховується за формулою

$$E_A = \frac{1}{1+10^{(R_B - R_A)/400}},$$

де E_A – очікуваний рахунок гравця А;

R_A – ELO рейтинг гравця А;

R_B – ELO рейтинг гравця В.

Наприкінці гри, виходячи з результату, фактичний ELO рейтинг гравця оновлюється, для цього використовується лінійне коригування, пропорційне

мірі, на яку гравець перевищив або занизив результативність. Гравець-переможець забирає очки у гравця, що програв:

– якщо виграє гравець з вищим рейтингом, кілька очок буде забрано з гравця з нижчим рейтингом;

– якщо виграє гравець з нижчим рейтингом, багато очок буде забрано з гравця з високим рейтингом;

– якщо відбувається нічия, гравець з нижчим рейтингом отримує кілька очок від гравця з вищим рейтингом. [26]

Рейтинг гравців оновлюється за такою формулою:

$$R'_A = R_A + K(S_A - E_A),$$

де R'_A – новий ELO рейтинг гравця А;

K – рейтинг коригування (К-фактор);

S_A – фактичний рахунок гравця А.

Рисунок 3.6 – ELO рейтинг моделі в залежності від кількості кроків симуляції

3.6 Інструкція користувача

Гра у футбол є популярним видом командного спорту та полягає у необхідності забити, тобто перемістити, м’яч у ворота супротивника, при цьому не давши супротивнику зробити гол у свої ворота. Той, хто заб’є більше м’ячів є переможцем у цій грі.

Після запуску комп’ютерної гри користувач бачить головне меню гри. Для того, щоб почати гру необхідно вибрати один з трьох рівнів складності: Easy – легкий, Normal – середній, Hard – складний; для виходу з гри необхідно натиснути кнопку з написом Exit. Головне меню наведено на рисунку 3.7.

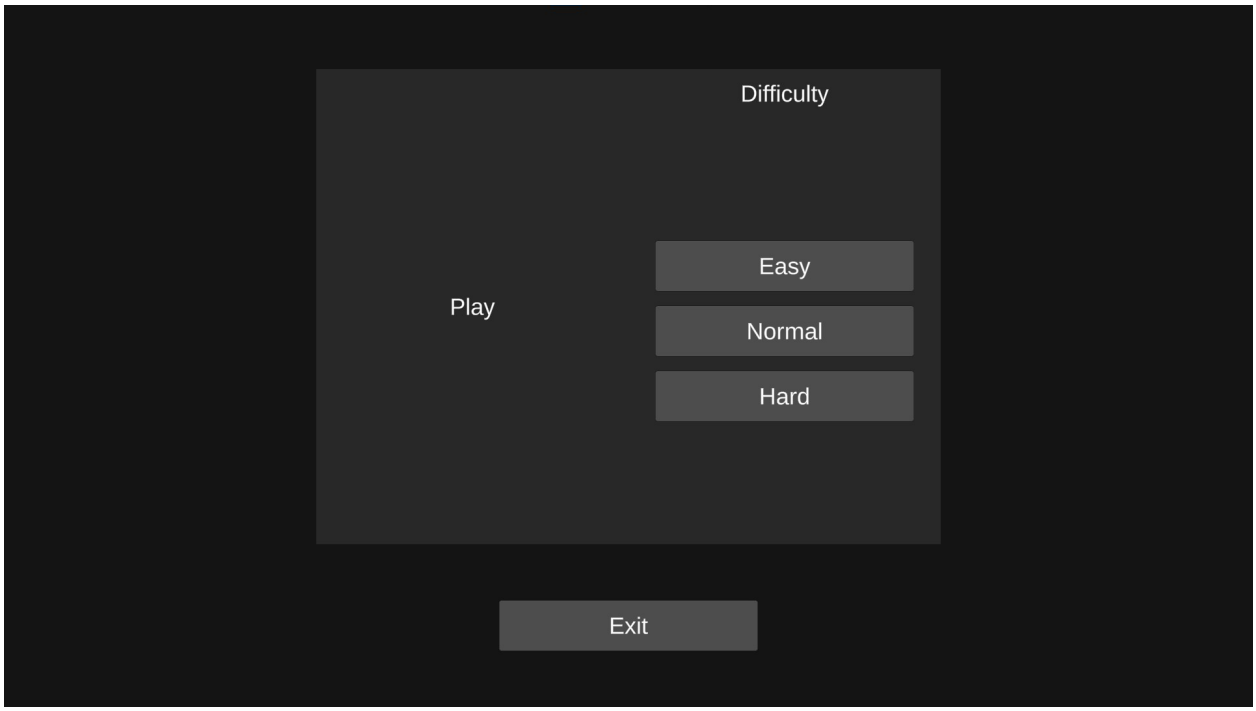

Рисунок 3.7 – Головне меню гри

Після вибору складності користувач потрапляє до ігрової сцени, в якій має змогу спостерігати за ходом гри. Найближчий до курсору користувача

агент підсвічується сірим кольором. Натиснувши клавішу Y можна взяти пряме керування над найближчим до курсору агентом.

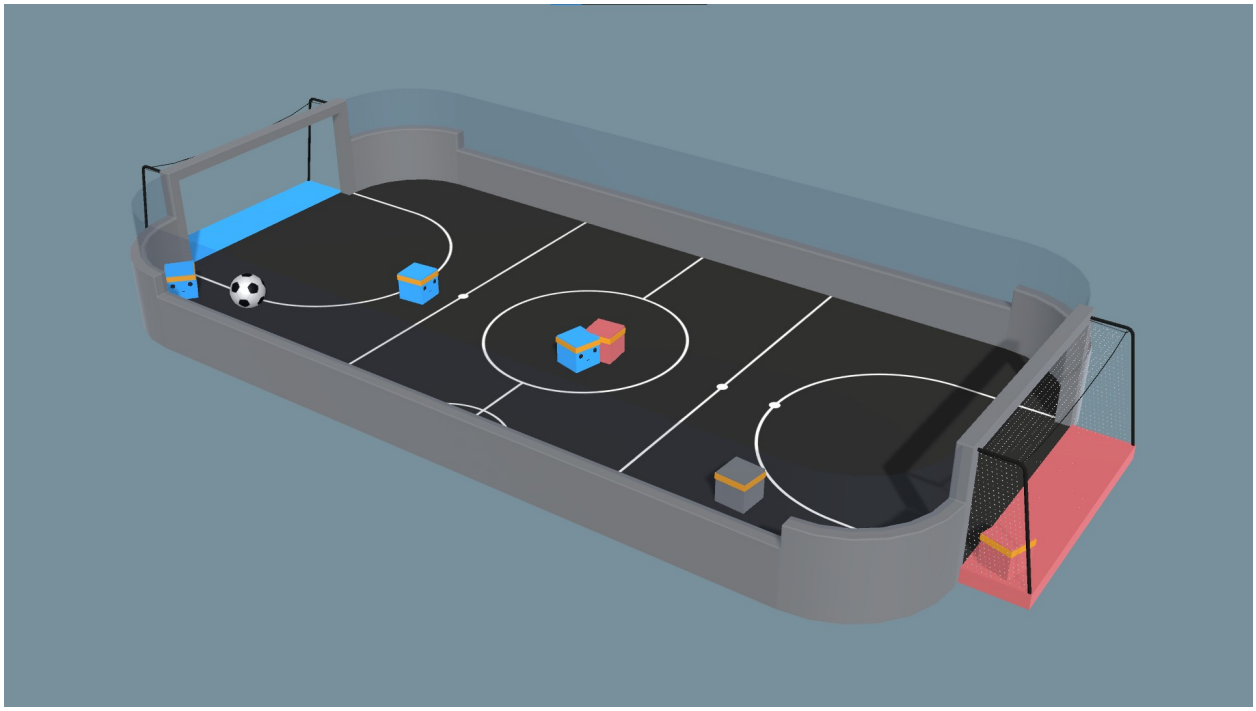

Рисунок 3.8 – Режим спостерігача

У режимі прямого керування користувач може пересуватись на наступні клавіші: W – вперед, A – вліво, S – назад, D – вправо. Рух миші по горизонтальній осі забезпечує обертання агента ліворуч та праворуч.

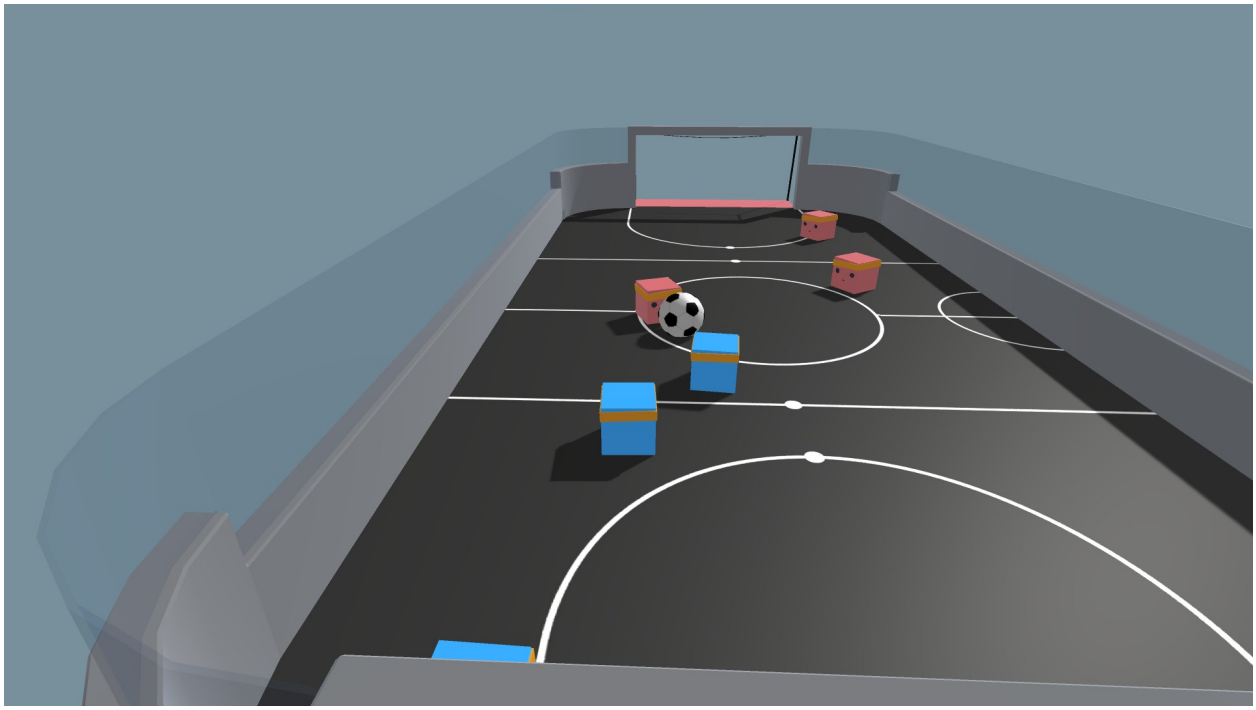

Рисунок 3.9 – Режим прямого керування

Натиснувши клавішу Escape у ігровій сцені, користувач перемикає ігрове меню, в якому можна відновити гру чи вийти до головного меню, а хід гри призупиняється.

Рисунок 3.10 – Ігрове меню

ВИСНОВКИ

У рамках кваліфікаційної роботи було створено комп’ютерну гру у футбол з використанням інтелектуальних агентів на основі штучних нейронних мереж. Під час роботи над цією комп’ютерною грою отримано і покращено навички роботи з ігровим рушієм Unity, отримано навички роботи з пакетом ML Agents.

На основі розробленого програмного рішення було створено комп’ютерну гру у футбол, окрім цього були вирішені наступні задачі: було проведено аналіз ігрових рушіїв, зокрема ігрового рушія Unity; проведено аналіз предметної області машинного навчання; здійснено аналіз існуючих алгоритмів навчання з підкріпленням; вивчено можливості пакету ML Agents; проведено інженерію винагород для алгоритму навчання з підкріпленням; проведено підбір гіперпараметрів для навчання моделі штучної нейронної мережі; проведено навчання (тренування) агентів із застосуванням алгоритму навчання з підкріпленням MA-POCA та механізму Self-play; здійснено тестування комп’ютерної гри; оцінено продуктивність та ефективність керованих штучним інтелектом агентів у середовищі футбольної гри.

У наступних версіях комп’ютерної гри буде замінено агентів зі звичайного куба, тобто простої геометричної фігури, на гуманоїдів, що дозволить наблизити складність симуляції до рівня складності взаємодій реального світу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529.

2. Schulman, J., Wolski, F., Dhariwal, P., et al. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

3. Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017). Mastering the game of go without human knowledge. Nature, 550(7676):354.

4. Espeholt, L., Soyer, H., Munos, R., et al. (2018). IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. arXiv preprint arXiv:1802.01561.

5. Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning. GitHub repository.

6. Oh, J., Chockalingam, V., Singh, S., and Lee, H. (2016). Control of memory, active perception, and action in minecraft. arXiv preprint arXiv:1605.09128.

7. Andrychowicz, M., Wolski, F., Ray, A., et al. (2017). Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058.

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

9. Puigdomènech Badia, A., Piot, B., Kapturowski, S., et al. (2020). Agent57: Outperforming the atari human benchmark. ArXiv, pages arXiv–2003.

10. Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. Behavioral and Brain Sciences, 40.

11. Juliani, A., Berges, V., Teng, E., et al. (2018). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627.

12.Unity Manual: Creating a 3D Game. URL: https://docs.unity3d.com/Manual/Quickstart3DCreate.html (дата звернення 17.04.2024)

13.Unity 5.x Shaders and Effects Cookbook. ISBN 978-1-78528-524-0.

14.Unity Manual: Order of execution for event functions. URL: https://docs.unity3d.com/Manual/ExecutionOrder.html (дата звернення 17.04.2024)

15.Background: Machine Learning. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_21/docs/Background-Machine-Learning.md (дата звернення 17.04.2024)

16.Cohen, A., Teng, E., Berges, V., et al. (2022). On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning. URL: https://rlg.mlanctot.info/papers/AAAI22-RLG_paper_32.pdf (дата звернення 25.04.2024)

17.Oliehoek, A. O., Amato, C. (2016). A Concise Introduction to Decentralized POMDPs. URL: https://www.khoury.northeastern.edu/home/camato/publications/OliehoekAmato16book.pdf (дата звернення 26.04.2024)

18.Lyu, X., Xiao, Y., Daley, B., et al. (2021). Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning. URL: https://xue-guang.com/publication/lyu-2021-contrasting/ (дата звернення 26.04.2024)

19.Foerster, J. N., Song, F., Hughes, E., et al. (2018). Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning. arXiv preprint arXiv:1811.01458.

20.Rashid, T., Samvelyan, M., Witt, C. S., et al. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. arXiv preprint arXiv:1803.11485.

21.Long, Q., Zhou, Z., Gupta, A. (2020). Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning. arXiv preprint arXiv:2003.10423.

22.Lowe, R., Tamar, A., Harb, J., et al. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. arXiv preprint arXiv:1706.02275.

23.Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention Is All You Need. arXiv preprint arXiv:1706.03762.

24.Baker, B., Kanitscheider, I., Markov, T. (2019). Emergent Tool Use From Multi-Agent Autocurricula. arXiv preprint arXiv:1909.07528.

25.Unity Manual: User interface (UI). URL: https://docs.unity3d.com/Manual/UIToolkits.html (дата звернення 17.04.2024)

26.ELO Rating System. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_21/docs/ELO-Rating-System.md (дата звернення 09.05.2024)

27.Gorokhovatskyi V.A. (2018) Image Classification Methods in the Space of Descriptions in the Form of a Set of the Key Point Descriptors. Telecommunications and Radio Engineering, 77 (9), pp. 787-797.

28.Tvoroshenko I.S., and Gorokhovatsky V.O. (2019) Intelligent classification of biophysical system states using fuzzy interval logic, Telecommunications and Radio Engineering, 78(14), pp. 1303–1315.

29.Gorokhovatsky V.A. Efficient Estimation of Visual Object Relevance during Recognition through their Vector Descriptions. Telecommunications and Radio Engineering. – 2016, Vol. 75, No 14. – P. 1271–1283.

30.Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, International Journal of Academic and Applied Research, 7(9), pp. 57-70.

31.Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. Int. scientific symp. «Intelligent Solutions-S».

Computational intelligence. Decision making theory: proceedings of the international symposium, September 28, 2023, Kyiv-Uzhorod, Ukraine, 25-27.

32.Gadetska, S.V., Gorokhovatskyi, V. O., Stiahlyk, N. I., Vlasenko, N.V. Statistical data analysis tools in image classification methods based on the description as a set of binary descriptors of key points. Radio Electronics, Computer Science, Control, 2021, №4, pp. 58-68.

33.Gorokhovatskyi, V., Vlasenko, N. (2021). Редукція опису зображення у складі множини дескрипторів на основі метричного критерію інформативності. Advanced Information Systems, 5(4), pp. 10-16.

34.Gorokhovatskyi V., Tvoroshenko I., Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, Indonesian Journal of Electrical Engineering and Computer Science, 33 (1), 113-125.

35.Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating Image Classification based on a Model for Estimating Descriptor-to-Class Distance. International Journal of Computing, 22(4), 485-492.

36.Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, IEEE Access, 11, 126938-126949.

37.V. Gorokhovatsky, Y. Putyatin and V. Stolyarov (2017) Research of Effectivenss of Structural Image Classification Methods using Cluster Data Model, Radio Electronics Computer Science Control, vol. 3, no. 42, pp. 78-85.

38.Gorokhovatsky V.A., Putyatin Ye.P. (2009) Image Likelihood Measures of the Basis of the Set of Conformities. Telecommunications and Radio Engineering, 68 (9), pp. 763-778.

39.Gorokhovatsky, V.A., Putyatin, Y.P. (2008) Structural recognition of images on the basis of voting models of attributes of typical points, Data recording, storage and processing, 75-85.

40.V. A. Gorokhovatskiy, (2011), Compression of descriptions in the structural image recognition, Telecommunications and Radio Engineering, vol. 70, no. 15, pp. 1363–1371, doi: 10.1615/TelecomRadEng.v70.i15.60.

41.Гороховатський В.О., Пупченко Д.В., Солодченко К.Г. (2018) Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення. Системи управління, навігації та зв'язку. С. 93–98.

42.Gorokhovatskyi V., Gadetska S., Stiahlyk N. (2020) Image structural classification technologies based on statistical analysis of descriptions in the form of bit descriptor set. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020), 2608, pp. 1027-1039.

43.Tvoroshenko, I., & Zarivchatskyi, R. (2020). Analysis of existing methods for searching object in the video stream, in Proc. VI Int. Sci. Practic. Conf. «About the problems of science and practice, tasks and ways to solve them», Milan, pp. 500-505.

44.Гороховатський В., Творошенко І., Сидоренко Д. (2021) Класифікація зображень із використанням кластерного подання, Міжн. наук. симпозіум «Інтелектуальні рішення-С». Обчислювальний інтелект. Теорія прийняття рішень (Вересень 29, 2021). Київ – Ужгород, С. 44-45.

45.Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., Vlasenko, N. (2023) Explanation of CNN Image Classifiers with Hiding Parts. In: J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot (eds), Explainable Deep Learning Artificial Intelligence, pp. 125-146, Academic Press, 346 p.

46.Gorokhovatsky, V.O. and Gadetska, S.V. (2019) Determination of Relevance of Visual Object Images by Application of Statistical Analysis of Regarding Fragment Representation of their Descriptions, Telecommunications and Radio Engineering, 78 (3), pp. 211–220.

47.Gorokhovatsky, V. (2014), Structural Analysis and Intellectual Data Processing in Computer Vision, SMIT, Kharkiv.

48.Gadetska, S.V., Gorokhovatsky, V.O. Statistical Measures for Computation of the Image Relevance of Visual Objects in the Structural Image Classification Methods. Telecommunications and Radio Engineering. – 2018, Vol. 77 (12), pp. 1041–1053.

49.Gadetska S., Gorokhovatskyi V., Stiahlyk N., Vlasenko N. (2022) Aggregate Parametric Representation of Image Structural Description in Statistical Classification Methods. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2022), 3137, pp. 68-77.

50.Гороховатський В.О., Гадецька С.В., Стяглик Н.І., Власенко Н.В. (2020) Класифікація зображень на підставі ансамблю статистичних розподілів за класами еталонів для компонентів структурного опису. Радіоелектроніка, інформатика, управління, №4 , с. 85–94.

51.Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudakova M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, *IEEE Access*, vol. 12, pp. 73376-73385.