МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ЯРЕМЧУК Владислав Миколайович

Програмний модуль інтелектуального агента для самостійної ходьби персонажа в середовищі Unity / Software Module for training of an intelligent agent for autonomous character walking in the Unity environment

Спеціальність 122 «Комп'ютерні науки»
освітньо-професійна програма – Штучний інтелект
Кваліфікаційна робота

Виконав студент групи КНШІ-41
В.М. Яремчук
_____

Науковий керівник
к.т.н., доцент Т.В. Лендюк
_____

Кваліфікаційну роботу
допущено до захисту
«___»_____20___р.

Завідувач кафедри
_____Н.В. Дзюбановська

ТЕРНОПІЛЬ – 2026

Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 «Комп’ютерні науки»
освітньо-професійна програма – Штучний інтелект

«ЗАТВЕРДЖУЮ»
В.о. завідувача кафедри
_____ Н.В. Дзюбановська

«____» _____20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Яремчуку Владиславу Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи
Програмний модуль інтелектуального агента для самостійної ходьби персонажа в середовищі Unity / Software Module for training of an intelligent agent for autonomous character walking in the Unity environment»
керівник роботи к.т.н., доцент Т. В. Лендюк
затверджений наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- провести аналіз можливостей середовища Unity для реалізації фізичної симуляції та навчання агентів;
- сформулювати постановку задачі навчання агента самостійній ходьбі у фізично-реалістичному середовищі;
- розробити модуль прийняття рішень на основі нейронної мережі із використанням Unity ML-Agents;
- провести навчання агента за допомогою алгоритму Proximal Policy Optimization;
- провести аналіз результатів навчання за допомогою TensorBoard та порівняння ефективності моделей.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
- зображення фізичних компонентів у середовищі Unity;
- зображення фізичного середовища та тіла агента у Unity;
- блок-схеми алгоритму роботи навчання агента самостійної ходьби;
- графіки результатів навчання моделі в TensorBoard.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Огляд літературних джерел відповідно до предметної області та складання плану роботи	до 01.01.2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02.2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент _____ *Яремчук В.М.*
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ *Лендюк Т. В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи: 51 с., 9 рис., 2 табл., 2 додатки, 41 джерел.

Об’єктом дослідження є процеси самостійного навчання інтелектуальних агентів фізично реалістичним рухам у віртуальних симуляційних середовищах.

Метою роботи є розроблення програмного модуля інтелектуального агента для навчання ходьби у середовищі Unity, який використовує методи підкріплювального навчання для формування стабільного та адаптивного пересування.

Методи дослідження: методи навчання з підкріпленням для формування оптимальної поведінки агента; методи оптимізації політики на основі алгоритму Proximal Policy Optimization (PPO); методи математичного моделювання процесу прийняття рішень у задачі автономного пересування агента.

Результатом роботи є програмний модуль, що інтегрує фізичні компоненти Unity та алгоритми машинного навчання, забезпечуючи агенту здатність самостійно підтримувати баланс, виконувати стабільні кроки та адаптуватися до змінних умов динамічного середовища.

Розроблений модуль може бути використаний як платформа для дослідження алгоритмів автономного пересування в робототехніці.

Ключові слова: ІНТЕЛЕКТУАЛЬНИЙ АГЕНТ, ПІДКРІПЛЮВАЛЬНЕ НАВЧАННЯ, UNITY, АВТОНОМНА ХОДЬБА, ФІЗИЧНА СИМУЛЯЦІЯ, ML-AGENTS.

ANNOTATION

Explanatory note to the qualification thesis: 51 pages, 9 figures, 2 tables, 2 appendix, 41 references.

The object of research is the processes of autonomous learning of physically realistic movements by intelligent agents in virtual simulation environments.

The purpose of the work is to develop a software module of an intelligent agent for learning walking behavior in the Unity environment, using reinforcement learning methods to achieve stable and adaptive locomotion.

Research methods: reinforcement learning methods for forming the agent's optimal behavior; policy optimization methods based on the Proximal Policy Optimization (PPO) algorithm; methods of mathematical modeling of the decision-making process in the task of autonomous agent locomotion.

The result of the work is a software module that integrates Unity physical components and machine learning algorithms, providing the agent with the ability to independently maintain balance, perform stable walking steps, and adapt to changing conditions of a dynamic environment.

The developed module can be used as a platform for researching autonomous locomotion algorithms in robotics.

Keywords: INTELLIGENT AGENT, REINFORCEMENT LEARNING, UNITY, AUTONOMOUS WALKING, PHYSICAL SIMULATION, ML-AGENTS.

ЗМІСТ

ВСТУП

У сучасну епоху розвитку робототехніки та штучного інтелекту особливу актуальність набуває створення агентів, здатних до самостійного навчання фізично реалістичним рухам у динамічному середовищі. Одним із ключових напрямів є навчання агентів ходьбі у віртуальних симуляціях, що дозволяє відтворювати складні механізми балансування, координації рухів та стабільного пересування. Такі моделі є базою для подальшого розвитку автономних роботизованих систем, які можуть функціонувати без постійного зовнішнього керування.

Традиційні підходи до керування рухом роботизованих агентів здебільшого базуються на заздалегідь визначених анімаціях, кінематичних моделях або простих алгоритмах контролю. У таких системах рух задається вручну, що суттєво обмежує можливість адаптації до нових умов та не дозволяє моделювати процес навчання поведінки. У результаті подібні підходи не відображають реальної складності задач автономного пересування в фізичному середовищі.

Використання алгоритмів machine learning (ML), зокрема навчання з підкріпленням, дозволяє перейти до принципово іншого підходу, де агент самостійно формує стратегію руху на основі взаємодії із середовищем. У процесі навчання він отримує винагороди за успішні дії, такі як підтримка рівноваги, просування у заданому напрямку або уникнення падіння, та штрафи за неефективну або нестабільну поведінку. Це забезпечує поступове формування оптимальних патернів ходьби без явного програмування кожного етапу руху.

Актуальність роботи обґрунтована необхідністю розробки програмного модуля інтелектуального агента для навчання ходьби у середовищі Unity, який може виступати як універсальна платформа для дослідження алгоритмів автономного пересування. У межах такого підходу реалізується фізично-орієнтована симуляція тіла агента, збір і обробка даних про його стан,

формування функції винагороди та поступова оптимізація поведінки за допомогою алгоритмів навчання з підкріпленням.

Запропоноване рішення інтегрує фізичні можливості Unity та інструментарій Unity ML-Agents Toolkit, що дозволяє реалізувати повний цикл навчання агента — від взаємодії із середовищем до оновлення параметрів моделі. Це забезпечує формування стійких навичок локомоції та відкриває можливості для подальших досліджень у сфері робототехніки, автономних систем та інтелектуального керування рухом.

Мета і завдання дослідження. Метою роботи є створення системи навчання автономної ходьби інтелектуального агента у симуляційному середовищі, зокрема, формування архітектури програмного модуля, який дозволяє агенту самостійно навчатися пересуванню на основі взаємодії з фізичним середовищем.

Для досягнення поставленої мети в роботі сформульовано такі завдання:

- провести аналіз можливостей середовища Unity для реалізації фізичної симуляції та навчання агентів;
- сформулювати постановку задачі навчання агента самостійній ходьбі у фізично-реалістичному середовищі;
- розробити модуль прийняття рішень на основі нейронної мережі із використанням Unity ML-Agents;
- провести навчання агента за допомогою алгоритму Proximal Policy Optimization;
- провести аналіз результатів навчання за допомогою TensorBoard та порівняння ефективності моделей.

Об’єктом дослідження є процес навчання інтелектуального агента автономній ходьбі у фізично-симульованому середовищі.

Предметом дослідження є методи, моделі та алгоритми програмного модуля інтелектуального агента для самостійної ходьби персонажа в середовищі Unity.

Для розв’язання задач у роботі застосовано такі методи досліджень: створення агентів, навчання агентів, використання алгоритмів machine learning

(ML), навчання з підкріпленням, фізично-орієнтована симуляція тіла агента, збір і обробка даних про його стан, формування функції винагороди та поступова оптимізація поведінки за допомогою алгоритмів навчання з підкріпленням.

Наукова новизна роботи полягає в тому, що запропоновано використання навчання з підкріпленням, де агент самостійно формує стратегію руху на основі взаємодії із середовищем. У процесі навчання агент отримує винагороди за успішні дії та штрафи за неефективну або нестабільну поведінку, що забезпечило поступове формування оптимальних патернів ходьби без явного програмування кожного етапу руху.

Практичне значення одержаних результатів полягає у використанні навчання з підкріпленням, де агент самостійно формує стратегію руху на основі взаємодії із середовищем Unity, що дало можливість реалізувати повний цикл навчання агента.

Публікації та апробація КР. Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах (додаток Б): І міжнародної науково-практичної конференції студентів і молодих вчених ICSPM 2026: Інтелектуальні обчислювальні системи та управління проєктами, Тернопіль, Україна, Західноукраїнський національний університет, 21-22 травня 2026 р.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 СТАН ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасні дослідження у галузі штучного інтелекту та робототехніки активно використовують симуляційні середовища для розробки алгоритмів автономного руху. Одним із таких напрямків є створення інтелектуальних агентів, здатних самостійно пересуватися у віртуальному середовищі, реагувати на змінні умови та навчатися ефективним паттернам ходьби. Симуляція у середовищі Unity дозволяє моделювати фізичні взаємодії, враховувати гравітацію, тертя та обмеження руху, що є критично важливим для розробки алгоритмів самонавчання [11, 21, 37].

Роботи, що вміють ходити, можуть застосовуватись для виконання різних прикладних задач у промисловості, сільському господарстві та сервісній сфері [1, 4, 13, 36, 38]. Вони можуть перевіряти стан рослин на полях, переносити невеликі вантажі, доставляти інструменти, добрива або корми для тварин. У складі логістичних систем на виробничих підприємствах такі роботи здатні переміщати комплектуючі між цехами, транспортувати дрібні пакунки на складі або виконувати інспекцію важкодоступних ділянок. Базовий штучний інтелект забезпечує орієнтацію на місцевості, планування траєкторії та виконання завдань без постійного контролю людини, що дозволяє підвищити ефективність рутинних операцій та зменшити фізичне навантаження на персонал [10, 21, 23].

Ходячі роботи можуть застосовуватись у тих умовах, де колісні системи неефективні. Вони здатні пересуватися нерівними, пересіченими або природними поверхнями, обходити перешкоди та підніматися на сходи або бордюри, що важко або неможливо для колісних роботів. Крім того, ходячі роботи можуть маневрувати у вузьких або засаджених ділянках, наприклад між рядами рослин на фермі або у складських приміщеннях із нестандартним розташуванням стелажів, та краще розподіляють вагу на опори, зменшуючи ризик пошкодження делікатних поверхонь. Це робить їх придатними для

застосувань, де потрібна гнучкість пересування та здатність адаптуватися до складного середовища [21, 39].

Одними з представників цього напряму є такі серії роботизованих систем:
Agrobot E-Series - система для автоматизованого збору полуниці [1].
У своїй роботі використовує:

- комп’ютерний зір для розпізнавання та визначення стиглості ягід;
- нейронні моделі для класифікації об’єктів у полі;
- систему прийняття рішень для вибору;
- роботизовані маніпулятори для фізичного збору плодів.

John Deere Autonomous Tractor - автономний трактор для обробки сільськогосподарських полів [7]. Використовує:

- GPS-навігацію та LiDAR для локалізації;
- алгоритми планування траєкторії руху;
- системи комп’ютерного зору для виявлення перешкод;
- модулі автономного керування рухом.

Naïo Oz - робот для автоматичного прополювання культур [14]. Застосовує:

- камери для розпізнавання рослин;
- моделі машинного навчання для відокремлення бур’янів від культур;
- алгоритми точкового впливу на небажану рослинність;
- автономну систему пересування між рядами.

DJI Agras T40 - агродрон для моніторингу та обробки полів [4]. Використовує:

- системи комп’ютерного зору для аналізу стану рослин;
- моделі обробки зображень для виявлення проблемних зон;
- автоматичне планування маршрутів польоту.

Додатково варто зауважити, що у сучасній ігровій індустрії на даний момент зростає використання методів штучного інтелекту для створення більш імерсивної поведінки персонажів. Тому замість класичного керування анімаціями через ріггінг усе частіше застосовуються фізично-орієнтовані моделі, де рух формується через взаємодію тіла з середовищем. Від чого слідує логічне

поширення підходу, у якому навчання ходьби агентів використовується замість ручного налаштування анімацій [27, 33, 35].

Використання методів машинного навчання та алгоритмів підкріплення дозволяє агентам розробляти власні стратегії руху, адаптуючись до різних типів поверхні, перепон та динаміки середовища. Це відкриває можливості для досліджень у галузях робототехніки, автономних систем та розумних симуляцій [23, 36].

На сьогодні для навчання агентів ходьби застосовуються методи навчання з підкріпленням, а також нейромережеві моделі. В основі таких підходів лежить оцінювання дій агента через функцію винагороди, яка враховує стабільність руху та ефективність пересування. Використання цих методів дозволяє отримувати агентів, здатних формувати природну ходьбу та адаптуватися до змін умов навколишнього середовища [5, 10, 23, 39].

Таким чином, предметна область дослідження охоплює створення інтелектуальних агентів у середовищі Unity з використанням методів машинного навчання для моделювання автономної ходьби та адаптивної поведінки, що є актуальним для досліджень у робототехніці, комп’ютерній графіці та симуляційних системах [9, 10, 23, 28, 37].

1.2 Огляд і аналіз існуючих рішень

У сфері розробки автономних агентів, здатних самостійно пересуватися, основна увага приділяється методам машинного навчання, фізичної симуляції та алгоритмам оптимізації руху. Ці підходи дозволяють агентам навчитися ходити, утримувати баланс та адаптуватися до різних умов середовища без ручного втручання.

1.2.1 Навчання з підкріпленням

Reinforcement Learning (RL) є одним із найбільш поширених та ефективних підходів для навчання агентів локомоції, зокрема ходьбі у фізичному

середовищі. Основна ідея цього методу полягає у тому, що агент навчається через взаємодію із середовищем, поступово формуючи оптимальну модель поведінки на основі отриманої винагороди. На відміну від класичних алгоритмів, де правила руху задаються вручну, у RL агент самостійно визначає найбільш ефективну послідовність дій для досягнення поставленої цілі [5, 10, 21, 23, 36].

У процесі навчання агент отримує інформацію про поточний стан середовища, наприклад положення тіла, швидкість руху, кути суглобів, контакт із поверхнею або відхилення від рівноваги. На основі цих даних модель приймає рішення щодо наступної дії, яка може включати прикладення сили до окремих суглобів або зміну напрямку руху [28, 33]. Після виконання дії середовище повертає агенту новий стан та числову винагороду, що характеризує якість виконаної поведінки.

Система винагород є одним із ключових елементів RL, оскільки саме вона визначає напрямок навчання агента. Позитивна винагорода може надаватися за підтримку рівноваги, стабільне пересування вперед, досягнення необхідної швидкості або правильну послідовність кроків. Натомість штрафи застосовуються у випадках падіння агента, втрати балансу, хаотичних рухів або надмірних витрат енергії. Таким чином агент поступово вчиться максимізувати сумарну винагороду, формуючи більш стабільну та ефективну модель ходьби [10, 13, 21, 23, 36].

Процес навчання складається з великої кількості ітерацій, під час яких агент багаторазово повторює спроби виконання руху. На початкових етапах поведінка агента зазвичай є нестабільною та хаотичною, оскільки модель ще не має сформованої стратегії пересування. Проте з кожною новою ітерацією алгоритм оновлює політику поведінки, враховуючи попередній досвід і результати отриманих винагород. У результаті агент поступово переходить від випадкових рухів до координованої та фізично коректної ходьби [10, 21, 23, 33].

У середовищі Unity ML-Agents цей підхід реалізується за допомогою пакету ML-Agents, який забезпечує інтеграцію ігрового рушія Unity із алгоритмами машинного навчання. Даний інструментарій дозволяє створювати

фізично обґрунтовані симуляції, налаштовувати середовище навчання, систему винагород та параметри агентів. Використання Unity разом із ML-Agents дає можливість проводити тренування у реальному часі, моделювати взаємодію з різними типами поверхонь, перешкодами та фізичними об’єктами, а також тестувати поведінку агента у складних динамічних умовах [10, 23, 28, 29, 33].

1.2.2 Фізично-орієнтовані середовища

Ці моделі базуються на симуляції фізичних властивостей тіла агента, включаючи масу окремих частин тіла, прикладені сили, моменти інерції та взаємодію із навколишнім середовищем. Кожна кінцівка або суглоб агента моделюється як окремий фізичний елемент із власними параметрами руху та обмеженнями. У процесі навчання агент поступово вчиться координувати рухи кінцівок, утримувати рівновагу та ефективно пересуватися у заданому напрямку.

Перевагою такого підходу є висока реалістичність отриманої ходьби, оскільки поведінка агента формується не через заздалегідь задані анімації, а через взаємодію фізичних законів і системи винагород. Це дозволяє агенту адаптуватися до нерівної поверхні, перешкод, зміни швидкості руху або зовнішніх впливів, наприклад поштовхів чи зміни сили тяжіння. Додатково такий підхід забезпечує більш природну поведінку моделі та можливість формування нестандартних способів пересування.

Під час аналізу середовищ для розробки агента самостійної ходьби були розглянуті декілька сучасних платформ фізичної симуляції, зокрема NVIDIA Isaac Sim, MuJoCo, Webots та Unity [5, 7, 17, 34, 35].

Середовище NVIDIA Isaac Sim забезпечує одну з найбільш точних фізичних симуляцій серед сучасних рішень. Воно використовує потужності GPU-обчислень та RTX-рендеринг, що дозволяє досягати високого рівня реалізму як у фізиці, так і у візуалізації сцени. Це особливо важливо для задач робототехніки та digital twin-систем, де необхідна максимальна наближеність до реального світу. Водночас дане середовище має суттєві недоліки: високі вимоги до апаратного забезпечення, складний процес встановлення та конфігурації, а

також значну складність інтеграції з навчальними пайплайнами. Через це Isaac Sim частіше використовується у промислових або дослідницьких проєктах із доступом до потужного обладнання [17].

MuJoCo є одним із найпоширеніших фізичних симуляторів у сфері навчання з підкріпленням та робототехніки. Його головною перевагою є дуже стабільна та математично точна симуляція динаміки тіл, зокрема систем із великою кількістю суглобів, що робить його придатним для задач навчання ходьби, маніпуляції об’єктами та балансування. Багато досліджень у навчанні із підкріпленням базуються саме на MuJoCo. Однак серед недоліків слід відзначити менш інтуїтивний інтерфейс, складніший процес створення та редагування сцен, а також відсутність зручного візуального редактора, що ускладнює розробку для новачків або швидке прототипування [7]. Приклад навчального середовища агента зображено на рисунку 1.1.

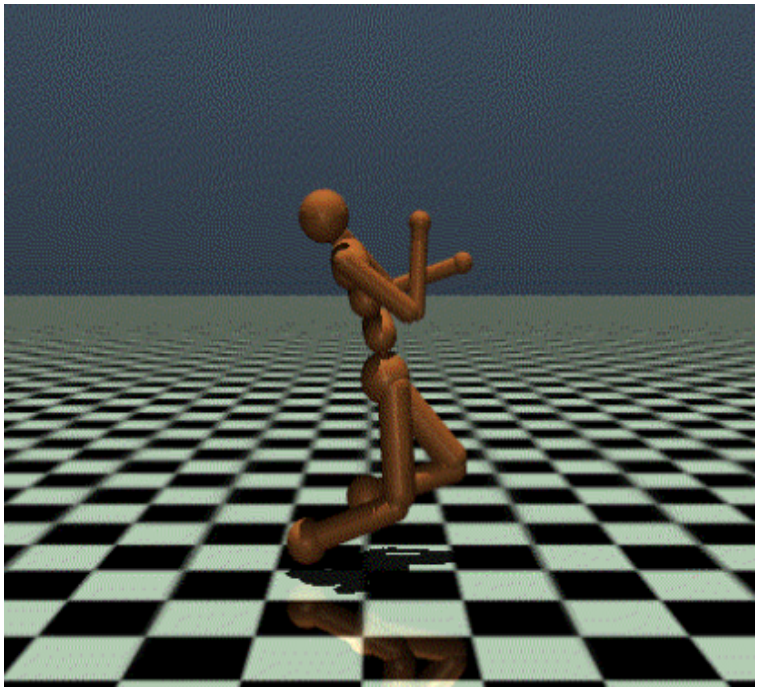

Рисунок 1.1 – Навчальне середовище агента в MuJoCo

Webots є одним із класичних симуляторів робототехніки, який широко використовувався для моделювання мобільних роботів, маніпуляторів та автономних систем. Його перевагою є відносно проста архітектура, підтримка

базових фізичних симуляцій та наявність готових моделей роботів, що дозволяє швидко запускати базові експерименти без складного налаштування середовища. Також Webots має відкриту документацію та підтримку декількох мов програмування для керування агентами. Водночас у контексті сучасних задач навчання з підкріпленням та складних фізичних агентів Webots можна вважати дещо застарілим рішенням. Його візуалізаційні можливості та фізичний рушій поступаються більш сучасним платформам, таким як Unity або NVIDIA Isaac Sim, а гнучкість у створенні складних інтерактивних середовищ є обмеженою. Крім того, інтеграція з сучасними ML-фреймворками та пайплайнами навчання агентів є менш зручною та потребує додаткових налаштувань. Через це Webots частіше використовується у навчальних або базових дослідницьких задачах, але рідше у складних системах навчання агентів самостійній локомоції [5]. Приклад реалізації навчального середовища агента в Webots зображено на рисунку 1.2.

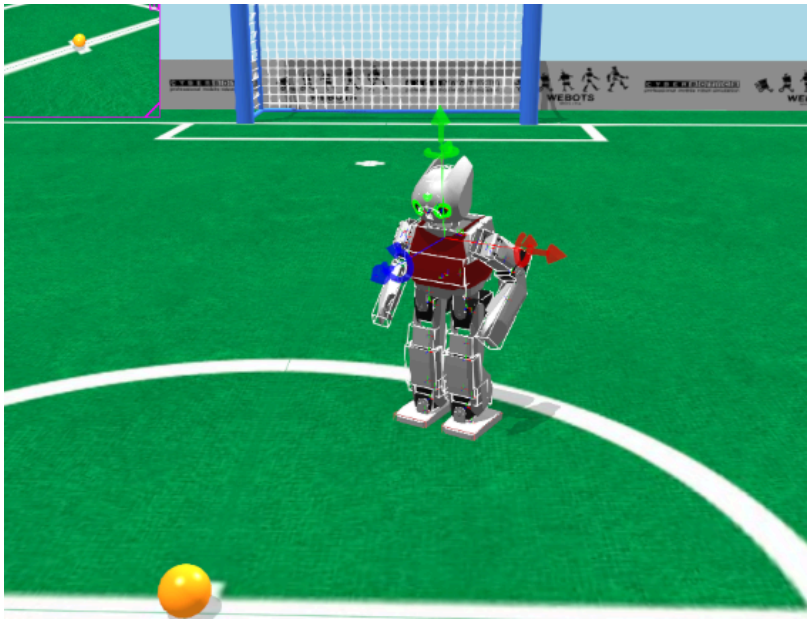

Рисунок 1.2 – Навчальне середовище агента в Webots

Середовище Unity у поєднанні з Unity ML-Agents Toolkit є більш збалансованим рішенням для задач навчання агентів самостійній ходьбі. Unity забезпечує достатньо реалістичну фізичну симуляцію через вбудований фізичний рушій, а також надає зручні інструменти для створення складних сцен і

моделювання поведінки об’єктів. Важливою перевагою є простий та інтуїтивний інтерфейс розробки, який дозволяє швидко створювати та змінювати середовище навчання без глибоких налаштувань фізичного рушія. Unity ML-Agents Toolkit працює, як обгортка для роботи із фреймворком PyTorch. Додатково Unity ML-Agents Toolkit має добре структуровану документацію, велику кількість прикладів та готових навчальних сцен, що значно спрощує процес входу в технологію та прискорює розробку моделей навчання з підкріпленням. Саме поєднання простоти, гнучкості та достатнього рівня фізичної точності робить Unity найбільш практичним вибором для даної задачі [30, 33, 35]. Приклад реалізації навчального середовища агента в Unity зображено на рисунку 1.1.

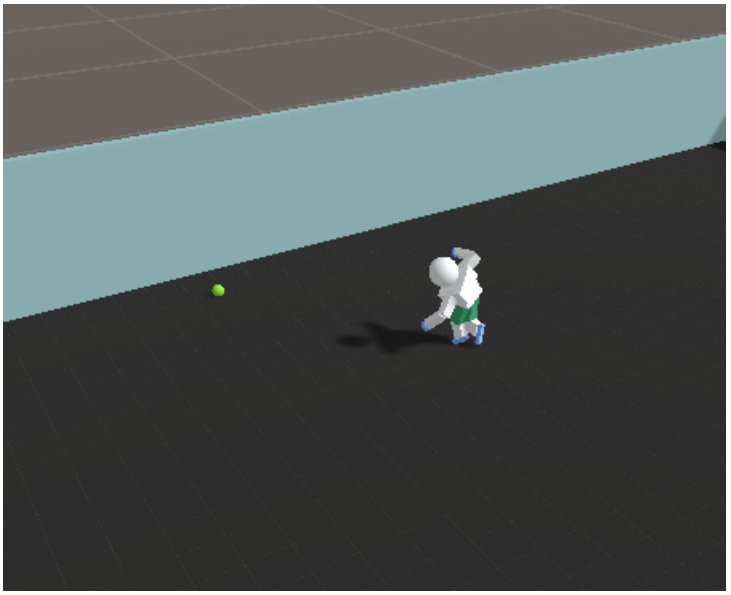

Рисунок 1.3 – Навчальне середовище агента в Unity

1.2.3 Нейронні мережі для прогнозування та генерації руху

Нейронні мережі використовуються як основний механізм обробки стану агента та генерації рішень щодо подальших дій у середовищі Unity ML-Agents Toolkit. Вхідними даними для такої моделі виступає вектор спостережень, який включає положення та орієнтацію тіла агента, лінійні та кутові швидкості, кути в суглобах, а також додаткові сигнали взаємодії із середовищем, наприклад контакти з поверхнею або зіткнення частин тіла. На основі цих даних модель

формує керуючі впливи для окремих суглобів або фізичних елементів тіла, що реалізується через зміну моментів сил або цільових кутів обертання [33].

У задачах навчання локомоції часто застосовується багатошарова перцептронна архітектура, однак у більш складних сценаріях можуть використовуватись рекурентні моделі або політичні функції, які враховують не лише поточний стан, але й попередні спостереження. Це дозволяє моделі утримувати часовий контекст руху та формувати більш узгоджену послідовність дій у динаміці. У результаті агент здатен прогнозувати наслідки своїх рухів на кілька кроків уперед та коригувати поведінку для підтримки стабільної ходьби і балансу.

Навчання таких моделей зазвичай здійснюється за допомогою алгоритмів навчання з підкріпленням, де нейронна мережа оптимізується на основі сигналів винагороди, що відображають якість поведінки агента. Для реалізації процесу оптимізації використовується фреймворк PyTorch, який забезпечує ефективне обчислення градієнтів та оновлення параметрів моделі [18].

Основною перевагою такого підходу є відсутність необхідності ручного задання правил руху або жорстких кінематичних сценаріїв. Поведінка агента формується автоматично в процесі навчання через взаємодію із середовищем та систему винагород. Це дозволяє отримати адаптивну модель локомоції, здатну знаходити ефективні стратегії пересування у складних фізичних умовах без явного програмування кожного етапу руху.

1.2.4 Підходи до оцінки ефективності ходьби

Для оцінки ефективності ходьби агента використовується комплексна система метрик, яка дозволяє аналізувати якість руху не лише з точки зору швидкості, а й з позиції фізичної коректності, стабільності та адаптивності поведінки. Такий підхід потрібен для того, щоб оцінка була об’єктивною і відображала реальну здатність агента виконувати рух в різних умовах середовища. Стабільність руху та баланс - одна з базових характеристик, що визначає здатність агента утримувати вертикальне положення тіла під час руху.

Оцінюється кількість падінь, частота втрати рівноваги, а також величина відхилення корпусу від оптимальної осі. Додатково може враховуватись плавність відновлення балансу після зовнішніх впливів або невдалих кроків. Висока стабільність свідчить про те, що агент правильно координує роботу суглобів і ефективно компенсує фізичні збурення [10, 23, 28].

Швидкість та плавність пересування - характеризує здатність агента досягати заданої швидкості руху без втрати контролю над тілом. Важливим є не лише середня швидкість, але й рівномірність руху в часі. Плавність оцінюється через відсутність різких стрибків прискорення, ривків, надмірних корекцій траєкторії або нестабільних фаз ходьби. Оптимальним вважається рух, який є безперервним і енергетично узгодженим.

Витрати енергії та оптимальність використання ресурсів - відображає ефективність використання фізичних зусиль під час пересування. Агент оцінюється за тим, наскільки економно він використовує сили у суглобах, чи немає зайвих компенсаторних рухів, надмірного напруження або хаотичних дій. У більшості випадків оптимізація спрямована на досягнення максимальної продуктивності руху при мінімальних витратах енергії, що є ознакою природної ходьби [23].

Здатність до адаптації - показує, наскільки гнучко агент реагує на зміни умов середовища. Сюди входить поведінка на нерівних поверхнях, при наявності перешкод, зміні швидкості руху або зовнішніх впливах, таких як поштовхи. Важливо, щоб агент не просто виконував фіксовану послідовність дій, а коригував свою стратегію залежно від ситуації, зберігаючи при цьому стабільність і ефективність ходьби [10, 23, 32].

1.3 Постановка задачі дослідження

Сучасні підходи до створення інтелектуальних агентів, здатних до автономного пересування, базуються на поєднанні фізичного моделювання та методів машинного навчання. У межах даного дослідження розглядається задача

побудови агента, який навчається самостійній ходьбі у фізично-реалістичному симуляційному середовищі Unity, використовуючи механізми навчання з підкріпленням.

Основна мета роботи полягає у розробці програмного модуля, який забезпечує можливість формування поведінки агента через взаємодію з середовищем без явного програмування правил руху, а лише на основі функції винагороди та досвіду навчання.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати функціональні можливості середовища Unity як платформи для фізичної симуляції та реалізації алгоритмів навчання агентів;
- визначити вимоги до системи навчання агента, зокрема до фізичної моделі тіла, середовища взаємодії та критеріїв оцінки якості ходьби;
- сформулювати математичну та логічну постановку задачі навчання автономної локомоції, включаючи опис станів, дій та функції винагороди;
- розробити архітектуру програмної системи, що складається з фізичного середовища, багатосегментної моделі агента, модуля спостереження, модуля прийняття рішень та механізму навчання;
- реалізувати модуль прийняття рішень на основі нейронної мережі з використанням Unity ML-Agents Toolkit, який забезпечує обробку спостережень та генерацію керуючих сигналів для суглобів агента;
- виконати навчання агента за допомогою алгоритму Proximal Policy Optimization (PPO), забезпечивши оптимізацію поведінкової політики через взаємодію з середовищем;
- організувати збір статистичних даних процесу навчання та виконати аналіз отриманих результатів із використанням інструментів візуалізації, зокрема TensorBoard;
- оцінити ефективність навченої моделі за критеріями стабільності руху, швидкості пересування, енергетичної ефективності та здатності до адаптації в різних умовах середовища.

Об’єктом дослідження є процес навчання інтелектуального агента автономній ходьбі у фізично-симульованому середовищі.

Предметом дослідження є методи, моделі та алгоритми програмного модуля інтелектуального агента для самостійної ходьби персонажа в середовищі Unity.

Практична значущість роботи полягає у можливості застосування отриманих результатів для задач робототехніки, симуляції руху, створення автономних систем пересування та розвитку методів навчання агентів у віртуальних середовищах.

2 МЕТОДИ ТА ЗАСОБИ ВИРІШЕННЯ ЗАДАЧІ

2.1 Аналіз математичних моделей та алгоритмів

Основною задачею цього дослідження є розробка інтелектуального агента, здатного самостійно пересуватися у фізично симульованому середовищі. Сучасні підходи до навчання ходьбі агентів часто стикаються з проблемами нестабільності руху, відсутності адаптації до змін середовища та високих обчислювальних витрат для тренування моделей [10, 11, 21, 23].

2.1.1 Мета дослідження

Мета дослідження полягає у створенні програмної платформи для навчання автономної ходьби агента на основі методів навчання з підкріпленням та фізично-орієнтованої симуляції. Розроблена система повинна забезпечувати формування природної поведінки агента без використання заздалегідь підготовлених анімацій руху. У процесі навчання агент має самостійно навчитись підтримувати рівновагу, координувати рухи кінцівок, виконувати стабільні кроки та ефективно пересуватись у заданому напрямку.

Основою дослідження є поєднання фізичної моделі тіла агента із алгоритмами навчання з підкріпленням, що дозволяє формувати поведінку на основі взаємодії із середовищем та системи винагород. Такий підхід забезпечує більш реалістичну модель руху та дозволяє досліджувати процес формування автономної ходьби у симуляційному середовищі.

2.1.2 Формалізація задачі

Для формалізації задачі використано декілька основних математичних підходів. Обчислення винагороди агента на кожному кроці наведено у наступній формулі:

$$R(s_t, a_t) = w_b \cdot B(s_t) + w_s \cdot S(s_t, a_t) - w_e \cdot E(a_t) \quad (1)$$

де, t - крок у епізоді;

s - стан агента;

a - дія агента;

$R(s_t, a_t)$ - винагорода агента у стані s_t при дії a_t;

$B(s_t)$ - показник підтримки балансу;

$S(s_t, a_t)$ - ефективність кроку;

$E(a_t)$ - витрати енергії на виконання дії;

w_b, w_s, w_e - вагові коефіцієнти для балансування компонентів винагороди.

Кумулятивна винагорода визначається як сума дисконтованих винагород [10, 23]:

$$G = \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \tag{2}$$

де, γ - коефіцієнт дисконтування ($0 \leq \gamma < 1$), а сумування ведеться по всіх кроках t епізоду;

T - кількість кроків за епізод;

G - накопичена (дисконтована) винагорода за епізод.

Функція оптимізації політики агента, відповідно до рівняння Беллмана оптимальності [10, 23], визначає процес оновлення параметрів моделі під час навчання:

$$\pi^* = arg\ max_{\pi} E[G] \tag{3}$$

де, π^* - оптимальна політика поведінки агента.

Наступна формула описує процес переходу системи у новий стан під впливом дій агента у фізичному середовищі, де стан системи включає параметри, що описують положення, швидкість, орієнтацію агента та його взаємодію із середовищем:

$$s_{t+1} = f_{phys}(s_t, a_t) \tag{4}$$

де, f_{phys} фізична модель середовища у стані s_t при дії a_t повертає наступний стан s_{t+1}, а саме динаміка суглобів, сила тяжіння, контакт з поверхнею.

Вона гарантує, що дії агента призводять до реалістичного пересування та збереження балансу.

У рамках дослідження алгоритм навчання агента використовує ці функції для поступового покращення політики руху. Нейронна мережа прогнозує оптимальні дії на основі поточного стану агента та навколишнього середовища, а система винагороди забезпечує коригування поведінки для стабільної і ефективної ходьби.

2.2 Розробка архітектури пропонованого рішення

Метою розробки запропонованого рішення є створення платформи для навчання агента автономній ходьбі у фізично-реалістичному середовищі Unity із використанням методів навчання з підкріпленням. Основна концепція системи полягає у тому, що агент навчається виконувати рухи самостійно на основі взаємодії із середовищем та системою винагород.

2.2.1 Ключові частини системи

Система поєднує:

- фізичне середовище - фізично-орієнтована сцена Unity з підлогою та гравітацією;
- фізичну модель агента - агент представлений як скелетна структура з Rigidbody та Joint-ами для симуляції суглобів;
- модуль спостереження - збір даних про стан агента (положення, швидкість, контакт з поверхнею);
- модуль прийняття рішень - нейронна мережа, що обчислює оптимальні дії агента через сили і моменти на Rigidbody [35];

- модуль винагороди - оцінка ефективності кроку через баланс, прогрес та витрати енергії.

2.2.2 Фізичне середовище Unity

Фізичне середовище є базовим елементом системи навчання агента, оскільки саме в ньому відбувається взаємодія моделі штучного інтелекту з об'єктами сцени та формування навичок пересування. Середовище реалізується у Unity, яке містить вбудований фізичний рушій для моделювання взаємодії тіл, гравітації та зіткнень [34, 35].

Основою сцени виступає поверхня пересування, що моделює підлогу або іншу опорну площину. Для цього використовується об’єкт із компонентом колайдера, який визначає межі фізичної взаємодії. Поверхня може мати різні фізичні параметри, зокрема коефіцієнт тертя, що впливає на стабільність ходьби агента та можливість ковзання.

Другим важливим елементом є гравітаційна модель середовища. У фізичному рушії Unity використовується глобальна сила тяжіння, яка впливає на всі об’єкти з компонентом фізичного тіла (Rigidbody). Це дозволяє агенту взаємодіяти з поверхнею природним чином: під час руху ноги створюють контакт із підлогою, а центр мас агента змінює своє положення відповідно до прикладених сил.

Для більш реалістичної симуляції у середовищі можуть використовуватися фізичні матеріали. Вони задають такі параметри взаємодії об’єктів, як:

- коефіцієнт тертя між поверхнями;
- пружність зіткнення.

Налаштування цих параметрів дозволяє контролювати поведінку агента під час руху, наприклад зменшувати ковзання або навпаки створювати складні умови пересування.

Додатково середовище може містити перешкоди та нерівності поверхні, які використовуються для тестування здатності агента адаптуватися до змінних

умов. Такі елементи реалізуються за допомогою стандартних об’єктів сцени з Collider-компонентами та можуть мати різну форму, розмір і розташування [31].

У результаті формується фізично коректне середовище, у якому агент взаємодіє з об’єктами відповідно до законів механіки. Це створює необхідні умови для навчання моделі стабільній та адаптивній ходьбі.

2.2.3 Фізична модель агента

Фізична модель агента визначає структуру тіла та спосіб взаємодії його частин із фізичним середовищем. Основною метою цієї моделі є створення динамічної системи, яка може реагувати на сили, підтримувати баланс і виконувати рухи, необхідні для ходьби. Реалізація моделі здійснюється у середовищі Unity з використанням стандартних фізичних компонентів рушія.

Агент представляється як сукупність окремих сегментів, які відповідають основним частинам тіла, наприклад: тулубу, стегнам, гомілкам та стопам. Кожен сегмент реалізується як окремий об’єкт сцени, що містить компонент Rigidbody, який відповідає за фізичні властивості тіла. Цей компонент визначає масу об’єкта, його інерцію, швидкість руху та реакцію на прикладені сили [34].

Для забезпечення взаємодії сегментів між собою використовуються суглобові з’єднання (Joint). Вони дозволяють обмежувати рух між частинами тіла та визначати допустимі кути обертання. Найчастіше для таких задач застосовуються універсальне з’єднання (ConfigurableJoint) або шарнірне з’єднання (HingeJoint), які дають змогу контролювати рух суглобів у заданих межах. Завдяки цьому створюється кінематична структура, подібна до реального механічного або біологічного механізму [29].

Ще одним важливим елементом фізичної моделі є Collider-компоненти, які визначають форму сегментів тіла для розрахунку зіткнень. Найчастіше використовуються капсульні колайдери (CapsuleCollider) його вигляд наведено на рисунку 2.1 або колайдер у формі коробки (BoxCollider) рисунок 2.2, оскільки вони забезпечують стабільні розрахунки фізики та добре підходять для моделювання кінцівок. Колайдер дозволяє агенту взаємодіяти з поверхнею

середовища, визначати момент контакту ніг із підлогою та запобігати проходженню об’єктів один крізь одного [31].

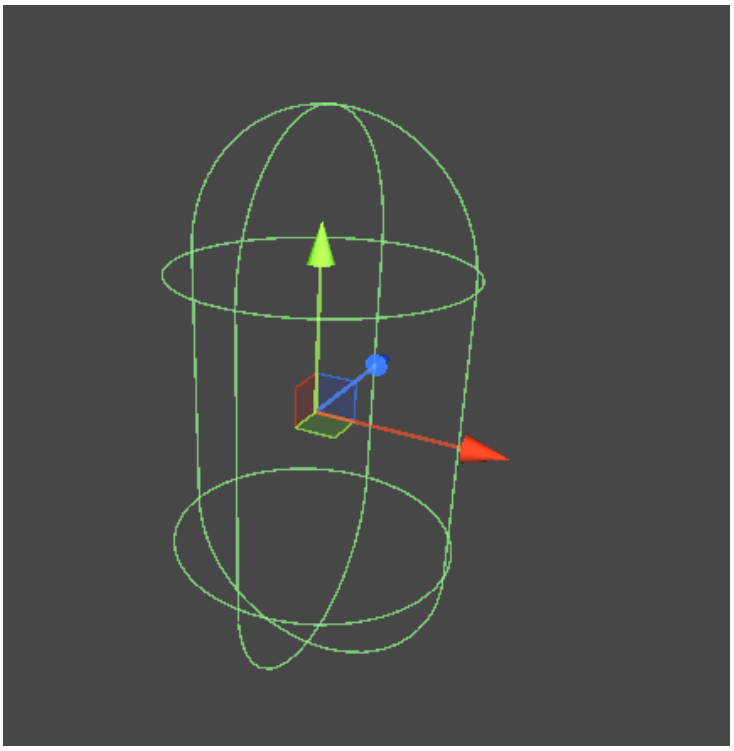

Рисунок 2.1 – Вигляд капсульного колайдера

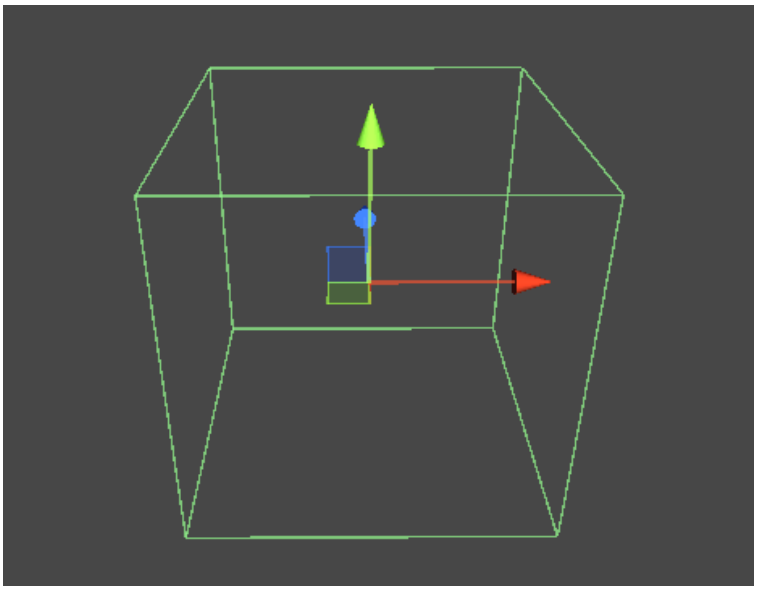

Рисунок 2.2 – Вигляд колайдера у формі коробки

Для підвищення стабільності моделі також враховується центр мас агента. Правильне розташування центру мас дозволяє забезпечити баланс під час руху та зменшити ризик падіння. У налаштуваннях Rigidbody центр мас може змінюватися вручну для досягнення більш реалістичної поведінки системи [35].

Керування рухом агента здійснюється через прикладання сил до сегментів тіла. Алгоритм керування отримує дані про стан системи та визначає величину сили або обертального моменту, які необхідно застосувати до суглобів або сегментів. У результаті цього агент може змінювати положення кінцівок, підтримувати рівновагу та здійснювати кроки [29].

Таким чином, фізична модель агента складається з набору сегментів із компонентами Rigidbody, системи суглобових з'єднань та Collider-елементів для взаємодії з середовищем. Така структура дозволяє створити реалістичну динамічну модель, здатну реагувати на фізичні сили та використовувати їх для формування навичок автономної ходьби.

2.2.4 Модуль спостереження

Модуль спостереження призначений для збору та формування інформації про поточний стан агента і середовища. Ці дані використовуються алгоритмом керування для прийняття рішень щодо подальших дій агента. Реалізація модуля виконується у середовищі Unity, де інформація отримується безпосередньо з фізичних компонентів об'єктів сцени.

Основною функцією модуля є збір параметрів стану агента. До таких параметрів належать координати сегментів тіла, їхня орієнтація у просторі та лінійна і кутова швидкість. Ці дані отримуються з компонентів Rigidbody, що дозволяє визначати поточний рух кожної частини тіла та загальний стан системи.

Окрім положення і швидкості, модуль спостереження фіксує орієнтацію тіла агента відносно поверхні. Для цього використовується інформація про кут нахилу тулуба або напрямок вектора "вгору". Такий параметр є важливим для оцінки балансу, оскільки значне відхилення від вертикального положення може свідчити про втрату стійкості або падіння.

Ще одним важливим джерелом інформації є контакт агента з поверхнею. Для визначення контакту використовуються Collider-компоненти та механізми обробки зіткнень. Система може визначати, чи торкається нога поверхні, чи знаходиться вона у повітрі. Ця інформація допомагає моделі зрозуміти фазу кроку та коригувати рухи для підтримання стабільності.

Зібрані дані формують вектор спостереження, який передається до алгоритму керування. Такий вектор може містити координати, швидкості, кути обертання та сигнали контакту з поверхнею. Отримана інформація використовується нейронною мережею або іншим алгоритмом прийняття рішень для визначення наступних дій агента [21, 23].

Таким чином, модуль спостереження забезпечує зв’язок між фізичним середовищем і алгоритмом керування, надаючи актуальні дані про стан агента. Це дозволяє системі аналізувати поточну ситуацію та адаптувати поведінку для досягнення стабільного і ефективного пересування.

2.2.5 Модуль прийняття рішень

Модуль прийняття рішень відповідає за визначення дій агента на основі даних, отриманих із модуля спостереження. Його основним завданням є аналіз поточного стану системи та обчислення таких керуючих сигналів, які дозволяють агенту підтримувати рівновагу і виконувати рух уперед [21, 22, 23].

Основу модуля складає модель штучного інтелекту, яка формує політику поведінки агента. Для навчання та інтеграції такої моделі у середовище Unity може використовуватися пакет Unity ML-Agents Toolkit [33], що забезпечує взаємодію між симуляційним середовищем та алгоритмами навчання з підкріпленням. Модель отримує на вході вектор спостережень, сформований із параметрів стану агента, і обчислює набір дій, які необхідно виконати на поточному кроці симуляції [33]. Дії агента можуть бути представлені у вигляді керуючих сигналів для фізичної моделі. Це можуть бути значення обертальних моментів або сил, що прикладаються до окремих сегментів тіла або суглобів. Отримані значення передаються до фізичних компонентів системи, зокрема до

Rigidbody та Joint-з’єднань, що призводить до зміни положення кінцівок і створення руху [29].

Модуль прийняття рішень працює у циклі, який повторюється на кожному кроці симуляції. Спочатку агент отримує поточний стан середовища, після чого модель обчислює оптимальні дії відповідно до своєї політики. Потім ці дії застосовуються до фізичної моделі, і фізичний рушій обчислює новий стан системи. Таким чином формується безперервний процес взаємодії між агентом та середовищем [10, 24, 26].

Під час навчання модель поступово вдосконалює свою політику дій, отримуючи сигнали винагороди за успішні або неуспішні рухи. Це дозволяє системі знаходити ефективні способи координації рухів і формувати стабільні патерни ходьби без попереднього програмування конкретних правил руху.

На таблиці 2.1 зображено схему проходження сигналів у системі.

Таблиця 2.1 – Алгоритм проходження сигналів у системі

Етап процесу	Операція	Опис та роль у передбаченні рухів
Збір ознак	Формування вектора	Система зчитує кутові швидкості колін, нахил тулуба та відстань до цілі. Це "зір" агента.
Прямий прохід	Матричне множення ваг	Нейронна мережа обробляє дані через навчені ваги. Це етап "прогнозування" найкращої реакції на стан.
Визначення стратегії	Активація виходу	Мережа видає не один рух, а розподіл ймовірностей або конкретні значення для кожного ступеня свободи (DOF).
Масштабування	Адаптація до фізики	Число від мережі (напр. 0.85) конвертується у фізичну силу (напр. 85 Нм) для приводу стегна.
Виконання	Прикладання сил у Unity	Фізичний рушій зміщує Rigidbody, завершуючи цикл передбаченого руху.

Процес починається з формування вектора станів у середовищі Unity, який подається на вхідний шар політики. Після обробки прихованими шарами

нейронної мережі, система генерує безперервні дії, що трансформуються у крутні моменти фізичних суглобів агента.

Таким чином, модуль прийняття рішень є центральним елементом архітектури системи, оскільки саме він перетворює інформацію про стан агента у конкретні дії, що керують його фізичною поведінкою у середовищі симуляції.

2.2.6 Модуль винагороди

Модуль винагороди використовується для оцінювання ефективності дій агента під час навчання. Його основне призначення полягає у формуванні числового сигналу, який показує, наскільки виконані дії наближають агента до бажаної поведінки [10, 21, 23]. У запропонованій системі винагорода формується на основі кількох ключових критеріїв, які характеризують якість пересування агента.

Першим критерієм є баланс агента. Під час руху важливо, щоб тулуб залишався у відносно стабільному положенні та не відхилявся надмірно від вертикальної осі. Якщо агент втрачає рівновагу або падає, система нараховує негативну винагороду. Натомість стабільне положення тіла забезпечує позитивний сигнал, що стимулює модель підтримувати правильний баланс.

Другим критерієм є прогрес пересування. Агент отримує позитивну винагороду за рух у заданому напрямку. Для цього вимірюється зміна координати агента вздовж основної осі руху між кроками симуляції. Чим більший поступальний рух вперед, тим більший позитивний внесок у загальну винагороду.

Третім компонентом є витрати енергії, які оцінюються через величину сил, що прикладаються до суглобів. Надмірні або різкі рухи можуть призводити до неефективного використання енергії, тому система зменшує винагороду у випадках, коли агент застосовує занадто великі керуючі сигнали. Це стимулює модель знаходити більш економні та стабільні способи пересування [10, 23]. Узагальнений опис показників системи винагород наведено у таблиці 2.2.

Таблиця 2.2 – Основні показники системи винагород

Етап	Умова	Дія системи (Наслідок)
Перевірка цілісності	Контакт таза або голови з поверхнею	Штраф за падіння. Термінація (скидання) епізоду навчання.
Аналіз руху	Просування вперед по цільовій осі	Бонус за прогрес. Заохочення агента рухатися до мети.
Контроль постави	Вертикальність тулуба	Підтримка рівноваги без нахилів.
Енергоефективність	Витрати зусиль моторами суглобів	Штраф за зусилля. Запобігання хаотичним рухам.

Формування винагороди здійснюється на кожному кроці симуляції. Отримане значення передається до алгоритму навчання, який використовує його для оновлення параметрів моделі. Таким чином агент поступово навчається координувати свої рухи так, щоб підтримувати баланс, рухатися вперед і виконувати дії з мінімальними енергетичними витратами.

Завдяки використанню модуля винагороди система може формувати складні патерни руху без прямого програмування конкретних правил. Агент самостійно знаходить оптимальну стратегію пересування шляхом багаторазової взаємодії з фізичним середовищем. Загальний алгоритм взаємодії всіх компонентів інтелектуального агента представлено у додатку А на рисунку 1.

2.2.7 Пакети та бібліотеки

Для реалізації платформи рекомендується використовувати наступні інструменти та пакети Unity:

а) Unity ML-Agents Toolkit - для навчання з підкріпленням та інтеграції нейронних мереж [33];

б) Unity Physics - стандартні компоненти для моделювання фізики агента [35];

в) Barracuda - для виконання нейронних мереж на платформі Unity без зовнішніх бібліотек [28].

Для роботи із інструментами вказаними вище потрібно додатково встановити наступні бібліотеки та інструменти:

а) Unity [34];

б) Microsoft Visual Studio - для роботи з кодом [14];

в) дистрибутив Anaconda для роботи з Python та бібліотеками PyTorch і TensorBoard [2, 18, 25].

Для реалізації системи навчання автономної ходьби необхідно використати набір інструментів та бібліотек, які забезпечують моделювання фізичного середовища, інтеграцію алгоритмів машинного навчання та взаємодію між компонентами системи. Основна реалізація виконується у середовищі Unity, яке надає необхідні інструменти для створення симуляційних середовищ та керування фізичними об’єктами.

2.2.7.1 ML-Agents Toolkit

Одним із ключових компонентів системи є Unity ML-Agents Toolkit, який використовується для реалізації алгоритмів навчання з підкріпленням. Даний пакет забезпечує взаємодію між симуляційним середовищем та моделлю штучного інтелекту [33]. Також для цього інструменту є велика документація та декілька проєктів прикладів для ознайомлення з його роботою [30]. ML-Agents використовує PyTorch як основний фреймворк для побудови та навчання нейронних мереж, які забезпечують процес прийняття рішень агентами та візуалізує результати навчання через фреймворк TensorBoard [18, 25].

ML-Agents дозволяє:

- створювати агентів, які отримують дані спостереження із середовища;
- формувати дії агента на основі політики нейронної мережі;

- обчислювати винагороду та використовувати її для навчання моделі;
- інтегрувати процес тренування з Python-середовищем [20].

Таким чином, цей пакет виступає центральним елементом, який поєднує фізичну симуляцію та алгоритми машинного навчання.

2.2.7.2 Unity Physics

Моделювання фізики агента здійснюється за допомогою стандартних компонентів фізичного рушія Unity Physics [35]. До них належать:

- Rigidbody — відповідає за фізичні властивості об’єктів, такі як маса, швидкість, інерція та реакція на прикладені сили [35];
- Collider — визначає геометрію об’єктів для розрахунку зіткнень [31];
- Joint-компоненти — використовуються для створення з’єднань між сегментами тіла агента [29].

Завдяки використанню цих компонентів створюється реалістична модель взаємодії агента з середовищем, що є необхідною умовою для навчання стабільної ходьби.

2.2.7.3 Barracuda

Для виконання навчених моделей безпосередньо у середовищі Unity використовується пакет Unity Barracuda. Він забезпечує можливість запуску нейронних мереж у форматі ONNX або інших сумісних форматах прямо у середовищі [28].

Основні функції Barracuda:

- виконання нейронних мереж без використання зовнішніх фреймворків;
- інтеграція з компонентами ML-Agents [33];
- підтримка виконання моделей на CPU або GPU.

Це дозволяє використовувати вже навчену модель для керування агентом у режимі реального часу.

Таким чином, використання зазначених пакетів і бібліотек дозволяє об’єднати фізичну симуляцію, алгоритми машинного навчання та систему керування агентом у єдину платформу, що забезпечує реалізацію процесу навчання автономної ходьби у середовищі Unity.

2.3 Обґрунтування вибору запропонованих методів та моделей

2.3.1 Підхід навчання з підкріпленням

У даній роботі для реалізації навчання автономної ходьби агента використовується підхід навчання з підкріпленням (Reinforcement Learning) [23]. Даний підхід дозволяє агенту самостійно формувати поведінку на основі взаємодії з середовищем без задання жорстко визначених правил руху.

Навчання відбувається у вигляді послідовних ітерацій, під час яких агент отримує інформацію про стан середовища, виконує певні дії та отримує числову оцінку винагороди, яка характеризує якість виконаних дій. На основі цієї інформації поступово формується стратегія поведінки агента.

2.3.2 Проксимальна оптимізація політики

У якості основного алгоритму використовується проксимальна оптимізація політики [20, 23, 24]. Даний алгоритм застосовується для оновлення політики агента на основі зібраних даних про взаємодію із середовищем.

Процес навчання включає:

- збір даних про дії агента та їх результати;
- обчислення значення винагороди;
- оновлення параметрів моделі;
- повторення циклу до досягнення стабільної поведінки.

Алгоритм працює ітеративно та дозволяє поступово покращувати результати навчання [23, 24].

Для реалізації політики агента використовується багатошарова нейронна мережа. Вона отримує на вході вектор спостережень та формує набір керуючих сигналів [4, 23].

Структура моделі включає:

- вхідний шар, який приймає параметри стану;
- один або кілька прихованих шарів;
- вихідний шар, що формує дії агента.

Вихідні значення мають безперервний характер і відповідають силам або моментам, які прикладаються до суглобів.

Вектор спостережень формується на основі даних про стан агента у середовищі.

До нього входять:

- положення сегментів тіла;
- швидкості руху;
- орієнтація тулуба;
- інформація про контакт із поверхнею.

Отримані значення передаються до нейронної мережі на кожному кроці симуляції.

Дії агента задаються у вигляді безперервних значень. Кожне значення відповідає керуючому впливу на певний суглоб або сегмент тіла.

На основі цих значень у фізичній моделі:

- прикладаються сили;
- задаються обертальні моменти;
- змінюється положення частин тіла.

Функція винагороди використовується для оцінки ефективності дій агента. Вона формується на кожному кроці симуляції та включає кілька складових:

- винагороду за рух до цілі;
- бонус за збереження балансу;
- штраф за падіння;
- штраф за надмірні зусилля.

3 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Опис набору даних та підготовка до експерименту

У даній роботі використовується підхід навчання з підкріпленням, тому класичний статичний набір даних відсутній. Замість цього дані формуються динамічно під час взаємодії агента із середовищем у процесі симуляції.

Навчальні дані генеруються в режимі реального часу та являють собою послідовності станів, дій та винагород, які отримує агент під час кожного епізоду.

Особливістю реалізованого середовища є наявність цільової точки, до якої повинен дістатися агент. Ця точка:

- розташовується на платформі;
- генерується у випадковій позиції;
- змінює своє положення на початку кожного нового епізоду.

Таким чином, кожен епізод навчання є унікальним, оскільки агент виконує задачу досягнення цілі в різних умовах [26].

Завдяки випадковому розміщенню цільової точки формується різноманітний набір траєкторій руху агента, що необхідно для навчання інтелектуального агента впевненої ходьби. Це дозволяє:

- уникнути перенавчання на фіксовану траєкторію;
- забезпечити узагальнення поведінки;
- формувати більш стійку політику пересування.

Під час кожного кроку симуляції формується набір даних, який включає:

- стан агента, наприклад позиція, швидкість, орієнтація, контакт із поверхнею;
- позицію цільової точки;
- виконану дію, наприклад керуючі сигнали для суглобів;
- отриману винагороду.

Ці дані використовуються алгоритмом навчання для оновлення параметрів моделі.

Перед запуском навчання було підготовлено фізичне середовище у Unity, яке включає:

- платформу для пересування агента;
- систему фізичних компонентів;
- механізм генерації цільової точки у випадковій позиції;
- налаштовані фізичні параметри середовища.

На початку кожного епізоду агент та цільова точка ініціалізуються у нових положеннях.

Навчання відбувається у вигляді серії епізодів. Кожен епізод завершується у випадку:

- досягнення цільової точки;
- втрати рівноваги агентом;
- перевищення максимальної кількості кроків.

Після завершення епізоду середовище повністю перезапускається з новим розташуванням цілі. Таким чином, набір даних у системі формується динамічно під час симуляції, а випадкове розміщення цільової точки забезпечує різноманітність траєкторій руху агента та сприяє формуванню узагальненої поведінки [33].

3.2 Реалізація та налаштування алгоритмів

3.2.1 Встановлення та конфігурація Unity

Для розробки середовища навчання інтелектуального агента використовується ігровий рушій Unity, який забезпечує фізично орієнтовану симуляцію та інтеграцію з алгоритмами машинного навчання.

Проєкт розробляється в операційній системі Windows 11, тому подальші дії можуть відрізнятися залежно від інших операційних систем. Встановлення Unity здійснюється за допомогою Unity Hub, що дозволяє керувати версіями редактора та проєктами. Для коректної роботи ML-Agents необхідно використовувати

версію Unity не нижче 6000.0; у даній роботі використано версію 6000.0.40f1 [32, 33, 34].

Unity забезпечує:

- роботу фізичного рушія;
- створення сцени навчання агента;
- інтеграцію з ML-Agents Toolkit [33].

Щоб налаштувати середовище Unity для використання його у навчанні агента ходьби потрібно виконати такі пункти:

а) завантаження Unity Hub з офіційного сайту [32];

б) встановити необхідної версії Unity за допомогою Unity Hub [34];

в) створити новий проєкт з використанням 3D-середовища;

г) встановити Unity ML-Agents Toolkit до створеного проєкту [33].

Unity Hub дозволяє централізовано керувати проєктами та версіями редактора, що критично для сумісності з ML-Agents.

3.2.2 Установка та налаштування середовища Python

Наступні пункти налаштування середовища вказані у офіційній документації до фреймворку Unity ML-Agents Toolkit [33]. Далі для того, щоб Unity міг взаємодіяти із бібліотеками Python, його потрібно правильно встановити і налаштувати. У документації до Unity ML-Agents Toolkit рекомендують встановлювати Python версії 3.10.12 через Conda [2, 19]. Для того щоб налаштувати Python і бібліотеку mlagents для роботи над проєктом, потрібно виконати такі кроки:

а) Завантажуємо та встановлюємо менеджер пакетів Conda з офіційного сайту [2];

б) Після встановлення Conda встановлюємо потрібну нам версію Python і бібліотеку mlagents. У терміналі виконуємо такі команди:

1) conda create -n mlagents python=3.10.12;
2) python -m pip install mlagents==1.1.0;
3) conda activate mlagents.

Далі потрібно налаштувати саме середовище Python. Для цього виконуємо у терміналі такі команди:

а) pip install torch torchvision torchaudio;

б) pip install numpy;

в) pip install tensorboard.

3.2.3 Налаштування фізичного тіла агента та середовище у Unity

Після налаштування середовища для реалізації логіки навчання агента, необхідно правильно сконфігурувати як саме фізичне тіло агента, так і середовище, у якому він буде взаємодіяти з фізикою.

У розділі 2.2.2 ми розглянули основні частини фізичного середовища потрібного для навчання агента ходьби, а саме:

- поверхня пересування, наприклад підлога;
- конфігурація гравітаційної моделі фізичного середовища;
- різні перешкоди, наприклад виступи, ящики тощо.

Для задачі навчання ходьби критично важливо, щоб середовище забезпечувало стабільну фізичну взаємодію між тілом агента та опорою.

Для цього створюється сцена з плоскою поверхнею, яка виконує роль основної опори. Вона повинна мати коректно налаштований Collider, щоб забезпечити взаємодію з тілом агента.

Далі налаштування глобальної фізики Unity, зокрема параметри гравітації. Вона задає природне падіння тіла агента і дозволяє моделювати реалістичну поведінку при втраті рівноваги у нашому випадку ми можемо залишити стандартні налаштування моделі гравітації [31, 33].

Загалом навчання агента є досить важкою задачею, як зі сторони налаштування всього середовища, так і самого навчання агента ходьби, тому ми обмежимось задачею навчання агента ходьби по рівній поверхні до цілі.

Також додатково було створено обмежувальні елементи стіни або межі сцени, які запобігають виходу агента за межі навчального простору. Це дозволяє стабілізувати процес навчання та уникнути неконтрольованих станів.

Таким чином, фізичне середовище формує базові умови, в яких агент може навчатися пересуванню, балансуванню та взаємодії з фізичними об’єктами сцени. На рисунку 3.1 зображено вигляд фізичного середовища створеного для навчання агента ходьби.

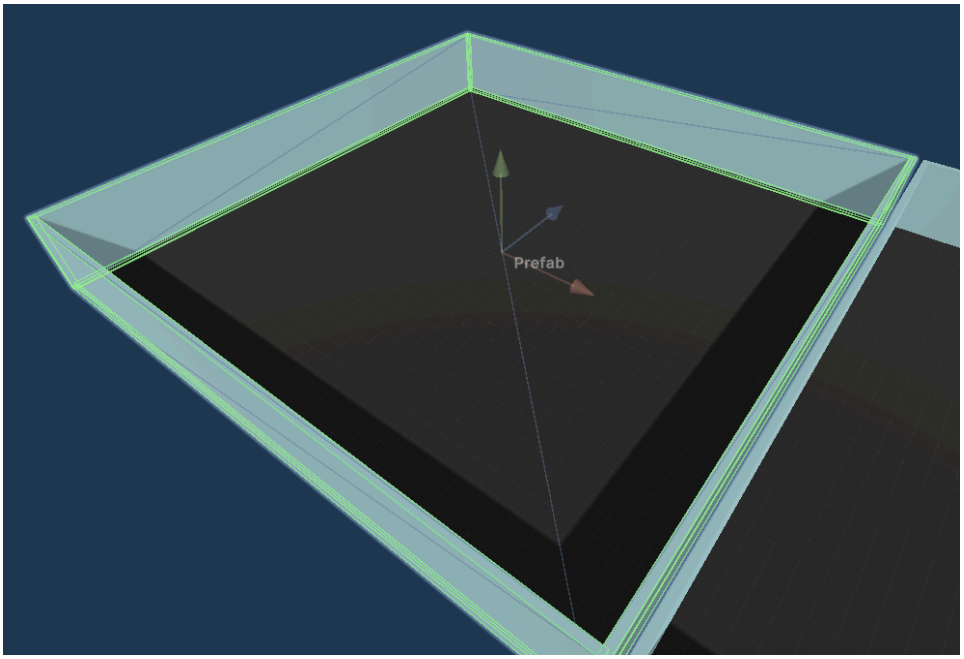

Рисунок 3.1 – Вигляд фізичного середовища сцени

Після підготовки фізичного середовища налаштуємо тіло агента. У даній роботі агент реалізований у вигляді скелетної 3D-моделі з розбиттям на окремі фізичні сегменти тіло, голова, кінцівки. Вони з’єднані між собою фізичними обмежувачами.

Фізична структура агента побудована модульно:

- торс - центральний елемент тіла, який містить є базою для всіх рухів;
- голова - окремий фізичний об’єкт, який під’єднаний через Joint і впливає на балансування;
- руки - симетричні кінцівки, які беруть участь у стабілізації та координації руху;
- ноги - основні елементи, що зможуть рухати агента до цілі.

З’єднання реалізовано за допомогою фізичного компонента ConfigurableJoint, що дає змогу контролювати ступені свободи кожної частини тіла та задавати силу руху сегмента, який приєднаний через це з’єднання. Таким чином можна імітувати роботу м’язів.

Кожна частина тіла має власний Collider і Rigidbody, що забезпечує фізичну взаємодію з середовищем. Всі сегменти працюють як єдина система, але при цьому зберігають локальну фізику, що дозволяє моделювати реалістичну поведінку руху та втрати рівноваги. На рисунку 3.2 зображено тіло агента та його фізичні компоненти [31, 35].

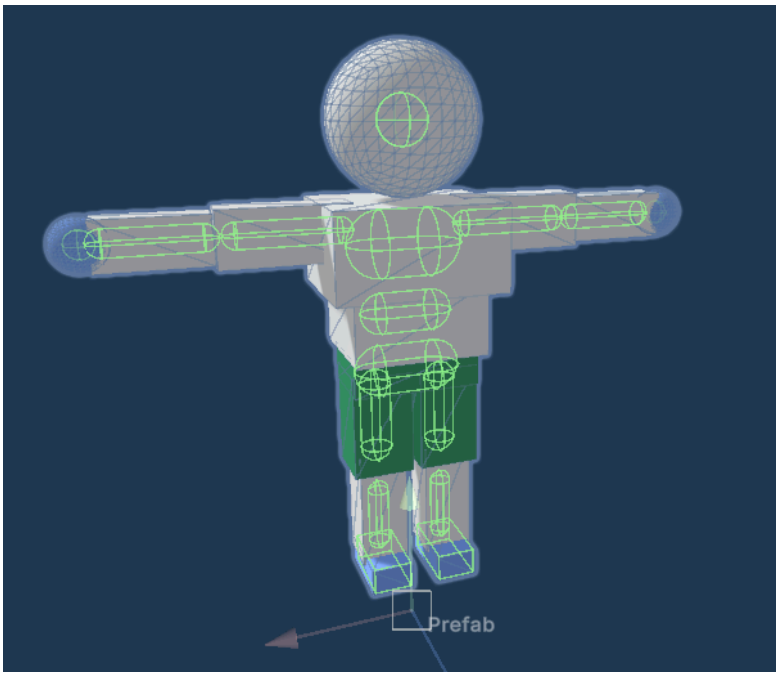

Рисунок 3.2 – Вигляд тіла агента

3.2.4 Реалізація програмної логіки агента та його взаємодії з середовищем

Далі після налаштування фізичного середовища та фізичного тіла агента було розроблено кодову частину програмного модуля, що побудований за модульним принципом та розділений на окремі функціональні компоненти. Така структура дозволяє розмежувати логіку керування агентом, керування фізичними частинами тіла, обробку сенсорних даних та взаємодію з алгоритмом підкріплювального навчання.

Основними складовими програмної реалізації є компонент WalkerAgent, який виконує роль центрального координуючого елемента системи навчання, модуль фізичного тіла BodyPart та BodyController, що відповідає за керування фізичним тілом агента, компонент OrientationController для стабілізації просторових спостережень, а також сенсорні компоненти GroundCollision і TargetContact, які фіксують взаємодію агента з середовищем.

Кожен із зазначених модулів виконує окрему функцію в загальному циклі навчання: від збору стану середовища до формування керуючих дій та оцінки їх ефективності через функцію винагороди.

3.2.4.1 Центральний компонент агента WalkerAgent

Для реалізації поведінки інтелектуального агента було створено програмний клас WalkerAgent, який наслідується від базового класу Agent бібліотеки ML-Agents [33]. Основна логіка агента відповідає за збір даних про стан фізичного тіла, обробку дій нейронної мережі, взаємодію із фізичним середовищем та формування системи винагород.

На початку роботи агента виконується ініціалізація всіх основних компонентів тіла та службових систем. Для цього використовується метод Initialize(), у якому агент отримує посилання на контролер орієнтації OrientationController та контролер фізичних суглобів BodyController. Далі кожна частина тіла реєструється у системі керування фізикою через метод SetupBodyPart(). Це дозволяє агенту отримувати доступ до параметрів окремих сегментів тіла, зокрема швидкості, обертання та сили суглобів.

У структурі агента були визначені основні фізичні компоненти тіла: таз, грудна клітка, хребет, голова, руки та ноги. Кожна частина тіла представлена окремим об’єктом, що дозволяє виконувати незалежне керування суглобами під час навчання. Такий підхід формує фізично-коректну модель персонажа.

Після початку нового епізоду навчання виконується метод OnEpisodeBegin(). У ньому відбувається скидання фізичного стану всіх частин тіла до початкових значень, а також випадкова зміна початкового кута повороту

агента. Додатково може випадково змінюватися цільова швидкість руху, що дозволяє підвищити узагальнюючу здатність моделі та уникнути перенавчання лише під один сценарій руху.

Для навчання нейронної мережі агент формує набір спостережень через метод CollectObservations(). До вхідних даних моделі передаються:

- поточна середня швидкість руху агента;
- бажана швидкість пересування;
- напрямок до цілі;
- орієнтація голови та корпусу;
- інформація про положення кожної частини тіла;
- лінійна та кутова швидкість сегментів;
- факт контакту кінцівок із поверхнею.

Для зменшення шуму вхідних даних використовується спеціальний об’єкт стабілізації орієнтації, відносно якого обчислюються більшість просторових параметрів. Це дозволяє зробити процес навчання стабільнішим та покращити збіжність моделі.

Обробка дій нейронної мережі виконується у методі OnActionReceived(). Безперервні значення, які повертає модель PPO, інтерпретуються як:

- цільові кути обертання суглобів;
- сила, з якою суглоби повинні виконувати рух.

Для цього використовуються методи ApplyJointTargetRotations() та ApplyJointStrengths(). Перший відповідає за встановлення необхідного положення суглобів, а другий - за силу моторів фізичних з’єднань. Таким чином агент поступово навчається координувати рухи кінцівок для підтримки рівноваги та пересування.

Оцінювання якості поведінки агента виконується у методі FixedUpdate(), де формується функція винагороди. Основною складовою винагороди є відповідність фактичної швидкості руху заданій цільовій швидкості. Для цього використовується окремий метод ComputeVelocityMatchReward(), який обчислює різницю між бажаною та реальною швидкістю пересування.

Додатково враховується напрямок погляду агента відносно цілі. Якщо персонаж рухається у правильному напрямку та підтримує стабільне положення корпусу, підсумкова винагорода збільшується. Такий механізм стимулює модель не лише рухатися вперед, а й зберігати контрольовану та фізично правдоподібну поведінку.

У результаті реалізована система забезпечує повний цикл взаємодії між фізичним тілом агента, середовищем Unity та алгоритмом підкріплювального навчання PPO, що дозволяє формувати адаптивну поведінку автономної ходьби.

3.2.4.2 Взаємодія нейронної мережі із фізичним тілом агента

Для керування фізичним тілом агента у процесі навчання було реалізовано класи BodyPart та BodyController. Вони забезпечують взаємодію між нейронною мережею та фізичною системою Unity.

Клас BodyPart використовується для представлення окремої частини тіла агента. Кожен сегмент містить фізичне тіло та фізичне з’єднання.

Під час початку нового епізоду метод Reset() повертає кожну частину тіла у початковий стан. При цьому скидаються позиція, обертання, лінійна та кутова швидкість фізичних об’єктів. Також очищується інформація про контакти із поверхнею та цільовими об’єктами.

Одним із головних механізмів системи є метод SetJointTargetRotation(), який використовується для встановлення цільового кута обертання суглоба. Значення, отримані від нейронної мережі, нормалізуються та переводяться у фізично допустимі межі обертання конкретного суглоба. Після цього формується нова орієнтація суглоба, яка застосовується до фізичної моделі.

Метод SetJointStrength() відповідає за керування силою суглобів. На основі значення дії нейронної мережі обчислюється максимальна сила, яка може бути прикладена до конкретного сегмента тіла. Це дозволяє агенту навчатися не лише змінювати положення кінцівок, а й контролювати інтенсивність руху та підтримку балансу.

Клас BodyController виконує роль централізованої системи керування всіма сегментами тіла агента. У методі SetupBodyPart() виконується створення та налаштування кожної фізичної частини: отримання фізичних компонентів, збереження початкового стану, налаштування параметрів суглобів та додавання системи перевірки контакту із землею.

3.2.4.3 Налаштування файлу конфігурації навчання агента

Конфігураційний файл визначає параметри процесу навчання агента для задачі самостійної ходьби в середовищі Unity ML-Agents. Він задає алгоритм оптимізації, структуру нейронної мережі, параметри винагороди та загальні налаштування тренування, які впливають на стабільність і якість формування рухової поведінки, ці параметри передаються через інтерфейс Unity ML-Agents Toolkit до моделі, реалізованої у PyTorch [18, 33].

Пройдемося по основним параметрам, що напряму впливають на навчання агента.

trainertype - у нашому випадку використовується алгоритм проксимальна оптимізація політики, його вибір було обґрунтовано у розділі 2.3.2.

batchsize - визначає кількість досвіду, що використовується для одного оновлення нейронної мережі. У конфігурації значення 2048 забезпечує достатній обсяг даних для зменшення шуму градієнтів та стабільного навчання.

buffersize - загальна кількість накопичених кроків досвіду перед оновленням моделі. Значення 20480 дозволяє агенту навчатися на більш репрезентативних вибірках поведінки, що підвищує стабільність політики.

learningrate - визначає швидкість оновлення ваг нейронної мережі. Значення 0.0003 забезпечує баланс між швидкістю навчання та стабільністю збіжності.

learningrateschedule - задає зміну швидкості навчання в часі. Використовується linear, що означає поступове зменшення learning rate до кінця навчання для більш стабільної фінальної політики.

beta - коефіцієнт ентропії, який регулює рівень випадковості дій агента. Значення 0.005 підтримує дослідження простору дій на ранніх етапах навчання.

epsilon - обмежує зміну політики між оновленнями. Значення 0.2 забезпечує стабільність навчання та запобігає різким змінам поведінки.

lambd - параметр Generalized Advantage Estimation, який визначає баланс між зміщенням і дисперсією оцінки винагород. Значення 0.95 забезпечує стабільне оцінювання переваг дій.

numepoch - кількість проходів по одному набору досвіду під час градієнтного оновлення. Значення 3 забезпечує компроміс між якістю навчання та його швидкістю.

normalize - вмикає нормалізацію вхідних спостережень. Використання true покращує стабільність навчання в задачах, де вхідні фізичні параметри мають різні масштаби.

hiddenunits - кількість нейронів у прихованих шарах мережі. Значення 256 дозволяє моделі навчати достатньо складні залежності між станом тіла та діями.

numlayers - кількість прихованих шарів нейронної мережі. Значення 3 забезпечує достатню глибину для моделювання складної поведінки ходьби.

visencodetype - тип енкодера візуальних даних. Використовується simple, що представляє базову згорткову архітектуру для обробки зображень.

gamma - коефіцієнт дисконтування майбутніх винагород. Значення 0.995 означає, що агент враховує довгострокові наслідки своїх дій.

strength - масштабування винагороди середовища. Значення 1.0 означає, що винагорода використовується без додаткового масштабування.

keepcheckpoints - кількість збережених контрольних точок моделі. Значення 5 дозволяє зберігати останні версії політики для аналізу або відновлення.

maxsteps - загальна кількість кроків навчання. Значення 30000000 забезпечує достатній обсяг досвіду для формування стабільної ходьби.

timehorizon - кількість кроків, які агент враховує перед додаванням досвіду в буфер. Значення 1000 дозволяє враховувати довготривалі залежності руху.

summaryfreq - частота запису статистики навчання. Значення 30000 використовується для моніторингу процесу через TensorBoard.

3.3 Результати експериментів та їх аналіз

Результати навчання агентів у середовищі Unity ML-Agents були проаналізовані за допомогою інструментів TensorBoard [25], що дозволяє відстежувати динаміку ключових метрик процесу навчання, зокрема сумарну винагороду, тривалість епізодів та функції втрат. Тестування проводилось для двох незалежних тренувань: firstTrainer рожевого кольору та trainerWalker32 синього кольору. Розглянемо результати тестування.

Показник сумарної винагороди, його графіки відображено на рисунку 3.3, демонструє суттєву різницю між моделями. Агент firstTrainer після приблизно 5 мільйонів кроків починає стрімко покращувати результат і досягає значень близько 1000, що свідчить про успішне засвоєння політики стабільної ходьби та ефективного пересування. Водночас trainerWalker32 демонструє низькі значення винагороди, менше 100, що вказує на недостатнє навчання та відсутність сформованої стратегії руху.

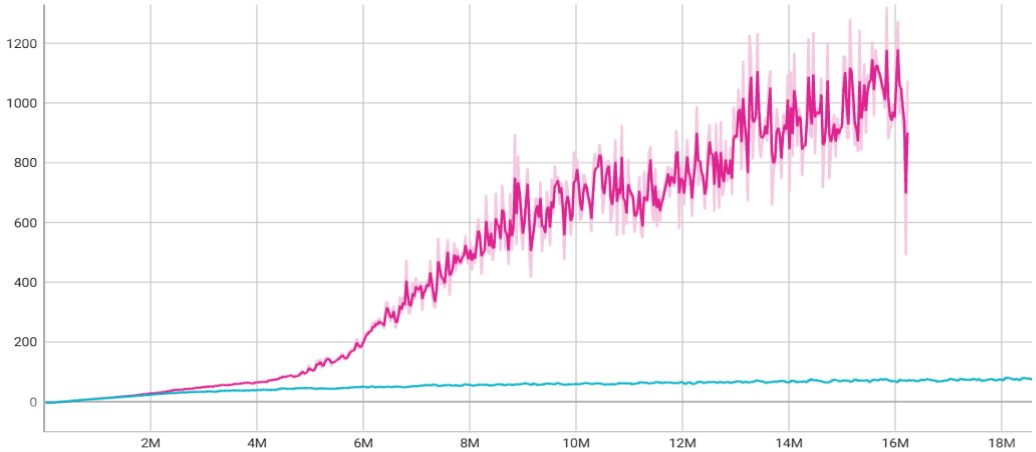

Рисунок 3.3 – Графік сумарної винагороди агента

Аналіз графіків довжини епізоду, що відображені на рисунку 3.4, підтверджує ці результати. У випадку firstTrainer тривалість епізодів поступово

зростає до 500 кроків, що означає здатність агента підтримувати баланс і здійснювати довготривалий рух без падіння. Для trainerWalker32 характерні короткі епізоди, що свідчить про часті втрати рівноваги та нестабільну поведінку.

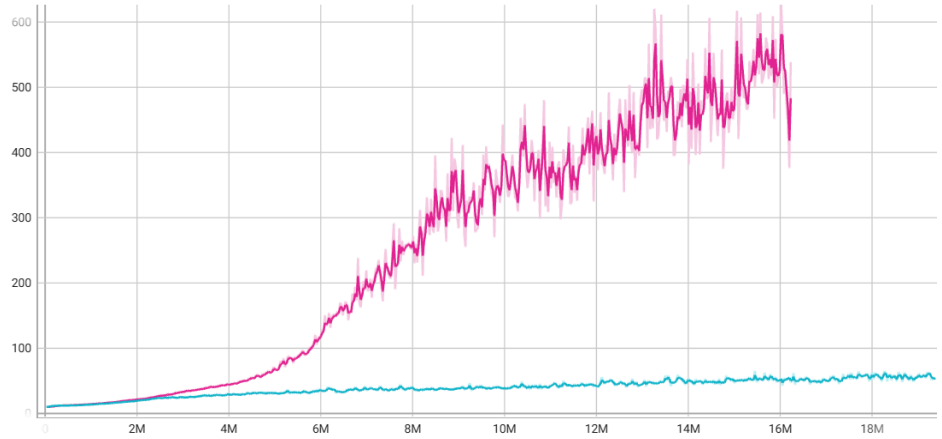

Рисунок 3.4 – Графік довжини епізоду

Метрика динаміки накопичувальної винагороди агента, графік, якої відображений на рисунку 3.5, показує якість прогнозування майбутньої винагороди. Для firstTrainer спостерігається початковий ріст помилки з подальшим зниженням, що відповідає етапу активного навчання та подальшого уточнення оцінки станів середовища. Це свідчить про покращення здатності моделі прогнозувати довгострокову винагороду в процесі навчання.

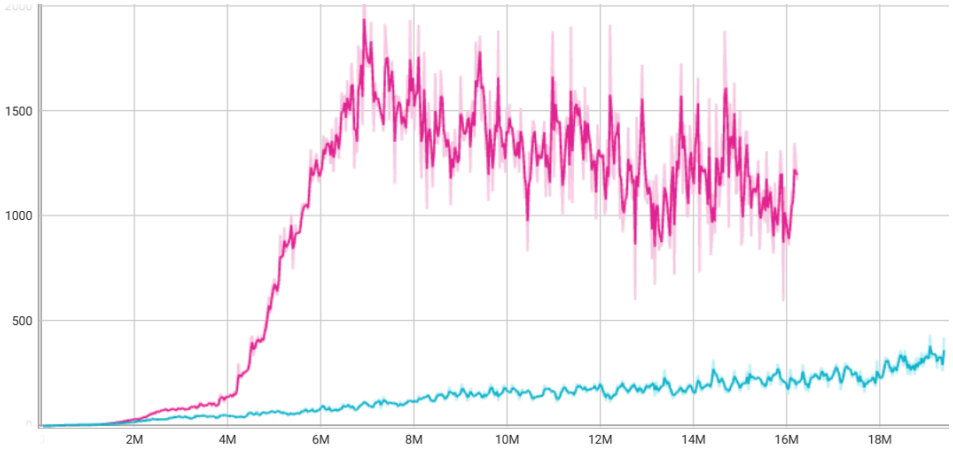

Рисунок 3.5 – Графік динаміки накопичувальної винагороди агента

Різниця двох моделей у їх способах ходьби. Модель firstTrainer знайшла оптимальніший, але специфічний спосіб ходьби. Агент рухався способом ковзання на одній нозі. Тобто одна нога завжди була на підлозі і тримала баланс агента, а інша нога штовхала все тіло до цілі. Такий спосіб руху був можливий через неправильну конфігурацію середовища, а саме недостатньо великий коефіцієнт тертя підлоги. Через що агент мав змогу водночас і ковзати на одній нозі по підлозі і штовхати тіло до цілі іншою ногою. Дана проблема була вирішена для навчання наступної моделі trainerWalker32. Тому у неї не такі великі показники винагород, як у firstTrainer, але ходьба виглядає більш правильною.

У цілому результати демонструють, що модель firstTrainer досягла збіжності та сформувала стабільну, проте некоректну політику ходьби, тоді як trainerWalker32 залишилась на етапі часткового навчання. Причинами різниці є випадкова ініціалізації ваг нейронної мережі та різні сценарії навчання.

Отримані результати узгоджуються з підходами навчання агентів у Unity ML-Agents, де ефективність політики напряму залежить від налаштування середовища, функції винагороди та архітектури нейронної мережі [1], [5].

ВИСНОВКИ

У ході виконання кваліфікаційної роботи здійснено комплекс досліджень і розробок, спрямованих на реалізацію та аналіз процесу навчання автономного агента в середовищі Unity. Отримані результати дали змогу сформувати уявлення про побудову фізичної симуляції та застосування методів машинного навчання для керування поведінкою агента.

Під час виконання кваліфікаційної роботи зроблено такі висновки:

1. Проведено аналіз можливостей середовища Unity для реалізації фізичної симуляції та навчання агентів, зокрема розглянуто механізми фізичного рушія, компоненти взаємодії об’єктів та інтеграцію з інструментами машинного навчання.

2. Сформульовано постановку задачі навчання агента самостійній ходьбі у фізично-реалістичному середовищі. Визначено основні вимоги до агента, середовища та критерії успішного навчання.

3. Розроблено модуль прийняття рішень на основі нейронної мережі з використанням Unity ML-Agents, що забезпечує обробку спостережень із середовища та формування керуючих дій агента в реальному часі.

4. Проведено навчання агента за допомогою алгоритму Proximal Policy Optimization, який дозволяє ефективно оптимізувати політику поведінки агента через взаємодію із симульованим середовищем та отримання винагороди за виконання цільових дій.

5. Виконано аналіз результатів навчання за допомогою TensorBoard. Досліджено динаміку зміни функції винагороди, стабільність навчання та ефективність різних конфігурацій моделей, а також проведено порівняння отриманих результатів.

6. Підготовлено графічний матеріал, що включає візуалізацію фізичних компонентів у середовищі Unity, демонстрацію фізичного середовища та структури тіла агента, блок-схеми алгоритму навчання агента самостійної ходьби, а також графіки результатів навчання моделі, отримані з TensorBoard.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agrobot. E-Series Strawberry Harvester. URL: https://www.agrobot.com/e-series (дата звернення: 29.04.2026).
2. Anaconda, Inc. Anaconda Distribution Installer. URL: https://www.anaconda.com/download (дата звернення: 14.03.2026).
3. Brockman G., Cheung V., Pettersson L. et al. OpenAI Gym // arXiv preprint arXiv:1606.01540. 2016.
4. Correll N., Bekris K. E., Berenson D. et al. Introduction to Autonomous Robots. — 2nd ed. — Cambridge : MIT Press, 2022. — 600 p.
5. Cyberbotics. Webots Robot Simulator Documentation. URL: https://cyberbotics.com/ (дата звернення: 11.04.2026).
6. DJI. Agras T40. URL: https://www.dji.com/t40 (дата звернення: 11.05.2026).
7. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 775 p.
8. DeepMind. MuJoCo Documentation. URL: MuJoCo Official Website (дата звернення: 12.04.2026).
9. John Deere. Autonomous Tractor. URL: https://www.deere.com/en/autonomous (дата звернення: 29.04.2026).
10. Janner M., Fu J., Zhang M., Levine S. When to Trust Your Model: Model-Based Policy Optimization // arXiv:1906.08253. 2019. https://arxiv.org/abs/1906.08253
11. Juliani A., Berges V.-P., Vckay E. et al. Unity: A General Platform for Intelligent Agents // arXiv preprint arXiv:1809.02627. 2018. https://arxiv.org/abs/1809.02627
12. Kober J., Bagnell J.A., Peters J. Reinforcement Learning in Robotics: A Survey // International Journal of Robotics Research. 2013. Vol. 32(11). P. 1238–1274.
13. Konda V. R., Tsitsiklis J. N. On Actor-Critic Algorithms // *SIAM Journal on Control and Optimization*. 2003. Vol. 42(4). P. 1143–1166. URL:

https://epubs.siam.org/doi/10.1137/S0363012901385691 (дата звернення: 11.04.2026).

14. Microsoft. Visual Studio Documentation. URL: https://learn.microsoft.com/visualstudio (дата звернення: 06.03.2026).

15. Naïo Technologies. Oz Weeding Robot. URL: https://www.naio-technologies.com/en/oz (дата звернення: 29.04.2026).

16. NVIDIA. CUDA Toolkit Documentation. URL: NVIDIA CUDA Toolkit (дата звернення: 11.04.2026).

17. NVIDIA. Isaac Sim Documentation. URL: NVIDIA Isaac Sim (дата звернення: 12.04.2026).

18. PyTorch Foundation. PyTorch Documentation. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 12.03.2026).

19. Python Software Foundation. Python Documentation. URL: https://docs.python.org/3/ (дата звернення: 12.03.2026).

20. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Pearson, 2021. 1136 p.

21. Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal Policy Optimization Algorithms // arXiv:1707.06347. https://arxiv.org/abs/1707.06347 (дата звернення: 12.04.2026).

22. Silver D., Hubert T., Schrittwieser J. et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm // arXiv preprint arXiv:1712.01815. 2017.

23. Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. 2nd ed. Cambridge, MA : MIT Press, 2018. 552 p.

24. Siciliano B., Sciavicco L., Villani L., Oriolo G. Robotics: Modelling, Planning and Control. — London : Springer, 2009. — 632 p.

25. TensorBoard. Official Documentation. URL: https://www.tensorflow.org/tensorboard (дата звернення: 11.03.2026).

26. Tobin J., Fong R., Ray A. et al. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World // arXiv:1703.06907. 2017. https://arxiv.org/abs/1703.06907

27. Unity Technologies. Animation Rigging Package Documentation. URL: https://docs.unity3d.com/Packages/com.unity.animation.rigging (дата звернення: 07.03.2026).

28. Unity Technologies. Barracuda Neural Network Inference Library Documentation. URL: https://docs.unity3d.com/Packages/com.unity.barracuda (дата звернення: 09.03.2026).

29. Unity Technologies. Configurable Joint Component Reference. URL: https://docs.unity3d.com/Manual/class-ConfigurableJoint.html (дата звернення: 04.03.2026).

30. Unity Technologies. ML-Agents Examples Documentation. URL: https://github.com/Unity-Technologies/ml-agents/tree/main/Project (дата звернення: 04.03.2026).

31. Unity Technologies. Unity Collider Components Documentation. URL: https://docs.unity3d.com/Manual/CollidersOverview.html (дата звернення: 04.03.2026).

32. Unity Technologies. Unity Hub Documentation. URL: https://docs.unity.com/en-us/hub (дата звернення: 04.03.2026).

33. Unity Technologies. Unity ML-Agents Toolkit Documentation. URL: https://github.com/Unity-Technologies/ml-agents (дата звернення: 11.05.2026).

34. Unity Technologies. Unity Official Website. URL: https://unity.com (дата звернення: 04.03.2026).

35. Unity Technologies. Unity Physics and Rigidbody Components Documentation. URL: https://docs.unity3d.com/Manual/class-Rigidbody.html (дата звернення: 04.03.2026).

36. Лубко Д. В., Шаров О. І. Методи та системи штучного інтелекту : навчальний посібник. — Таврійський державний агротехнологічний університет

імені Дмитра Моторного. URL: https://elar.tsatu.edu.ua/handle/123456789/7618 (дата звернення: 14.05.2026).

37. Ужгородський національний університет. Методи та системи штучного інтелекту : конспект лекцій. URL: https://dspace.uzhnu.edu.ua/jspui/handle/lib/63178 (дата звернення: 14.05.2026).

38. Запорізький національний технічний університет. Інтелектуальні системи : навчальний посібник. URL: https://eir.zp.edu.ua/items/72604548-8d38-4ab4-9093-4e3f17f08c31 (дата звернення: 14.05.2026).

39. Запорізький національний технічний університет. Нейронні мережі : навчальний посібник. URL: https://eir.zp.edu.ua/items/84a582c5-c596-4083-b9c4-41f5381fc136 (дата звернення: 14.05.2026).

40. Яремчук В. М., Лендюк Т. В. Програмний модуль інтелектуального агента для самостійної ходьби персонажа в середовищі Unity // Інтелектуальні обчислювальні системи та управління проєктами (ICSPM 2026) : матеріали I Міжнародної науково-практичної конференції студентів та молодих учених, 21–22 травня 2026 р. Тернопіль : Західноукраїнський національний університет, 2026. С. 116–120.

41. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Алгоритми роботи інтелектуального агенту для самостійної ходьби

Рисунок А.1 – Узагальнений алгоритм виконання дії агентом

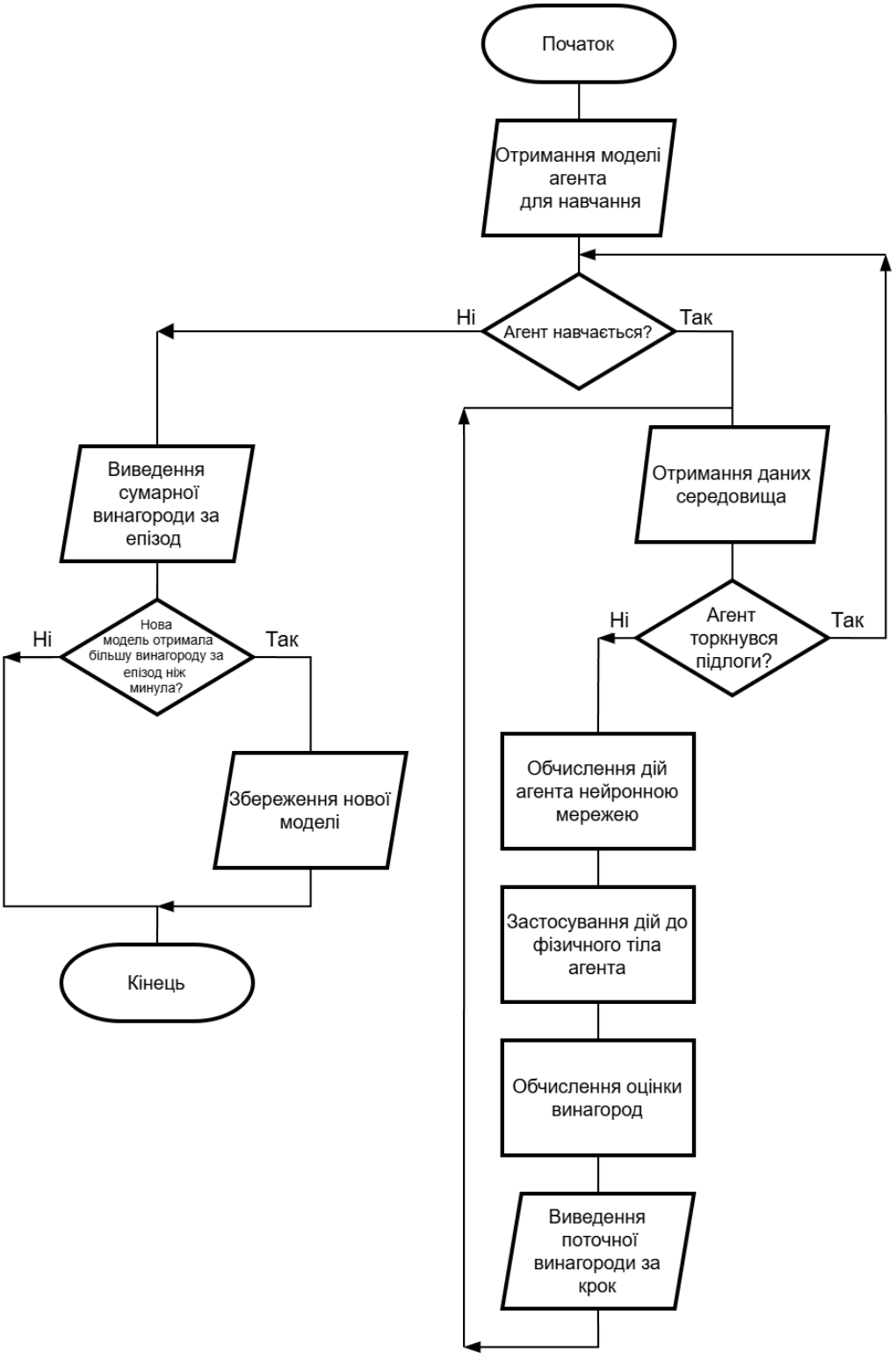

Рисунок А.2 – Алгоритм навчання агента самостійної ходьби

Додаток Б

Копія публікації

Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

Матеріали

І Міжнародної науково-практичної конференції студентів і молодих вчених

«ICSPM 2026: Intelligent Computing Systems and Project Management / Інтелектуальні обчислювальні системи та управління проєктами»

21–22 травня 2026 р.

м. Тернопіль - 2026

УДК 004.8:005.8](063)

Присвячено 60-річчю Західноукраїнського національного університету

ICSPM 2026: Інтелектуальні обчислювальні системи та управління проєктами : збірник тез доповідей І Міжнародної науково-практичної конференції студентів і молодих вчених (Тернопіль, 21–22 травня 2026 року). – Тернопіль : ЗУНУ, 2026. – 190 с.

До збірника увійшли тези доповідей учасників І Міжнародної науково-практичної конференції студентів і молодих вчених «ICSPM 2026: Intelligent Computing Systems and Project Management / Інтелектуальні обчислювальні системи та управління проєктами», що відбулася на базі кафедри інформаційно-обчислювальних систем і управління факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету.

Матеріали відображають результати досліджень за такими напрямами:

1. Інтелектуальні обчислення та програмні системи.
2. Проєктно-орієнтоване управління та процеси прийняття рішень.
3. Штучний інтелект, аналітика даних і кібернетика.
4. Прикладні технології та інформаційно-комунікаційні системи.
5. Сучасні інформаційні технології в економіці, праві та освіті.

Рекомендовано до друку на засіданні кафедри інформаційно-обчислювальних систем і управління Західноукраїнського національного університету (протокол №12 від 26.05.2026 р.).

За зміст наукових праць, точність наведених цитувань, перекладу та достовірність фактологічних матеріалів відповідальність несуть автори публікацій. У збірнику збережено авторську стилістику, пунктуацію та орфографію.

ЗМІСТ

УДК 004.8 : 004.94 : 004.85

ПРОГРАМНИЙ МОДУЛЬ ІНТЕЛЕКТУАЛЬНОГО АГЕНТА ДЛЯ САМОСТІЙНОЇ ХОДЬБИ ПЕРСОНАЖА В СЕРЕДОВИЩІ UNITY

Яремчук Владислав Миколайович, студент,
yuramchuk2016@gmail.com

Науковий керівник: Лендюк Тарас Васильович,
к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем,
Західноукраїнський національний університет
tl@wunu.edu.ua,
ORCID: 0000-0001-9484-8333

Сучасні дослідження штучного інтелекту активно використовують симуляційні середовища для навчання автономних агентів руху. У середовищі Unity це реалізується через фізичну симуляцію об’єктів, включно з гравітацією, тертям і зіткненнями, що дозволяє відтворювати реалістичну поведінку агентів [2, 3].

Ходячі роботи можуть застосовуватись у логістиці, агросекторі та сервісних задачах: переміщення вантажів, інспекція територій, навігація в складних приміщеннях. Їхня перевага — здатність працювати на нерівних поверхнях і в обмежених просторах, де колісні системи малоефективні.

Підхід Unity ML-Agents дає змогу навчати агентів самостійно формувати стратегії ходьби через взаємодію з середовищем і системою винагород [1]. Для інференсу нейронних моделей у Unity використовується бібліотека Barracuda, що забезпечує виконання нейромереж без зовнішніх сервісів [4].

Використання навчання з підкріпленням дозволяє агентам адаптуватися до змін середовища, оптимізуючи баланс, швидкість і стабільність руху. У результаті формуються моделі автономної ходьби, які не потребують ручного програмування правил руху.

Метою розробленої платформи є навчання агента автономній ходьбі у фізично-реалістичному середовищі Unity із використанням методів навчання з підкріпленням. Система побудована як інтеграція фізичної симуляції, моделі агента та алгоритмів машинного навчання.

Фізичне середовище реалізоване у Unity та базується на використанні стандартного фізичного рушія, який моделює гравітацію, тертя та взаємодію тіл через компоненти Rigidbody і Collider [2, 3]. Сцена включає опорну поверхню, фізичні матеріали та, за необхідності, перешкоди, що дозволяє створювати умови для навчання стабільної та адаптивної ходьби.

Агент представлений як багатосегментна структура, де кожна частина тіла має власний Rigidbody, а з’єднання між ними реалізовані через Joint-компоненти, структуру агента відображено на рисунку 1. Така модель дозволяє відтворювати

рухи, підтримувати баланс і реагувати на фізичні сили відповідно до законів механіки [2, 3].

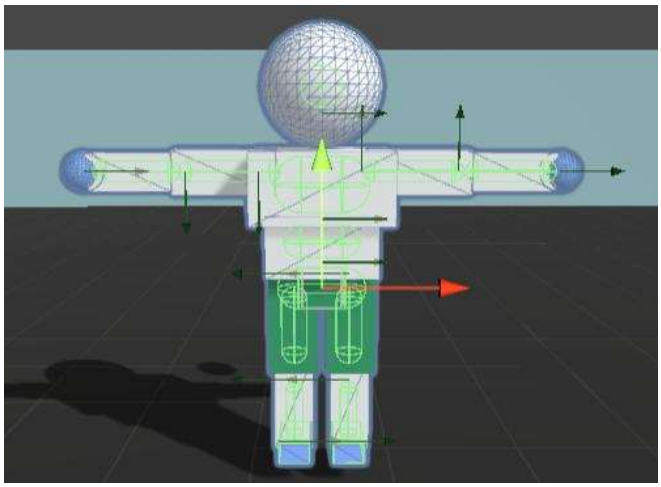

Рисунок 1 – Вигляд тіла агента у середовищі Unity

Збір даних про стан агента здійснюється через модуль спостереження, який формує вектор параметрів, що включає положення, швидкість, орієнтацію та контакт із поверхнею. Ці дані передаються до моделі прийняття рішень.

Модуль прийняття рішень реалізований на основі нейронної мережі, яка формує керуючі сигнали для руху агента. Для інтеграції алгоритмів навчання використовується Unity ML-Agents Toolkit, що забезпечує взаємодію між середовищем та моделлю штучного інтелекту [1]. Виконання навчених моделей у середовищі Unity здійснюється за допомогою бібліотеки Barracuda [5].

Навчання агента відбувається із застосуванням алгоритму Proximal Policy Optimization (PPO), який дозволяє поступово покращувати політику дій на основі отриманої винагороди [1]. Винагорода формується з урахуванням кількох факторів: збереження балансу, руху вперед, енергоефективності та штрафів за падіння. Це дозволяє агенту самостійно формувати ефективні та стабільні патерни ходьби без явного програмування рухів.

Таким чином, запропонована система поєднує фізичну симуляцію, нейромережеві моделі та алгоритми навчання з підкріпленням, що забезпечує формування автономної поведінки агента у складному динамічному середовищі [1, 5].

Для формалізації задачі використано декілька основних математичних підходів. Обчислення винагороди агента на кожному кроці наведено у наступній формулі:

$$R(s_t, a_t) = w_b \cdot B(s_t) + w_s \cdot S(s_t, a_t) - w_e \cdot E(a_t) \ , \qquad (1)$$

де $R(s_t, a_t)$— винагорода агента у стані s_t при дії a_t;
$B(s_t)$ — показник підтримки балансу;
$S(s_t, a_t)$ — ефективність кроку (швидкість та прогрес вперед);
$E(a_t)$ — витрати енергії на виконання дії;

w_b, w_s, w_e — вагові коефіцієнти для балансування компонентів винагороди.

Кумулятивна винагорода визначається як сума дисконтованих винагород [5]:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k} \tag{2}$$

де R_{t+k}— винагорода на кожному кроці епізоду;

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

G_t — накопичена (дисконтована) винагорода за епізод.

Функція оптимізації політики агента, відповідно до рівняння Беллмана оптимальності [5], визначає процес оновлення параметрів моделі під час навчання:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \tag{3}$$

де π^* — оптимальна політика поведінки агента;

γ — коефіцієнт дисконтування, а сумування ведеться по всіх кроках епізоду T.

Наступна формула описує процес переходу системи у новий стан під впливом дій агента у фізичному середовищі, де стан системи включає параметри, що описують положення, швидкість, орієнтацію агента та його взаємодію із середовищем:

$$s_{t+1} = f_{\text{phys}}(s_t, a_t) \tag{4}$$

де f_{phys} — фізична модель середовища: динаміка суглобів, сила тяжіння, контакт з поверхнею (гарантує, що дії агента призводять до реалістичного пересування та збереження балансу).

У рамках дослідження алгоритм навчання агента використовує ці функції для поступового покращення політики руху. Нейронна мережа прогнозує оптимальні дії на основі поточного стану агента та навколишнього середовища, а система винагороди забезпечує коригування поведінки для стабільної і ефективної ходьби.

Результати навчання агентів у середовищі Unity ML-Agents були проаналізовані за допомогою інструментів TensorBoard, що дозволяє відстежувати динаміку ключових метрик процесу навчання, зокрема сумарну винагороду, тривалість епізодів та функції втрат. Тестування проводилось для двох незалежних тренувань: firstTrainer рожевого кольору та trainerWalker32 синього кольору. Розглянемо результати тестування.

Показник сумарної винагороди, його графіки відображено на рисунку 2, демонструє суттєву різницю між моделями. Агент firstTrainer після приблизно 5 мільйонів кроків починає стрімко покращувати результат і досягає значень близько 1000, що свідчить про успішне засвоєння політики стабільної ходьби та ефективного пересування. Водночас trainerWalker32 демонструє низькі значення

винагороди, менше 100, що вказує на недостатнє навчання та відсутність сформованої стратегії руху [1].

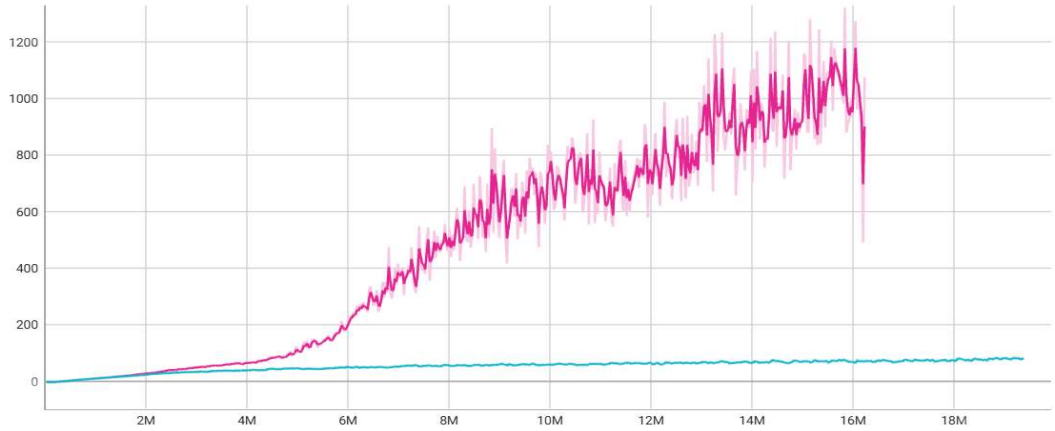

Рисунок 2 – Графік динаміки накопичувальної винагороди агента

Аналіз графіків довжини епізоду, що відображені на рисунку 3, підтверджує ці результати. У випадку firstTrainer тривалість епізодів поступово зростає до 500 кроків, що означає здатність агента підтримувати баланс і здійснювати довготривалий рух без падіння. Для trainerWalker32 характерні короткі епізоди, що свідчить про часті втрати рівноваги та нестабільну поведінку [1].

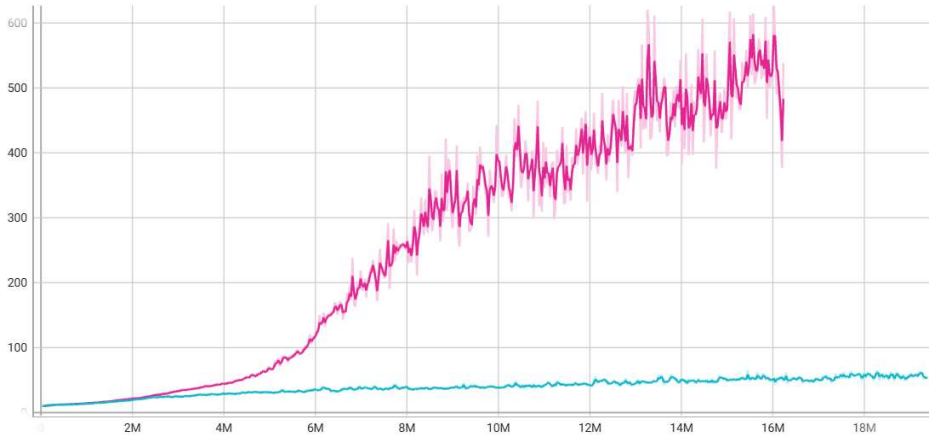

Рисунок 3 – Графік довжини епізоду

Метрика динаміки накопичувальної винагороди агента, графік, якої відображений на рисунку 4, показує якість прогнозування майбутньої винагороди. Для firstTrainer спостерігається початковий ріст помилки з подальшим зниженням, що відповідає етапу активного навчання та подальшого уточнення оцінки станів середовища. Це свідчить про покращення здатності моделі прогнозувати довгострокову винагороду в процесі навчання [1].

Різниця двох моделей у їх способах ходьби. Модель firstTrainer знайшла оптимальніший, але специфічний спосіб ходьби. Агент рухався способом ковзання на одній нозі. Тобто одна нога завжди була на підлозі і тримала баланс агента, а інша нога штовхала все тіло до цілі.

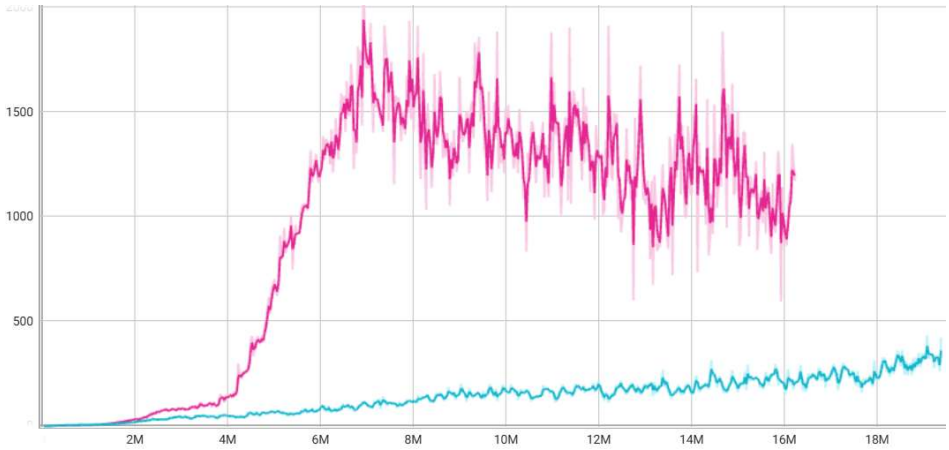

Рисунок 4 – Графік динаміки накопичувальної винагороди агента

Такий спосіб руху був можливий через неправильну конфігурацію середовища, а саме недостатньо великий коефіцієнт тертя підлоги. Через що агент мав змогу водночас і ковзати на одній нозі по підлозі і штовхати тіло до цілі іншою ногою. Дана проблема була вирішена для навчання наступної моделі trainerWalker32. Тому у неї не такі великі показники винагород, як у firstTrainer, але ходьба виглядає більш правильною.

У цілому результати демонструють, що модель firstTrainer досягла збіжності та сформувала стабільну, проте некоректну, політику ходьби. Тоді як trainerWalker32 залишилась на етапі часткового навчання. Причинами різниці є випадкова ініціалізації ваг нейронної мережі та різні сценарії навчання.

Отримані результати узгоджуються з підходами навчання агентів у Unity ML-Agents, де ефективність політики напряму залежить від налаштування середовища, функції винагороди та архітектури нейронної мережі [1], [5].

Список використаних джерел

1. Unity Technologies. Unity ML-Agents Toolkit Documentation. Unity Technologies, 2024. https://github.com/Unity-Technologies/ml-agents

2. Unity Technologies. Unity Physics and Rigidbody Components Documentation. Unity Manual, 2024. https://docs.unity3d.com/Manual/class-Rigidbody.html

3. Unity Technologies. Unity Collider Components Documentation. Unity Manual, 2024. https://docs.unity3d.com/Manual/CollidersOverview.html

4. Unity Technologies. Barracuda Neural Network Inference Library Documentation. Unity Manual, 2024. https://docs.unity3d.com/Packages/com.unity.barracuda

5. Reinforcement Learning: An Introduction / Richard S. Sutton, Andrew G. Barto. 2nd ed. Cambridge, MA: MIT Press, 2018. 552 p.

винагороди, менше 100, що вказує на недостатнє навчання та відсутність сформованої стратегії руху [1].

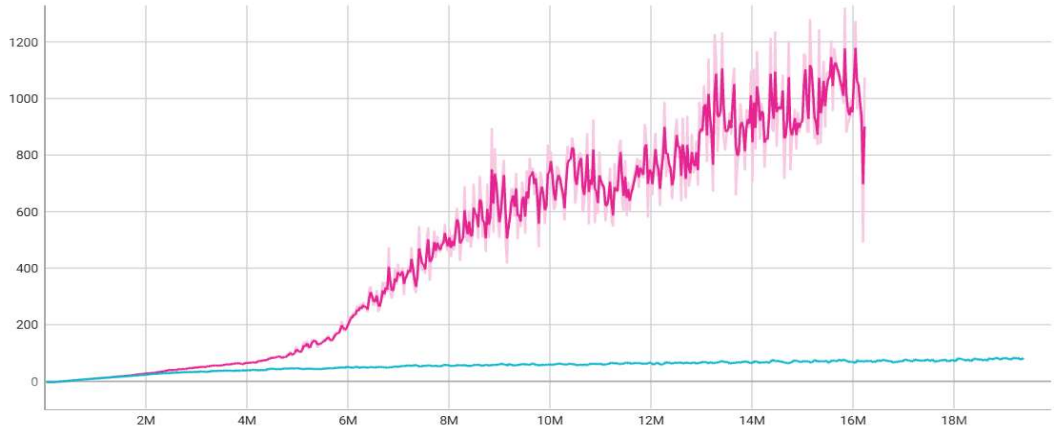

Рисунок 2 – Графік динаміки накопичувальної винагороди агента

Аналіз графіків довжини епізоду, що відображені на рисунку 3, підтверджує ці результати. У випадку firstTrainer тривалість епізодів поступово зростає до 500 кроків, що означає здатність агента підтримувати баланс і здійснювати довготривалий рух без падіння. Для trainerWalker32 характерні короткі епізоди, що свідчить про часті втрати рівноваги та нестабільну поведінку [1].

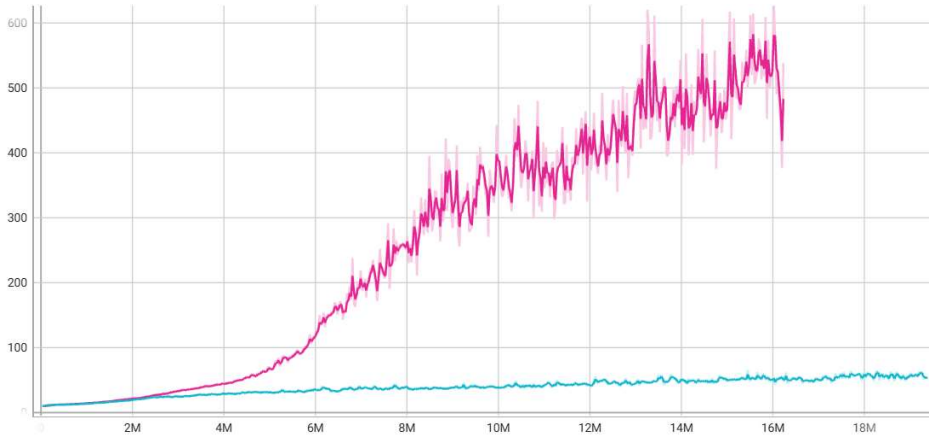

Рисунок 3 – Графік довжини епізоду

Метрика динаміки накопичувальної винагороди агента, графік, якої відображений на рисунку 4, показує якість прогнозування майбутньої винагороди. Для firstTrainer спостерігається початковий ріст помилки з подальшим зниженням, що відповідає етапу активного навчання та подальшого уточнення оцінки станів середовища. Це свідчить про покращення здатності моделі прогнозувати довгострокову винагороду в процесі навчання [1].

Різниця двох моделей у їх способах ходьби. Модель firstTrainer знайшла оптимальніший, але специфічний спосіб ходьби. Агент рухався способом ковзання на одній нозі. Тобто одна нога завжди була на підлозі і тримала баланс агента, а інша нога штовхала все тіло до цілі.