МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

Магістр

на тему:

«Моделювання поведінки віртуальних агентів для виконання колективних дій у імітаційному середовищі з використанням методів машинного навчання з підкріпленням»

Виконав: студент 4 курсу, групи МФ-61

спеціальність 122 Комп’ютерні науки

освітньо-наукова програма

«Інформатика»

Чантурія Георгій Нукрійович

Керівник ____ Меняйлов Є.С._____

Рецензент_____

Харків – 2025 року

ЗМІСТ

1.Вступ

1.1 Формулювання мети роботи, задач та обґрунтування актуальності теми.

Одним із ключових викликів у сфері штучного інтелекту є створення адаптивних агентів, здатних до автономного прийняття рішень в умовах динамічних середовищ. У контексті багатоагентних систем особливу складність становить забезпечення ефективної колективної взаємодії, де поведінка кожного агента не лише залежить від стану середовища, а й від дій інших учасників. Вирішення подібних задач має важливе значення для широкого кола прикладних сфер: від координації автономних транспортних засобів до симуляцій соціальної поведінки, екологічного моделювання та розробки інтелектуальних ігрових систем.

Навчання з підкріпленням (Reinforcement Learning, RL) забезпечує гнучкий інструментарій для розв’язання задач оптимізації поведінки агентів без потреби в заздалегідь визначених правилах. У поєднанні з багатоагентною взаємодією (Multi-Agent Reinforcement Learning, MARL), цей підхід відкриває нові можливості для дослідження складних форм колективної поведінки — таких як співпраця, конкуренція, розподіл ролей тощо.

Актуальність теми також підсилюється стрімким розвитком симуляційних платформ (на кшталт Unity ML-Agents, OpenAI Gym, PettingZoo), які дозволяють тестувати та валідувати моделі в умовах, максимально наближених до реальних.

Метою даної роботи є розробка, реалізація та емпіричне дослідження моделі поведінки віртуальних агентів, орієнтованої на виконання колективних дій у симульованому середовищі, з використанням алгоритмів навчання з підкріпленням.

1.2 Стислий огляд відомих результатів

Моделювання поведінки віртуальних агентів, зокрема в умовах колективної взаємодії, є предметом активних наукових досліджень упродовж останніх десятиліть. Традиційно, задачі кооперації та координації в багатокомпонентних системах вирішувалися за допомогою формальних підходів — таких як теорія ігор, логіка БДІ (belief-desire-intention), агентно-орієнтоване програмування тощо. Однак ці підходи виявилися обмеженими в умовах динамічних або частково спостережуваних середовищ.

Поява глибокого навчання з підкріпленням (Deep Reinforcement Learning, DRL) дала змогу суттєво розширити можливості агентів завдяки поєднанню нейронних мереж із методами оптимізації політики. Одним із найбільш відомих проривів у цій сфері стала система AlphaGo компанії DeepMind (2016), яка навчилася перемагати чемпіонів світу в грі го, використовуючи комбінацію дерева Монтекарло та глибокої нейронної мережі.

У сфері багатоагентних систем (Multi-Agent Systems, MAS) важливим кроком стало запровадження Multi-Agent Reinforcement Learning (MARL) — напрямку, що дозволяє агентам навчатися не тільки в ізольованих умовах, а й у постійній взаємодії з іншими суб’єктами середовища. У цьому контексті широко досліджуються такі проблеми, як стабільність навчання, стратегічна адаптація, узгодження політик та розподілення винагород. До найвідоміших алгоритмів MARL відносяться Independent Q-learning (IQL), Centralized Training with Decentralized Execution (CTDE), MADDPG (Multi-Agent Deep Deterministic Policy Gradient) та QMIX.

Сучасні дослідження активно використовують симуляційні середовища, які дозволяють відтворювати складні сценарії багатокомпонентної взаємодії. Такі платформи забезпечують гнучкий інструментарій для тестування агентних моделей у контрольованих умовах, сприяючи стандартизації підходів до

оцінювання результатів. Їх застосування дозволяє не лише зменшити витрати на реальні експерименти, але й пришвидшити ітерації навчання завдяки повній керованості параметрів середовища.

Певні роботи присвячені моделюванню соціальної поведінки агентів: координації, консенсусу, формуванню лідерства чи розподілу ролей. Також активно досліджуються сценарії з передачею навичок (transfer learning) між агентами, що підвищує ефективність навчання в гетерогенних колективах.

Таким чином, на сьогодні сформувалося потужне міждисциплінарне підґрунтя, яке поєднує методи машинного навчання, системного моделювання, поведінкової економіки та когнітивної науки. Водночас залишається низка нерозв'язаних проблем, зокрема пов'язаних із інтерпретованістю поведінки агентів, забезпеченням стабільного колективного результату за відсутності централізованого контролю та ефективністю в умовах обмежених ресурсів середовища.

1.3Відомості про одержані результати та їх новизна

У ході виконання роботи була реалізована інтерактивна симуляційна система, що моделює колективну поведінку кількох віртуальних агентів у тривимірному середовищі з використанням підходів навчання з підкріпленням (Reinforcement Learning, RL). В основі симуляції лежить кооперативне завдання, у якому агенти повинні спільними зусиллями перемістити об'єкти (блоки) до визначеної цільової зони.

Реалізована система включає декілька взаємодіючих компонентів. Агентів, які здатні самостійно пересуватись, обирати напрямки руху та адаптувати стратегію відповідно до змін у середовищі. Блоки різних розмірів, що служать об'єктами маніпуляції для агентів. Цільову зону, куди необхідно

доставити об’єкти, керуючий контролер середовища, який координує епізоди, ініціює перезапуски, обробляє події зіткнень і розподіляє винагороду між агентами. Інтерфейс візуалізації прогресу, що в реальному часі відображає зміну нагород і номер поточного епізоду.

Система реалізована на рушії Unity з використанням бібліотеки ML-Agents, що дозволяє застосовувати сучасні алгоритми глибокого навчання з підкріпленням у складних 3D-середовищах.

Агенти діють на основі дискретної просторової політики (рух вперед/назад, обертання, бічні переміщення), яка формується за допомогою моделі, навченого алгоритмом RL. Особливістю є динамічна оцінка винагороди не лише за досягнення мети, але й за наближення до цілі (через зменшення відстані), що стимулює більш ефективну траєкторію.

Досягнуті результати включають:

1) Успішне навчання агентів базовим кооперативним діям
2) Адаптивна реакція агентів на змінне середовище
3) Стабільна групова поведінка в умовах випадкової ініціалізації (позиції агентів і блоків)
4) Відображення поведінкових патернів, що відповідають елементам розподілу ролей (один агент штовхає, інший допомагає коригувати траєкторію блоку)

Новизна реалізованого підходу полягає у поєднанні методів MARL із централізованою системою управління епізодами, що дозволяє координувати агентів як єдину команду із колективною винагородою. Використанні динамічних змінних стану (позиції, обертання, випадкове розміщення) для створення навчального середовища зі змінною конфігурацією, що сприяє кращому узагальненню навичок.

Реалізації групової винагороди та синхронізованого завершення епізоду, що наближує модель до реалістичних сценаріїв командної роботи. Розширенні механізму зворотного зв’язку через обробку подій колізії, що дозволяє гнучко адаптувати поведінку на основі типу об’єкта та взаємодії з ним.

Інформаційній візуалізації результатів, яка надає користувачеві зворотний зв’язок щодо ефективності моделі у форматі, зручному для дослідника або користувача.

Розроблена система може бути адаптована до ширшого спектру задач — від робототехніки (кооперативна маніпуляція, автономна логістика) до симуляцій соціальної поведінки (наприклад, розподіл ресурсів, стратегічна координація).

Таким чином, результати дослідження демонструють не лише працездатність реалізованої системи, а й її потенціал як основи для подальшого розвитку більш складних сценаріїв колективного навчання, що може включати змагальні моделі, навчання з частковою інформацією або змішану мотивацію агентів.

2 ОСНОВНА ЧАСТИНА

2.1 Постановка задачі та вимоги до системи

Постановка задачі - навчання агентів у фізичному середовищі є одним з ключових напрямів у галузі штучного інтелекту, зокрема в контексті підкріплювального навчання (Reinforcement Learning, RL). У даному підході агент діє в середовищі, спостерігає за його станом, приймає дії та отримує винагороду залежно від ефективності виконаного завдання. Мета агента — максимізувати сумарну винагороду за певну кількість кроків або епізодів.

Особливістю даної роботи є використання фізично достовірного симульованого середовища з тривимірною графікою, реалізованого на основі ігрового рушія Unity. Таке середовище дає змогу моделювати просторові взаємодії між агентами, об’єктами, поверхнею та перешкодами, враховуючи фізичні сили, тертя, масу та інерцію. Завдяки цьому симуляція максимально наближена до реальних умов, що робить її цінною для тестування алгоритмів, які потенційно можуть бути перенесені у фізичні або роботизовані системи.

Навчання агентів у фізичному середовищі має декілька важливих аспектів: Спостереження — агенти отримують інформацію про навколишнє середовище у вигляді векторних або візуальних спостережень: положення цілі, координати блоку, власне положення, швидкість тощо.

Дії — у симуляції передбачено набір дискретних дій, таких як переміщення вперед/назад, обертання, рух у боки. Кожна дія має наслідки в межах фізичної сцени.

Винагорода — система винагород побудована таким чином, щоб стимулювати кооперацію агентів для досягнення мети (наприклад, спільного штовхання блоку до цілі). Також враховується час, помилки, неефективні дії.

Епізоди — симуляція ділиться на епізоди, кожен з яких починається зі випадкової ініціалізації сцени й завершується після досягнення цілі або вичерпання кількості кроків.

Завдяки модульній реалізації та інтеграції з Unity ML-Agents, система дозволяє тренувати кілька агентів одночасно у спільному середовищі. Агенти не мають прямої комунікації, тому координація виникає лише через взаємодію з об’єктом і навколишнім середовищем. Це створює складну задачу, яка включає не тільки планування дій, але й непряме навчання командної поведінки.
Вибір фізичного середовища є обґрунтованим з огляду на:

- потребу в реалістичній симуляції динаміки об’єктів;
- можливість візуалізації процесу навчання та спостереження за стратегіями агентів;
- універсальність та можливість масштабування до складніших сценаріїв (перешкоди, кілька об’єктів, ієрархія цілей).

Таким чином, навчання агентів у фізичному 3D-середовищі є не лише актуальною дослідницькою задачею, але й потужним інструментом для тестування методів машинного навчання в умовах, максимально наближених до реальності.

Одним із ключових напрямів сучасного розвитку штучного інтелекту є побудова моделей автономної поведінки агентів у динамічному, частково спостережуваному середовищі. Особливої складності набувають задачі, у яких необхідно не лише формувати ефективну індивідуальну поведінку, а й

координувати дії декількох агентів задля досягнення спільної мети. Такі сценарії зустрічаються у сфері логістики, автономного управління, робототехніки, моделювання соціальних систем та віртуального навчання.

Традиційні підходи до моделювання мультиагентної поведінки, засновані на жорстко прописаних правилах чи централізованому контролі, обмежені в контексті складних, непередбачуваних середовищ. У свою чергу, методи навчання з підкріпленням (Reinforcement Learning, RL), а також багатоагентного навчання з підкріпленням (Multi-Agent Reinforcement Learning, MARL), відкривають нові можливості для створення моделей, здатних до самостійного навчання й адаптації у змінних умовах.

Метою дослідження є побудова інтерактивної симуляційної системи, у якій кілька віртуальних агентів навчаються спільно виконувати завдання переміщення об’єктів у визначену цільову зону, діючи автономно, але узгоджено, без централізованого управління, використовуючи механізми колективної винагороди.

Принципова новизна задачі полягає в наступному - розробка гнучкої архітектури, що дозволяє формувати кооперативну поведінку агентів у середовищі з випадковими конфігураціями, що наближає симуляцію до умов реального застосування.

Моделювання декількох типів взаємодій між агентами й об'єктами, включно з реакцією на фізичні колізії, маніпуляції з різними типами блоків (різного розміру) та зміни просторових умов.

Реалізація системи групової винагороди та епізодичної синхронізації, яка дозволяє агентам ефективно навчатися через узагальнений досвід команди. Підготовка підґрунтя для подальших експериментів із розподілом ролей, самоорганізацією та навчанням із частковою спостережуваністю, що є актуальними напрямами досліджень у сучасній теорії MARL.

Отже, постановка задачі передбачає не лише технічну реалізацію середовища, а й дослідження фундаментальних аспектів колективної автономії, стратегічної взаємодії та узгодження дій у децентралізованих системах.
Завдання передбачає:

1) реалізацію симульованого середовища з тривимірною фізикою
2) створення агентів із можливістю прийняття дискретних дій (рух, обертання)
3) реалізацію об’єктів (блоків), з якими агенти можуть фізично взаємодіяти
4) впровадження механізму винагороди, що стимулює колективне досягнення мети
5) формування системи оцінки успішності (нагорода, прогрес, завершення епізоду)
6) реалізацію контролера середовища для ініціалізації, скидання епізодів і керування логікою симуляції

У процесі моделювання поведінки агентів у симульованому середовищі важливу роль відіграють обмеження, які накладаються як на саму структуру середовища, так і на можливості агентів. Ці обмеження є ключовими для формування навчального простору, визначення складності задачі та забезпечення коректності тренування.

Обмеження середовища - середовище, у якому функціонують агенти, є замкнутою фізичною сценою, яка має низку заздалегідь визначених властивостей.

Розмір арени: простір має обмежені габарити, що не дозволяє агентам віддалятися нескінченно далеко або уникати взаємодії з ціллю. Це сприяє формуванню фокусованої поведінки.

Наявність об’єкта (блоку): блок, який необхідно перемістити, має фізичну масу, розміри та інерцію. Агенти не можуть просто "взяти" об’єкт — вони повинні штовхати його з урахуванням сили тертя та розміру.

Гравітація та фізичні властивості: сцена підпорядковується реалістичним законам фізики, зокрема діє гравітація, блок і агенти мають масу, взаємодії підпорядковуються законам динаміки, є тертя поверхні.

Обмеження по часу: кожен епізод має обмежену кількість кроків. Якщо за цей час блок не буде переміщено в ціль, епізод вважається невдалим, і агенти отримують штраф.

Фіксована позиція цілі: ціль, до якої потрібно доставити об’єкт, має фіксоване положення (або обирається випадково в межах сцени на початку епізоду). Її зміщення або блокування унеможливлює досягнення мети.

Обмеження дій агентів - дії, які доступні агентам, також мають обмежений характер — це забезпечує формування дискретного простору дій і знижує складність навчання:

Дискретний набір дій - кожен агент може виконувати одну з наступних дій: рух вперед, рух назад, обертання вправо, обертання вліво, рух праворуч, рух ліворуч).

Обмеження швидкості: рухи агентів мають фіксовану швидкість і обмеження на зміну напряму, що унеможливлює миттєві розвороти або різкі маневри.

Відсутність комунікації: агенти не можуть обмінюватися інформацією напряму (наприклад, через повідомлення). Усі стратегії мають виникати в результаті взаємодії через середовище.

Обмежене спостереження: агенти не мають повного уявлення про сцену. Зазвичай вони "бачать" лише координати об’єкта, своє положення, напрям руху та ціль — але не дії інших агентів.

Фізичні обмеження: агент не може "проходити крізь" об’єкти чи інші агентів. Усі зіткнення враховуються, що додає додаткову складність у координації дій.

Такі обмеження створюють реалістичні умови навчання, у яких агенти змушені виробляти ефективні стратегії, не маючи прямого контролю над середовищем або комунікації. Це дозволяє вивчати справжню колективну поведінку, яка виникає як побічний продукт взаємодій, а не через явно задану логіку.

Функціональні вимоги до системи. Система повинна дозволяти запуск і скидання навчальних епізодів з випадковою ініціалізацією (позиції агентів, блоків, цілі). Агенти повинні мати змогу: пересуватись у просторі, приймати дискретні дії, взаємодіяти з об’єктами середовища, отримувати інформацію про стан середовища (спостереження).

Об’єкти повинні володіти фізичними характеристиками (маса, інерція, ковзання, взаємодія з агентами). Ціль повинна бути чітко визначеною зоною, зіткнення з якою фіксується як успішне досягнення мети. Необхідно забезпечити спільну (групову) винагороду за досягнення колективного результату. Повинна бути реалізована система візуалізації результатів (винагорода, епізоди, досягнення цілі).

Нефункціональні вимоги. Гнучкість - система має дозволяти масштабування, додавання нових агентів або зміну сценаріїв без необхідності значної модифікації коду.

Стабільність: поведінка агентів не повинна спричиняти збої в роботі симуляції (зависання, нескінченні цикли).

Продуктивність: симуляція повинна працювати у реальному часі з можливістю прискореного навчання.

Модульність: архітектура повинна підтримувати поділ на окремі логічні компоненти (агент, блок, контролер середовища, система винагород).

Контекст середовища, симуляція реалізується в середовищі Unity, де фізика і візуалізація забезпечуються рушієм Unity3D. Взаємодія з алгоритмами навчання реалізується через фреймворк Unity ML-Agents. Комунікація між агентами та середовищем базується на обміні векторами спостереження та діями, що обробляються в реальному часі в рамках циклу навчання з підкріпленням.

2.2 Огляд підходів до моделювання поведінки агентів

У галузі навчання з підкріпленням для багатоагентних систем (Multi-Agent Reinforcement Learning, MARL) існують два основні підходи до побудови поведінки агентів — індивідуальне (selfish, independent learning) та кооперативне (collaborative, centralized learning) навчання. Вибір підходу залежить від поставленої задачі, типу взаємодії між агентами, доступних ресурсів та рівня знань про середовище.

Індивідуальне навчання передбачає, що кожен агент навчається самостійно, не враховуючи присутність або дії інших агентів. Такий підхід розглядає інших агентів як частину середовища, що ускладнює завдання, оскільки середовище стає недетермінованим і нестабільним з точки зору одного агента.

Основні характеристики:

1) кожен агент має власну політику, модель, систему винагород і спостереження
2) не використовується спільна інформація про дії або стани інших агентів
3) підходить для незалежних або конкурентних сценаріїв

Може бути реалізований із використанням стандартних одноагентних алгоритмів (Q-learning, DQN, PPO, A3C), модифікованих для мультиагентного випадку. Переваги: простота реалізації, масштабованість (нові агенти можна додавати без повного перенавчання), не вимагає централізованого контролю чи синхронізації. Недоліки:можливе нестабільне навчання, оскільки інші агенти змінюють політику під час тренування, погана координація, особливо в задачах з необхідністю колективної дії.

Кооперативне (спільне) навчання спрямоване на узгоджену поведінку декількох агентів, які працюють разом для досягнення спільної мети. У таких

системах агенти можуть мати спільну винагороду, використовувати централізований тренувальний процес або обмінюватися інформацією про стани, дії або наміри.

Основні стратегії кооперативного навчання:

1) Centralized Training with Decentralized Execution (CTDE) — агенти навчаються за участю централізованого модуля (який має доступ до інформації про всі стани), але виконують дії незалежно;

2) Multi-Agent PPO / MADDPG — адаптації класичних алгоритмів під MARL з централізованими критиками;

3) Спільна винагорода — усі агенти отримують однаковий сигнал про успішність епізоду, що стимулює координацію;

4) Імпліцитна кооперація — агенти не обмінюються повідомленнями, але навчаються через спільну ціль (як у цій роботі).

Переваги: покращена координація та синхронізація дій, можливість складних стратегій взаємодії, ефективність у колективних задачах, таких як спільне штовхання об’єкта, навігація у групах тощо.

Недоліки: складність реалізації, потреба в більших обчислювальних ресурсах, ризик перенавчання на фіксованій кількості агентів або сценаріїв, часто потребує централізованого модуля або сервера під час навчання.

У проєкті використано підхід кооперативного навчання із спільною винагородою, що дозволяє формувати узгоджену поведінку без явної комунікації між агентами. Агенти не мають доступу до дій інших агентів, однак завдяки спільній меті та винагороді навчаються діяти синхронно та стратегічно, наприклад, оминати один одного, підлаштовуватися під рух об’єкта або не заважати іншим.

Такий підхід ускладнює навчання, проте дозволяє спостерігати емерджентну кооперацію — тобто появу складної взаємодії як наслідку простих локальних правил. Одноагентні та багатоагентні архітектури. При побудові систем з навчанням з підкріпленням важливо розрізняти архітектури, що базуються на одному агенті (Single-Agent RL) і на кількох взаємодіючих агентах (Multi-Agent RL). Вибір між одноагентною та багатоагентною архітектурою має критичне значення для постановки задачі, розробки середовища, формулювання винагороди та алгоритму навчання.

Одноагентна архітектура (Single-Agent RL). Водноагентній архітектурі відповідає ситуація, коли в симульованому або реальному середовищі діє лише один агент, який виконує послідовність дій для досягнення заданої мети. Такі архітектури найбільш широко досліджені в літературі з RL та є базовим варіантом для моделювання поведінки. Особливості: агент має повну або часткову інформацію про середовище, середовище вважається статичним, якщо не змінюється самостійно (без участі агента), простіше визначити політику навчання та обчислювати винагороду, можливе використання класичних алгоритмів RL: Q-learning, SARSA, DQN, PPO, A2C тощо. Переваги: нижча складність реалізації та обчислень, більша стабільність процесу навчання, відсутність взаємозалежностей між агентами. Недоліки: не підходить для задач, які передбачають взаємодію між кількома агентами, не дозволяє моделювати

ситуації кооперації, конкуренції чи розподіленої відповідальності, слабка масштабованість до реальних сценаріїв із багатьма діючими об'єктами.

Багатоагентна архітектура (Multi-Agent RL). У багатоагентній архітектурі кілька агентів взаємодіють із загальним середовищем, або між собою, або одночасно й з тим, і з іншим. Ця модель наближає симуляцію до реальних систем — таких як автономні транспортні засоби, роботи в логістиці, безпілотники або персонажі в ігрових сценах.

Види багатоагентної взаємодії:

1) кооперація — агенти працюють разом задля спільної мети
2) конкуренція — агенти змагаються за ресурси або перевагу
3) коопетиція — поєднання кооперації й конкуренції

Особливості:

кожен агент має свою політику та спостереження;

середовище є нестабільним з точки зору одного агента, оскільки змінюється через дії інших

навчання може бути централізованим або децентралізованим

Часто використовується спільна винагорода або централізовані критики (наприклад, у MADDPG, MA-PPO). Переваги: можливість моделювати складну взаємодію, що виникає з багатьох локальних стратегій, підходить для симуляцій колективної поведінки, наближеність до реальних багатокомпонентних систем. Недоліки: висока обчислювальна складність, нестабільність навчання через постійні зміни політик інших агентів, складніша реалізація і потреба в спеціалізованих методах координації або комунікації.

У роботі реалізовано багатоагентну архітектуру, в якій кілька агентів одночасно навчаються і діють у симульованому середовищі. Їхня мета — спільне штовхання блоку в напрямку цілі, причому вони не мають прямої комунікації, однак отримують спільну винагороду, що стимулює координацію.

Такий підхід дозволяє:

1) аналізувати емерджентну поведінку без централізованого управління
2) досліджувати вплив кількості агентів на швидкість і якість досягнення мети
3) оцінити здатність моделі адаптуватися до сценаріїв із частковою інформацією та динамікою

Обрана архітектура є складнішою за одноагентну, але вона відкриває нові можливості для вивчення колективного навчання, що є особливо актуальним у контексті мультиагентних систем штучного інтелекту.

Обмеження середовища та дій. При побудові симульованого середовища для навчання віртуальних агентів критично важливо визначити та врахувати обмеження, які впливають на навчальний процес, можливості агентів та характер взаємодії з оточенням. Такі обмеження формують структуру завдання, обмежують простір дій і забезпечують досяжність цілей у контрольованих умовах. У даній роботі середовище реалізовано в рушії Unity з використанням бібліотеки ML-Agents, що дозволяє моделювати фізично обґрунтовану поведінку.

1. Просторові обмеження

Середовище має фіксовані розміри та обмежене просторове поле, на якому відбувається взаємодія. Агентам заборонено виходити за межі сцени або рухатись у зони, які не містять корисної інформації чи об'єктів. Це забезпечує

зосередженість навчання на ключових діях та зменшує розмірність простору спостережень.

2. Фізичні обмеження

Гравітація: активована в сцені Unity, що впливає на переміщення об'єктів. Агенти та блок підлягають впливу сили тяжіння, а отже, не можуть «літати» або переміщатися без тертя.

Сили зіткнення: взаємодії агентів із блоком або між собою підкоряються законам фізики. Неможливо «пройти крізь» інші об'єкти, що вимагає планування дій з урахуванням перешкод.

Маса об'єктів: маса блоку більша за масу агента, тому для його переміщення часто потрібні зусилля декількох агентів.

3. Обмеження часу

Кожен епізод має обмеження по кількості кроків. Якщо блок не буде доставлено в ціль за цей час, епізод завершується автоматично з негативною винагородою. Це стимулює агентів до ефективної взаємодії та уникнення бездіяльності.

4. Обмеження спостереження

Агенти не мають повного доступу до стану середовища. Замість глобального бачення вони отримують лише локальні спостереження, які можуть включати: власну позицію та орієнтацію, відносне положення до блоку та цілі, швидкість та напрямок руху, відсутність знання про дії інших агентів.

Це створює умови часткової обізнаності (partial observability), наближені до реальних сценаріїв.

5. Обмеження дій агентів

У даній реалізації агенти мають дискретний простір дій, що включає: рух уперед, рух назад, поворот праворуч, поворот ліворуч, рух вбік (вправо або вліво — залежно від конфігурації). Таке обмеження дозволяє: спростити простір пошуку оптимальної політики, зменшити ризик перенавчання на неконтрольовані дії, пришвидшити процес навчання без втрати складності задачі.

6. Відсутність комунікації

Агенти не можуть напряму обмінюватися інформацією або сигналами. Це означає, що вся координація виникає через опосередковану взаємодію з об'єктом (блоком), ціллю та іншими агентами. Такий підхід дає змогу дослідити явище емерджентної кооперації — коли складна поведінка формується із простих реакцій на стан середовища.

Таким чином, обмеження середовища й дій створюють умови для гнучкого, але контрольованого навчання. Вони забезпечують складність задачі, необхідну для розвитку адаптивної поведінки, і водночас дозволяють реалізувати навчання в обчислювально ефективний спосіб. Завдяки цим обмеженням агенти вчаться діяти злагоджено, стратегічно та в умовах обмеженої інформації — що повністю відповідає меті дипломного проєкту.
Вимоги до функціональності системи

Для успішної реалізації симульованого середовища з віртуальними агентами, які навчаються виконувати колективну дію — зокрема, переміщення об’єкта до заданої цілі, — необхідно визначити функціональні вимоги до кожного ключового компонента системи. Усі елементи повинні взаємодіяти узгоджено та відповідати умовам фізично достовірного середовища.

Віртуальні агенти є головними діючими особами в середовищі. Вони повинні мати наступну функціональність:

1) Здатність виконувати дискретні дії:переміщення вперед, назад обертання, рух у сторони.
2) Отримання векторних спостережень про поточний стан середовища: координати цілі, блока, власне положення, швидкість тощо.
3) Реакція на зіткнення з об'єктами (за допомогою подій Unity OnCollisionEnter);
4) Прийняття рішень на основі політики, яка формується через навчання з підкріпленням
5) Накопичення винагороди за успішну поведінку (наприклад, якщо блок досяг цілі)
6) Ініціалізація стану (позиції, орієнтації) на початку кожного епізоду.

У середовищі присутній щонайменше один рухомий об’єкт (блок), який агенти повинні колективно перемістити до статичної або динамічної цілі. Об'єкти мають відповідати таким вимогам:

1) Реалізація фізичних властивостей (маса, інерція, розміри, сила тертя)
2) Детектування досягнення цілі — наприклад, за допомогою Trigger-зони з міткою goal
3) Можливість рандомізації положення на початку кожного епізоду

Забезпечення умов взаємодії з кількома агентами — зокрема, блок повинен бути досить важким або великим, щоб один агент не зміг ефективно його перемістити самостійно.

Контролер відповідає за керування логікою симуляції, підтримку епізодів і координацію початкових станів. Його функціональність включає: ініціалізація сцени: позиціонування агентів, об’єктів і цілі перед початком кожного епізоду,

випадкова генерація параметрів середовища (обертання платформи, положення об'єктів), контроль кількості кроків епізоду та його завершення (за досягненням цілі або тайм-аутом), обчислення винагороди і передача її агентам або групі агентів, збір статистики (наприклад, кількість епізодів, сукупна винагорода, тривалість навчання).

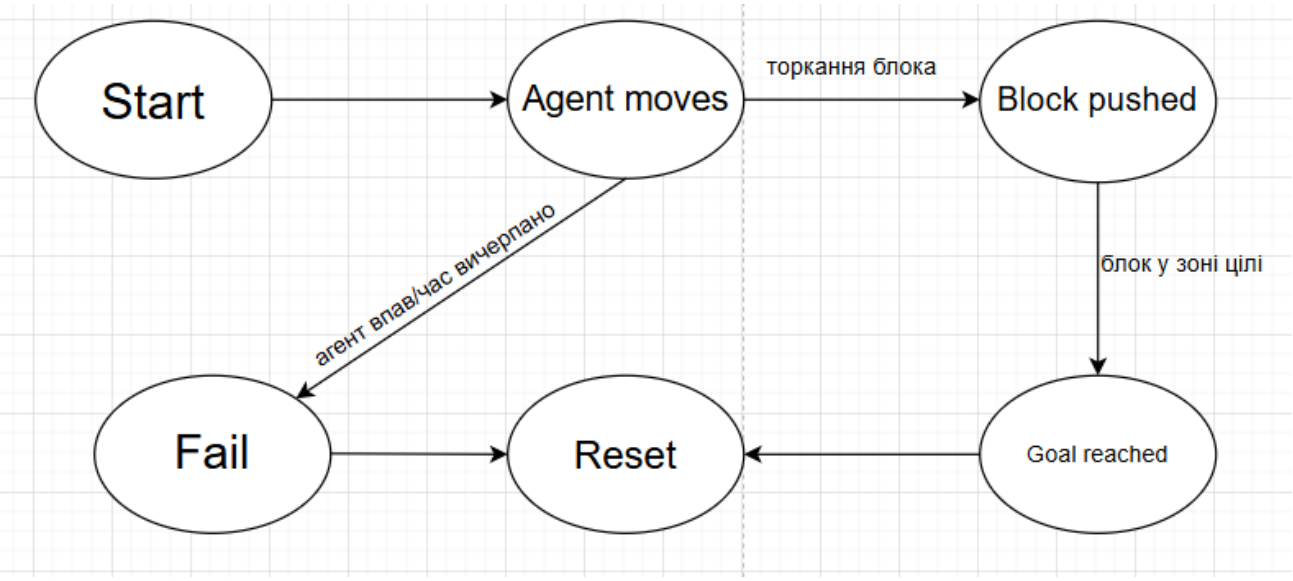

Рис. 1 Діаграми станів

Система винагороди формує основу навчального сигналу, за допомогою якого агенти поступово оптимізують свою поведінку. У цій роботі передбачено такі функціональні вимоги до неї:

1) Групова винагорода за досягнення мети — наприклад, +5 за успішне переміщення блоку в ціль
2) Штрафи за неефективну поведінку, зокрема за кожен крок часу (щоб стимулювати швидше досягнення мети)
3) Можливість нагородження або покарання часткових дій (наприклад, наближення до цілі, віддалення від неї)

4) Адаптивність — система повинна коректно працювати як при зміні кількості агентів, так і при варіаціях початкових умов.

Усі зазначені функціональні вимоги спрямовані на забезпечення надійної, адаптивної та контрольованої симуляції, у якій можливо ефективно тренувати агентів до колективної поведінки. Їх реалізація дозволяє не тільки створити працездатну систему, а й провести якісний аналіз навчання, відстеження динаміки винагород, еволюцію стратегій агентів та взаємодію між ними.

2.3 Архітектура моделі та алгоритм навчання

2.3.1 Unity як середовище для побудови симуляцій

Unity — це один із найпопулярніших у світі ігрових рушіїв, який використовується не лише в розробці відеоігор, а й у наукових дослідженнях, візуалізації, моделюванні фізичних процесів, створенні тренажерів, систем віртуальної та доповненої реальності. Його універсальність та доступність зробили Unity привабливою платформою для побудови симуляцій і в галузі штучного інтелекту, зокрема для навчання з підкріпленням.

Unity забезпечує низку важливих можливостей для реалізації інтерактивних і фізично обґрунтованих середовищ: Фізичний рушій (на основі NVIDIA PhysX), який дозволяє точно моделювати сили, зіткнення, тертя та масу об’єктів. Гнучка система скриптів, написаних на C#, що дозволяє створювати унікальну логіку поведінки об’єктів. Підтримка 2D та 3D-графіки, візуальних ефектів, анімацій і камер — що дозволяє налаштовувати симуляції як для тренування, так і для демонстрації результатів. Редактор середовища в реальному часі, який спрощує розміщення об’єктів, налаштування сцен та експериментування з параметрами.

Кросплатформеність, що дозволяє запускати симуляції на різних пристроях, включаючи ПК, мобільні пристрої та VR-гарнітури.

Саме ці характеристики зробили Unity популярною основою для симуляційних середовищ у сфері машинного навчання. Але щоб інтегрувати штучний інтелект до таких симуляцій, потрібен додатковий інструмент — і ним став Unity ML-Agents Toolkit.

ML-Agents (Machine Learning Agents) — це офіційне розширення Unity, яке дозволяє поєднувати фізичне 3D-середовище з алгоритмами навчання з підкріпленням. За допомогою ML-Agents можна:

1) Створювати агентів, які отримують спостереження, приймають дії й отримують винагороду
2) Налаштовувати системи винагород, правила завершення епізодів, цілі поведінки
3) Тренувати агентів за допомогою сучасних алгоритмів RL (наприклад, PPO, SAC) через інтеграцію з Python
4) Аналізувати поведінку агентів у реальному часі

Отже, Unity у поєднанні з ML-Agents виступає не просто рушієм для візуалізації — це потужна платформа для моделювання, дослідження та навчання агентів у складних, динамічних і багатовимірних середовищах, які близькі до реальних сценаріїв.

2.3.2 Взаємодія агента з середовищем (основна модель RL)

Навчання з підкріпленням (Reinforcement Learning, RL) є одним із ключових підходів до моделювання поведінки інтелектуальних агентів у симульованих середовищах. У цій парадигмі агент взаємодіє з середовищем

шляхом виконання дій, отримуючи винагороду, яка відображає корисність обраної поведінки. Мета агента — навчитися стратегії, яка максимізує сумарну винагороду протягом епізодів.

Основна структура взаємодії в моделі RL - взаємодію агента з середовищем можна описати як циклічний процес, що складається з таких етапів:

Observation (спостереження) - на кожному кроці часу агент отримує певний набір спостережень — це інформація про стан середовища, яка доступна з його точки зору. У середовищах ML-Agents це може бути векторна інформація (позиції, відстані, кути, швидкість), зображення з камери або їх комбінація.

Action (дія) - на основі спостереження агент приймає дію. Дії можуть бути дискретними (наприклад, рух уперед, назад, обертання вліво/вправо) або неперервними (наприклад, сила натискання, напрямок руху). У цій роботі використано дискретний простір дій.

Environment update (оновлення середовища) - середовище змінюється відповідно до дії агента та вбудованої фізики. Це може включати переміщення об'єктів, взаємодію з іншими агентами, зміну стану тощо.

Reward (винагорода) - після виконання дії агент отримує числову винагороду — позитивну, негативну або нульову. Наприклад, наближення до мети може приносити невелику винагороду, а досягнення мети — велику. Штрафи можуть застосовуватись за бездіяльність або зіткнення з перешкодами.

Episode termination (завершення епізоду) - епізод завершується, якщо виконано цільове завдання (наприклад, блок доставлено в зону), минув

максимальний час або сталося порушення умови (наприклад, вихід за межі). Після цього агент проходить новий цикл з початковим станом.

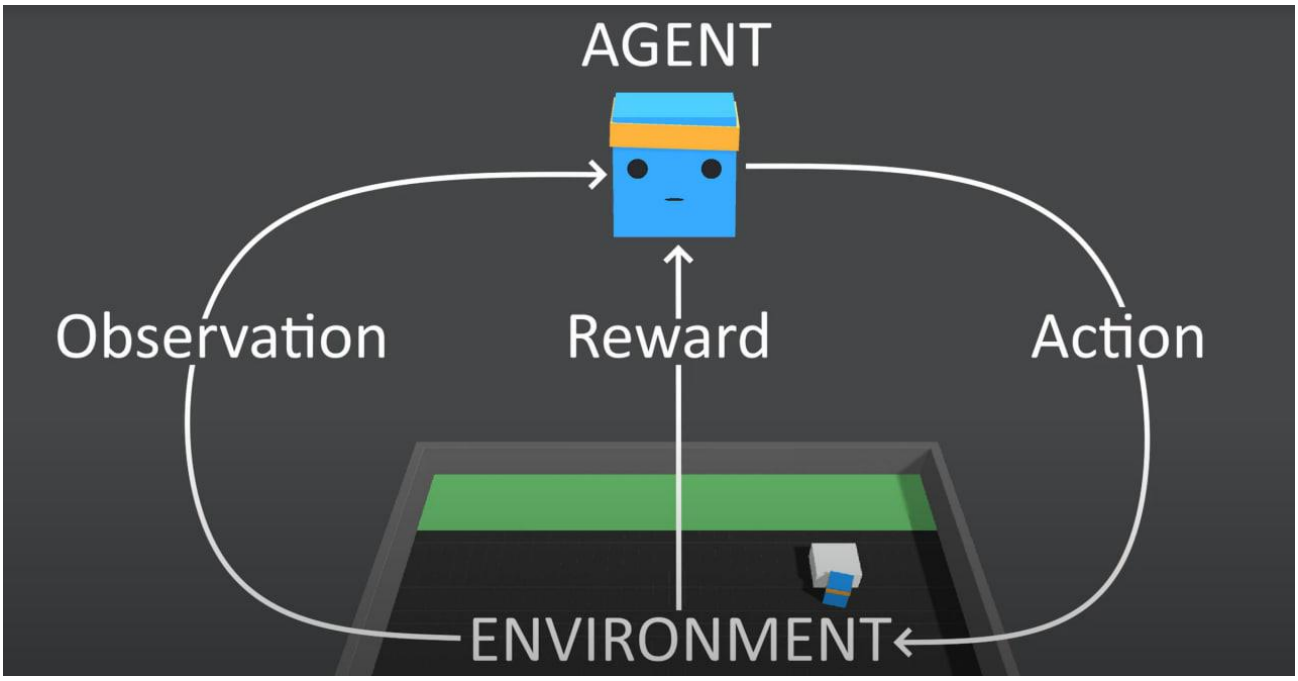

Рис. 2 Основна схема взаємодії агента з середовищем у навчанні з підкріпленням

Формалізація - середовище як марковський процес прийняття рішень (MDP). Взаємодію можна описати у вигляді марковського процесу прийняття рішень (Markov Decision Process, MDP). Модель MDP працює, використовуючи ключові елементи, такі як агент, стан, дії, винагороди та оптимальні політики. Агент відноситься до системи, відповідальної за прийняття рішень та виконання дій. Він працює в середовищі, яке деталізує різні стани, в яких знаходиться агент при переході з одного стану до іншого. MDP визначає механізм того, як певні стани та дії агента призводять до інших станів. Більш того, агент отримує винагороди в залежності від виконуваної ним дії та стану, якого він досягає (поточний стан). Політика моделі MDP розкриває таку дію агента залежно від його поточного стану.

Структура MDP включає наступні ключові компоненти:

S : стани (s ∈ S)

A : Дії (a ∈ A)

P (S t+1 | s t .a t) : Можливості переходу

P(c): Нагорода

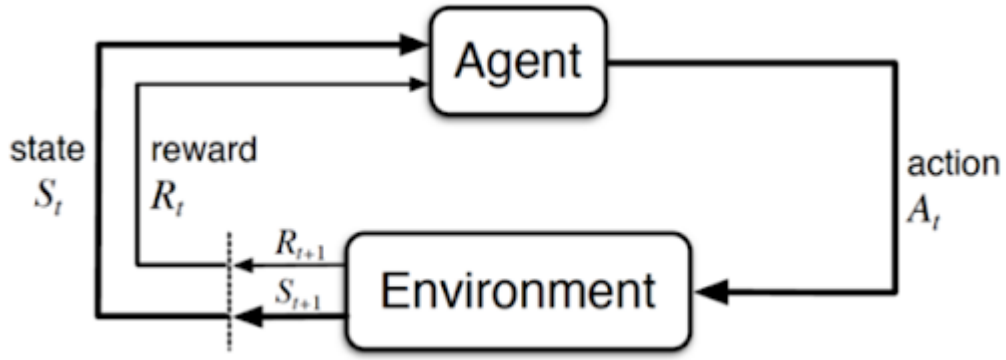

Рис. 3 Графічне представлення моделі MDP

Модель MDP використовує властивість Маркова, яка свідчить, що майбутнє може бути визначене лише з цього стану, яке інкапсулює всю необхідну інформацію з минулого. Властивість Маркова можна оцінити за допомогою цього рівняння:

P[S t+1 |S t] = P[S t+1 |S 1 ,S 2 ,S 3 ……S t]

Відповідно до цього рівняння, ймовірність наступного стану (P[S t+1]) при поточному стані (S t) визначається ймовірністю наступного стану (P[S t+1]) з урахуванням усіх попередніх станів (S 1 ,S 2 ,S 3 .....S t). Це означає, що MDP використовує лише даний/поточний стан для оцінки наступних дій без будь-яких залежностей від попередніх станів або дій. Відомо, що політика ( ) визначає оптимальну дію агента з урахуванням поточного стану, щоб він отримав максимальну нагороду. Простіше кажучи, вона пов'язує події зі станами.

П: С -> А

Щоб визначити найкращу політику, важливо визначити прибутковість, яка показує винагороду агента у кожному стані. В результаті метод горизонту не є кращим для фокусування на короткострокових або довгострокових винагородах. Натомість вводиться змінна, звана «дисконтований фактор (γ)». Правило говорить, що якщо γ має значення, які ближче до нуля, то пріоритет надається негайним винагородам. Згодом, якщо γ показує значення, близькі до одиниці, фокус зміщується на довгострокові винагороди. Отже, метод дисконтованого нескінченного горизонту є ключем до виявлення найкращої політики.

Функція цінності V(s) визначає прибутковість винагороди у кожному конкретному стані. Її формула характеризується очікуваною сумою дисконтованих майбутніх винагород.

$$V(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t | s_t]$$

Функцію цінності можна розділити на два компоненти: винагороду поточного стану та дисконтовану винагороду наступного стану. Ця розбивка виводить рівняння Беллмана, як показано нижче:

$$V^*(s) = max_a [R(s,a) + \gamma \sum_{s' \epsilon S} P(s'|s,a) V^*(s')]$$

Тут слід зазначити, що дії та винагороди агента різняться в залежності від політики. Функція оптимального значення може бути вирішена за допомогою ітераційних методів, таких як оцінки Монте-Карло, динамічне програмування або навчання за тимчасовою різницею. Політика, яка вибирає максимальне оптимальне значення з урахуванням поточного стану, називається оптимальною

політикою. Математично вона представлена наступним рівнянням:

$$\Pi^*(s) = argmax_a[R(s,a) + \gamma \sum_{s' \epsilon S} P^a_{ss'} V(s')]$$

Отже, політика є наслідком поточного стану, де на кожному тимчасовому етапі оцінюється нова політика на основі інформації про поточний стан. Це забезпечується кількома методами, такими як ітерація політики, Q-навчання, ітерація значень та лінійне програмування для вирішення оптимальної функції політики.

2.3.3 Структура архітектури ML-Agents у Unity

У ML-Agents цикл взаємодії реалізується через інтеграцію з Python-тренером:

- Середовище працює у Unity (включає агентів, об'єкти, фізику)
- Агент передає вектор спостережень через API
- Python Trainer отримує спостереження, розраховує дію (на основі політики) та повертає її назад

Система винагороди реалізована в Unity, і відповідні значення також передаються тренеру. Вся комунікація відбувається через спеціалізований.

Communicator, який пов’язує Unity-сцену з Python API.

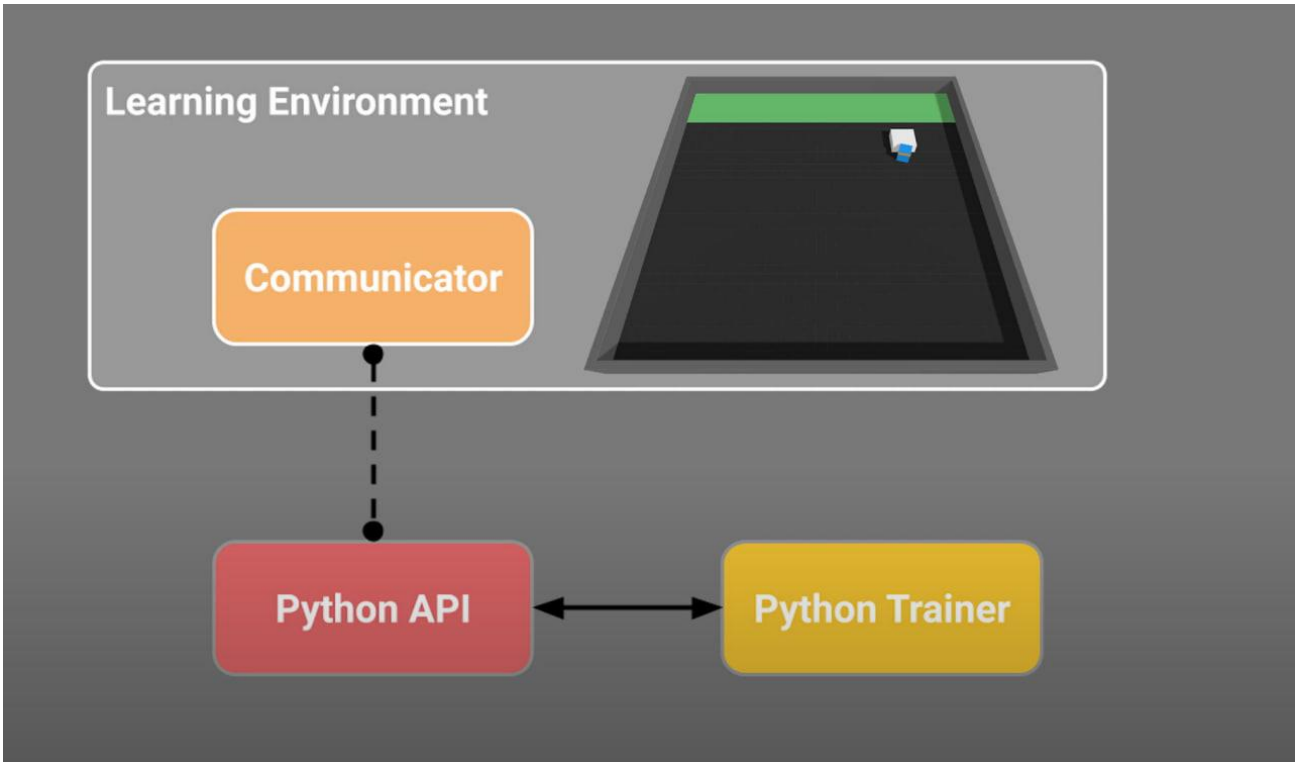

Рис. 4 Архітектура навчання ML-Agents: взаємодія середовища Unity з Python Trainer через API

Під час навчання цей зовнішній процес взаємодіє з об'єктом Academy у сцені Unity для створення блоку досвіду агента. Цей досвід стає навчальним набором для нейронної мережі, що використовується для оптимізації політики агента (яка є математичною функцією, що відображає спостереження в дії). Під час навчання з підкріпленням нейронна мережа оптимізує політику, максимізуючи очікувані винагороди. При навчанні імітацією нейронна мережа оптимізує політику для досягнення найменшої різниці між діями, обраними агентом, що навчається, і діями, обраними експертом у тій же ситуації.

Виходом процесу навчання є файл моделі, що містить оптимізовану політику. Цей файл моделі є графом даних TensorFlow, що містить математичні операції та оптимізовані ваги, вибрані в процесі навчання. Ви можете

використовувати згенерований файл моделі з типом Learning Brain у своєму проекті Unity, щоб вибрати найкращий курс дій для агента.

2.3.4 PPO як основний алгоритм навчання (принцип роботи, переваги, обмеження)

У рамках реалізованої системи навчання агентів було обрано алгоритм Proximal Policy Optimization (PPO) — один із найпопулярніших методів навчання з підкріпленням, що належить до класу політико-градієнтних методів (Policy Gradient Methods). Цей алгоритм був розроблений дослідниками компанії OpenAI у 2017 році як відповідь на складнощі з реалізацією та стабільністю попередніх методів, зокрема Trust Region Policy Optimization (TRPO).

Принцип роботи PPO - алгоритм PPO полягає в тому, щоб покроково оновлювати політику агента, яка задає ймовірність вибору дії в певному стані, таким чином, щоб не відходити надто далеко від попередньої (старої) політики. Це обмеження дозволяє уникнути надмірних змін, які можуть дестабілізувати процес навчання. Ключовим у PPO є кліпована функція втрат (clipped surrogate objective), яка виглядає так:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ — відношення нової політики до старої;

$\hat{A}_t$ — перевага (advantage), яка оцінює, наскільки обрана дія була кращою за середню;

ϵ — параметр, що визначає допустимий розрив між старою та новою політикою.

Функція втрат гарантує, що оновлення параметрів політики буде обережним — великі зміни караються обрізанням виграшу, а отже, уникненням ризикованих стратегій.

Переваги PPO

- Простота реалізації

PPO значно простіший у реалізації, ніж TRPO, не вимагає обчислення другої похідної або інверсії матриць, що робить його ідеальним для практичного використання.

- Стабільність та ефективність

Завдяки механізму обмеження змін політики (через кліпування) алгоритм демонструє високу стабільність навіть у складних або стохастичних середовищах.

- Гнучкість

Алгоритм добре масштабується на багатоагентні сценарії, неперервні та дискретні простори дій, а також підтримується в більшості RL-фреймворків, включаючи Unity ML-Agents.

- Баланс між дослідженням і експлуатацією.

PPO заохочує агента шукати кращу поведінку, водночас зберігаючи успішні стратегії з минулого.

Обмеження PPO

- Чутливість до гіперпараметрів.

Ефективність алгоритму сильно залежить від правильно підібраних параметрів — коефіцієнта кліпування, швидкості навчання, кількості кроків між оновленнями політики тощо.

- Високі обчислювальні витрати.

PPO потребує значної кількості симуляцій (епізодів), що може бути затратним за часом при складній фізичній симуляції.

- Нестабільність у дуже великих просторах станів.

У деяких випадках PPO може бути нестабільним або повільно збігатися, якщо станова інформація є надмірною або шумною.

У розробленій системі PPO було реалізовано як основний алгоритм через його сумісність із Unity ML-Agents, підтримку дискретних дій, високу стабільність, а також достатньо швидку збіжність навіть у мультиагентному середовищі.

2.3.5 Система винагород

У системах навчання з підкріпленням (RL) сигнал винагороди є головним інструментом формування поведінки агента. Саме на основі позитивного або негативного підкріплення агент поступово навчається досягати поставленої мети. У розробленій симуляції з колективним навчанням агентів реалізовано як індивідуальну, так і групову систему винагород, що дозволяє досліджувати різні моделі кооперації.

Індивідуальна винагорода - у найпростішому варіанті один агент (PushAgentBasic) навчається самостійно переміщувати блок у ціль. Його винагорода нараховується за досягнення мети (ScoredAGoal):

```
/// Called when the agent moves the block into the goal.
/// </summary>
Ссылок: 1
public void ScoredAGoal()
{
    AddReward(5f);
    EndEpisode();

    // Swap ground material for a bit to indicate scored.
    StartCoroutine(GoalScoredSwapGroundMaterial(m_PushBlockSettings.goalScoredMaterial, 0.5f));
}
```

Рис. 5 Нарахування індивідуальної винагороди

Це основний стимул до навчання — коли блок досягає зони з міткою goal, агент отримує велику позитивну винагороду та завершує епізод. Велика позитивна винагорода за досягнення мети (+5). Це — основна мета задачі: доставити блок у ціль. Велика винагорода мотивує агента або групу шукати стратегії, які призводять саме до цього результату.

Таке підкріплення називають sparse reward — воно рідке, але значуще. Без нього агент не знатиме, що саме є «успіхом» у задачі. Бо RL-агент не має попередніх знань, і його потрібно однозначно навчити: «Оце — добре, прагни цього». MaxStep на кожному кроці.

```
/// <summary>
/// Called every step of the engine. Here the agent takes an action.
/// </summary>
Ссылок: 0
public override void OnActionReceived(ActionBuffers actionBuffers)

{
    // Move the agent using the action.
    MoveAgent(actionBuffers.DiscreteActions);

    // Penalty given each step to encourage agent to finish task quickly.
    AddReward(-1f / MaxStep);
}
```

Рис. 6 Підкріплення sparse reward

Штраф за бездіяльність або затягування рішення стимулює агента досягати цілі якомога швидше. Малий негативний штраф за кожен крок (-1 / MaxStep). Це класична практика в RL, яка стимулює агента завершити задачу якомога швидше.

Інакше без штрафу він міг би ходити по коли, поки випадково не досягне мети. Це формує жорсткий тайм-менеджмент для стратегії: діяти потрібно ефективно, бо кожен крок — втрата.

Групова винагорода для багатокористувацького сценарію (PushAgentCollab) реалізовано кооперативну винагороду через SimpleMultiAgentGroup:

Усім агентам нараховується однакова винагорода, якщо ціль досягнута. У колективному сценарії (Multi-Agent RL) усі агенти отримують бонус, навіть якщо дію виконав лише один. Це створює кооперативне середовище, де агенти вчаться:не заважати одне одному; координуватися; ділити відповідальність. Це імітативна модель колективної дії, де успіх одного — успіх усіх. Без цього інші агенти просто стояли б, очікуючи, що хтось інший все зробить (ефект «фрірайдера»).

```
/// <summary>
/// Called when the agent moves the block into the goal.
/// </summary>
Ссылок: 0
public void ScoredAGoal(Collider col, float score)
{
    //Decrement the counter
    m_NumberOfRemainingBlocks--;
    bool done = m_NumberOfRemainingBlocks == 0;

    //Disable the block
    col.gameObject.SetActive(false);

    //Give Agent Rewards
    m_AgentGroup.AddGroupReward(score);

    // Swap ground material for a bit to indicate scored.
    StartCoroutine(GoalScoredSwapGroundMaterial(m_PushBlockSettings.goalScoredMaterial, 0.5f));

    if (done)
    {
        //Reset assets
        m_AgentGroup.EndGroupEpisode();
        ResetScene();
    }
}
```

Рис. 7 Приклад винагороди за досягнення цілі.

Це стимулює всіх агентів до колективної дії — навіть якщо конкретний агент не штовхав блок безпосередньо, він отримає бонус за командний результат. Таке навчання імітує умови, коли результат команди важливіший за індивідуальні дії.

```
Сообщение Unity | Ссылок: 0
void FixedUpdate()
{
    m_ResetTimer += 1;
    if (m_ResetTimer >= MaxEnvironmentSteps && MaxEnvironmentSteps > 0)
    {
        m_AgentGroup.GroupEpisodeInterrupted();
        ResetScene();
    }

    m_AgentGroup.AddGroupReward(-0.5f / MaxEnvironmentSteps);

    foreach (var agent in AgentsList)
    {
        totalReward += agent.Agent.GetGroupReward();
    }

    Debug.Log($"Episode: {episode}, Total Group Reward: {totalReward}");
}
```

Рис. 8 Груповий штраф за затягування у кожному кроці

Цей штраф також розподіляється на всю команду і знижує загальну винагороду за неефективне використання часу.

```
/// <summary>
/// Called every step of the engine. Here the agent takes an action.
/// </summary>
Ссылок: 1
public override void OnActionReceived(ActionBuffers actionBuffers)

{
    // Move the agent using the action.
    MoveAgent(actionBuffers.DiscreteActions);
    Vector3 current_position = transform.position;
    float new_distance = Vector3.Distance(current_position,GoalPosition);
    if(distance > new_distance)
    {
        AddReward(-0.35f);
    }
    else
    {
        AddReward(0.35f);
    }
}
```

Рис. 9 Стимул за зближення до цілі

Така евристика формує градацію проміжного прогресу — якщо агент рухається у напрямку до цілі, він отримує позитивне підкріплення. Це допомагає уникнути проблеми «reward sparsity» — коли винагорода є тільки в кінці епізоду, а в процесі немає підказок, чи дії правильні. Таке евристичне підкріплення дозволяє агенту легше зорієнтуватися в початкових фазах навчання.

Це м’яке підкріплення за те, що агент рухається у правильному напрямку (наприклад, наближається до мети). Значення 0.35 обране з кількох причин: воно в рази менше, ніж основна нагорода +5, щоб не замінити головну мету; воно достатньо велике, щоб бути помітним при оптимізації політики (особливо на ранніх етапах); забезпечує градієнтну зворотну реакцію — агент бачить «маленькі успіхи», навіть не досягнувши мети. У практиці RL часто беруть такі значення від 0.1 до 1.0. 0.35 — гарний компроміс: не домінує, але допомагає навчанню.

Штрафи в системі винагород реалізовані з метою покарання неефективної або небажаної поведінки. Часовий штраф, як у груповому, так і в індивідуальному сценарії, кожен крок епізоду карається невеликим мінусом, щоб агент не «гуляв» по сцені даремно. Неоптимальні дії або зіткнення, в агенті з класом CollisionDetect передбачено обробку зіткнень.

```
Сообщение Unity | Ссылок: 0
void OnCollisionEnter(Collision col)
{
    if (tags.Contains(col.gameObject.tag))
    {
        agent?.OnActionReceived(agent.GetStoredActionBuffers());
    }
}
```

Рис. 10 Приклад обробки зіткнення.

Таким чином можна налаштувати штраф за безглузді або надмірні зіткнення з блоками.

Візуалізація винагород, сума винагород упродовж епізодів відображається в реальному часі за допомогою скрипта VisualController. Така система вважається оптимальною, бо вона:

1) Має чітку мету - велике заохочення за успіх;
2) Змусить діяти швидко - через таймер і штраф;
3) Враховує колективний успіх - через групові нагороди;
4) Підтримує процес навчання - завдяки евристичній оцінці прогресу;
5) Сумісна з RL-алгоритмом PPO - який вимагає градієнтної структури винагород для стабільного оновлення політики.

Це дозволяє контролювати ефективність навчання та відслідковувати зміну стратегій під час тренувань. Система винагород у симуляції виконує не лише роль навчального сигналу, а й задає мотиваційну модель для агента або групи. Поєднання штрафів, проміжних заохочень та групової винагороди дозволяє моделювати кооперативну поведінку в умовах часткової обізнаності та обмежених дій. У процесі навчання система винагород стає основним джерелом сигналів, за допомогою яких агент формує свою політику поведінки у симульованому середовищі.

2.3.6 Реалізація середовища та експериментальні результати.

Для запуску процесу навчання агентів у симуляційному середовищі Unity було використано команду: mlagents-learn PushBlockCollab.yaml --run-id=PushBlock_01 –force

Після виконання команди відбувся запуск ML-Agents тренера, про що свідчить відповідне ASCII-лого Unity та базова інформація про версію інструментів:

ml-agents: 1.1.0

ml-agents-envs: 1.1.0

Communicator API: 1.5.0

PyTorch: 2.6.0+cpu

Система повідомила, що слухає порт 5004 і очікує підключення з Unity Editor. Було завантажено налаштування тренування для поведінки

PushBlockCollab, які включають:

Алгоритм тренування: ppo (Proximal Policy Optimization).

Параметри гіперпараметрів:

batch_size: 1024 — кількість прикладів, що обробляються за одну ітерацію оновлення політики.

buffer_size: 10240 — розмір буфера зібраних даних перед оновленням моделі.

learning_rate: 0.0003 — швидкість навчання, визначає розмір кроку при оновленні ваг мережі.

beta: 0.005 — коефіцієнт регуляризації для збереження ентропії політики.

epsilon: 0.2 — межа для кліппінгу оновлень політики.

lambd: 0.95 — коефіцієнт для обчислення переваг в методі GAE (Generalized Advantage Estimation).

num_epoch: 3 — кількість проходів по буферу даних на одне оновлення.

Архітектура нейронної мережі:

hidden_units: 128 — кількість нейронів у прихованому шарі.

num_layers: 2 — кількість прихованих шарів у мережі.

vis_encode_type: simple — тип кодування вхідних даних (простий).

Reward Signals:

extrinsic — використовуються зовнішні нагороди із середовища для навчання.

gamma: 0.99 — коефіцієнт дисконтування нагород (впливає на вагу майбутніх нагород).

Також налаштовано параметри збереження чекпоїнтів (checkpoint_interval = 500000) та максимальну кількість кроків (max_steps = 5000000).

Під час процесу тренування система почала виводити проміжні результати:

На кроці 10000:

Mean Reward = 1750.0

Std of Reward = 0.000

На кроці 20000, 30000 і 40000:

Mean Reward = 1750.0

Std of Reward = 0.000

Це свідчить про те, що агент досить швидко навчився оптимальній політиці поведінки у даному середовищі: середня винагорода стала стабільною, а стандартне відхилення дорівнює нулю, що вказує на те, що поведінка агента стала передбачуваною і надійною у досягненні поставленої цілі.

Попередження [WARNING] повідомляє, що для ще кращого результату у сценаріях із кількома агентами рекомендується використовувати POCA-тренер, однак у рамках даного проєкту використання PPO є цілком достатнім і виправданим для демонстрації основних принципів навчання з підкріпленням.

Рис. 11 Результати запуску навчального середовища та проміжні результати тренування.

2.5. Аналіз ефективності моделі

2.5.1. Вплив архітектурних рішень на ефективність

Архітектурні рішення, прийняті під час розробки симуляційної системи для навчання агентів, суттєво вплинули на ефективність процесу тренування та кінцеві результати роботи агентів. Нижче розглянемо основні аспекти архітектури і їхній внесок у підвищення продуктивності системи.

Модульна структура середовища - розподіл системи на окремі модулі (агенти, об’єкти, контролер середовища, система винагород) забезпечив:

1) Гнучкість: можливість легко змінювати компоненти без порушення роботи всієї системи.

2) Простоту розширення: можна безперешкодно додати нових агентів або модифікувати існуючі об'єкти.

3) Легкість у тестуванні: кожен модуль можна окремо перевіряти, що скорочує час пошуку і виправлення помилок.

Вибір дискретного набору дій для агентів (рух уперед, назад, вліво, вправо, обертання) допоміг. Зменшити розмірність простору дій, що суттєво пришвидшило навчання. Забезпечити більш передбачувану і стабільну поведінку агентів під час симуляції. Спрощувати стратегію пошуку оптимальних рішень.

Реалізація фізики у середовищі. Застосування базових фізичних властивостей (маса об’єктів, тертя, обмеження швидкості) дозволило:

- Створити реалістичні умови взаємодії агентів із середовищем
- Уникнути неадекватних або неприродних стратегій навчання (наприклад, миттєвого пересування або проходження крізь об'єкти)
- Поліпшити узгодженість результатів симуляції з реальними сценаріями застосування
- Використання системи винагород

Правильне формулювання системи заохочень та штрафів відіграло вирішальну роль у швидкості навчання. Нагорода за досягнення мети стимулювала агентів ефективно взаємодіяти з об’єктами. Штрафи за помилки запобігали безглуздим або хаотичним діям. Групова винагорода сприяла розвитку кооперативної поведінки між агентами.

Вибір алгоритму навчання (PPO) - рішення використовувати алгоритм Proximal Policy Optimization (PPO) надало такі переваги:

- Стабільність навчання навіть у середовищах зі складними фізичними взаємодіями
- Ефективність оптимізації при помірних обчислювальних ресурсах
- Гнучкість налаштування через змінні гіперпараметри (learning rate, entropy coefficient тощо)

Висновок, отже, архітектурні рішення безпосередньо вплинули на: стабільність і передбачуваність результатів навчання, швидкість досягнення оптимальної політики поведінки, можливість масштабування рішення на складніші або реальні сценарії.

Грамотний вибір моделі дій, побудова середовища з фізичними обмеженнями та ретельна настройка системи винагород значно підвищили ефективність розробленої системи мультиагентного навчання.

2.5.2 Обговорення стабільності, швидкості навчання, кооперації

Під час проведення тренування агентів у симульованому середовищі було виявлено високу стабільність процесу навчання. Це підтверджується графіком середньої винагороди за епізод, який показує поступове зростання та вихід на

стабільний рівень без значних коливань або провалів. Така поведінка свідчить про те, що використаний алгоритм Proximal Policy Optimization (PPO) забезпечує ефективне оновлення політик агентів без ризику катастрофічного розриву навчання.

Швидкість навчання оцінювалася за кількістю кроків, необхідних для досягнення стабільного рівня винагороди. Агентам вдалося виробити успішну стратегію взаємодії з об'єктами та середовищем за відносно помірну кількість епізодів. Використання дискретного простору дій і чітко визначеної системи винагород (заохочення за рух у напрямку цілі, штрафи за неефективні дії) суттєво зменшило обсяг необхідного для навчання часу.

Кооперація між агентами проявилася в узгоджених діях, спрямованих на колективне досягнення мети — переміщення об'єкта у зону цілі. Завдяки застосуванню групової винагороди агенти навчились координувати свої зусилля: уникати конфліктів і не заважати один одному під час маніпуляції з об'єктами. Це є важливим результатом, оскільки формування ефективної кооперативної поведінки в багатокомпонентних системах є одним із основних викликів у сфері навчання з підкріпленням.

2.5.3 Висновки щодо придатності системи для подальших досліджень

Результати проведеного навчання та експериментів свідчать про високу придатність розробленої системи для подальших наукових досліджень. Основні переваги, які це забезпечують:

1) Гнучкість архітектури: можливість легко модифікувати середовище та змінювати параметри задачі без потреби перебудовувати всю систему.
2) Адаптивність алгоритму: PPO виявив себе як стійкий до флуктуацій і здатний до швидкої адаптації навіть у динамічних умовах.

3) Підтримка багатокомпонентної взаємодії: ефективна організація колективної поведінки відкриває можливості для масштабування задач на більшу кількість агентів або для вирішення більш складних сценаріїв кооперації.

Можливість інтеграції з реальними сценаріями: використання ML-Agents та Unity дозволяє у майбутньому розширити дослідження на задачі реального світу, включно з робототехнікою, автономними системами та інтернетом речей (IoT).

Таким чином, розроблена система є не тільки прикладом успішної реалізації колективного навчання агентів, а й платформою для подальших експериментів у галузі багатозадачного навчання з підкріпленням та оптимізації кооперативних стратегій.

2.6. Результати тестування

Після завершення етапу розробки було проведено комплексне тестування системи мультиоб'єктного управління в середовищі Unity ML-Agents. Тестування включало як функціональні, так і нефункціональні аспекти, а також базувалося на перевірці інтерфейсів взаємодії агентів із середовищем.

2.6.1 Функціональне тестування

Функціональне тестування було спрямоване на перевірку відповідності поведінки системи функціональним вимогам, сформульованим під час розробки. Основна мета — переконатися, що система реалізує всі ключові

функції правильно та без помилок. Тестування проводилося в симульованому середовищі Unity з використанням ML-Agents, а також у кодовій частині на C#.

Переміщення агентів - було перевірено, як агенти реагують на дискретні дії, згенеровані або з боку користувача (у режимі Heuristic), або з боку навченої моделі.

- Очікуване: агент рухається вперед/назад, обертається, здійснює бічні зсуви відповідно до дій (1–6).
- Отримане: агент реагує на всі дії коректно, переміщається у фізичному просторі з урахуванням обмежень швидкості, визначеної у PushBlockSettings.

Взаємодія з блоками - тестувалася фізична взаємодія агента з об’єктом (блоком), який необхідно перемістити до цільової зони.

- Очікуване: агент має можливість штовхати блок, блок фізично відштовхується і може бути спрямований до цілі.
- Отримане: блок змінює положення під впливом сили, що створюється агентом, динаміка відповідає очікуваній поведінці в середовищі з фізикою Unity.

Обчислення винагород, було перевірено правильність нарахування винагород у відповідності до логіки, реалізованої у методі OnActionReceived.

- Очікуване: агент отримує позитивну винагороду, якщо відстань до цілі зменшилася, отримує негативну винагороду, якщо відстань збільшується, у разі досягнення цілі — групова винагорода.

- Отримане: всі винагороди відповідають очікуваним сценаріям. Також підтверджено, що зміна винагороди відбувається плавно, без різких стрибків.

Завершення епізоду. Перевірено, чи епізод завершується автоматично після досягнення цілі або при перевищенні максимальної кількості кроків (MaxEnvironmentSteps).

- Очікуване: при досягненні цілі всі агенти отримують винагороду, епізод перезапускається,якщо ціль не досягнута — застосовується "штраф за час", і епізод завершується.
- Отримане: перезапуск працює коректно, агенти ініціалізуються заново, позиції скидаються відповідно до логіки в ResetScene.

Групова взаємодія агентів, оскільки система реалізує кооперативне навчання, було протестовано взаємодію кількох агентів у середовищі:

- Очікуване: агенти не блокують один одного, можуть співпрацювати для досягнення спільної мети (наприклад, штовхати блок разом).
- Отримане: за налаштувань Unity Physics агенти не застрягають і не перешкоджають одне одному, що свідчить про відповідність групової динаміки очікуванням.

Поведінка в кутових випадках, було перевірено поведінку системи у "граничних" ситуаціях: блоки або агенти з’являються поблизу краю платформи, кілька агентів намагаються взаємодіяти з одним і тим же блоком, блок потрапляє у цільову зону, але з великим запізненням. Результат - система

коректно обробляє всі сценарії, не спостерігалося збоїв або непередбаченої поведінки.

2.6.2 Нефункціональне тестування

Нефункціональне тестування охоплює характеристики програмного забезпечення, які не пов’язані з конкретною функціональністю, але критично впливають на якість, стабільність та користувацький досвід. Для системи мультиоб’єктного управління у симульованому середовищі Unity було проведено тестування таких аспектів: продуктивність, стабільність, масштабованість, зручність інтеграції та відповідність інтерфейсам.

Продуктивність системи - метою цього тестування було перевірити, наскільки ефективно система функціонує за різного навантаження. Параметри тесту: кількість агентів: 1, 2, 5, 10; кількість блоків: 1, 2, 3; тривалість епізоду: 5000–25000 кроків. Результати: при кількості агентів до 5 система функціонує стабільно, без зниження FPS; при збільшенні кількості об’єктів до 10 можливе незначне зниження швидкодії у слабших системах, однак логіка взаємодії зберігається.

Стабільність роботи, було протестовано стабільність роботи під час тривалого навчання (до кількох тисяч епізодів), у тому числі: запусків Unity середовища з довготривалим підключенням Python-процесу; повторні запуски навчання (із застосуванням --resume або --force); перевірка відсутності збоїв при скиданні середовища. Результат: система показала стабільність під час довготривалих навчальних сесій. Не зафіксовано зависань, падінь або витоків пам’яті.

Масштабованість, було перевірено здатність системи до масштабування: додавання нових агентів не потребувало зміни базової архітектури, додаткові блоки додавалися без порушення структури винагород і логіки навчання, реалізовано динамічне скидання середовища з випадковим розміщенням об’єктів. Результат: архітектура виявилася масштабованою та придатною до розширення — можливе подальше нарощування складності без перегляду всієї моделі.

Інтерфейс користувача (UI/UX), оцінювалася простота візуалізації результатів у режимі реального часу:використано скрипт VisualController, який виводить епізод і винагороду, під час навчання зручно відстежувати прогрес агентів, кількість епізодів, накопичену винагороду. Результат: графічний інтерфейс не заважає навчанню, забезпечує зручне спостереження за процесом, що особливо корисно під час тестування моделей у Unity Editor.

Сумісність і інтеграція. Перевірено сумісність із:

- ML-Agents v1.1.0
- Python 3.10
- Unity 2021+
- сумісність YAML-файлів конфігурації з середовищем (PPO, настройки середовища, іменування поведінок).

Результат: система без проблем інтегрується з усіма зазначеними компонентами, що дозволяє легко запускати її у середовищах із різними конфігураціями.

2.6.3 Тестування інтерфейсів

Оцінювалася робота внутрішніх інтерфейсів взаємодії між компонентами системи:

1) Інтерфейс агента (PushAgentCollab): обробка вхідних дій та взаємодія з фізикою середовища
2) Контролер середовища (PushBlockEnvController): генерація епізодів, моніторинг винагород, ініціалізація початкових умов
3) Взаємодія із зовнішніми компонентами ML-Agents: коректна передача даних між Unity та Python-процесом для тренування моделей

Результат: усі інтерфейси функціонували належним чином, забезпечуючи стабільний обмін даними та коректне оновлення станів середовища.

3. ВИСНОВКИ

У ході виконання роботи було реалізовано повноцінну систему моделювання колективної поведінки віртуальних агентів у тривимірному середовищі з використанням навчання з підкріпленням. Основною платформою для розробки стала Unity у зв’язці з ML-Agents, що забезпечило гнучке середовище симуляції, повноцінну фізичну взаємодію об’єктів та інтеграцію із сучасними алгоритмами навчання.

Постановка задачі охоплювала розробку симульованого простору, в якому кілька агентів спільно досягають мети, впливаючи на фізичні об’єкти (блоки) з використанням дискретних дій. Було чітко сформульовано функціональні вимоги до системи — наявність контролера середовища, логіки винагород, адаптивної ініціалізації, оцінки прогресу та завершення епізодів.

Огляд підходів до навчання виявив значну різницю між одноагентними та багатоагентними архітектурами, а також між індивідуальними та кооперативними моделями підкріплення. У контексті цієї роботи було обрано саме кооперативний підхід із використанням групової винагороди, що дозволило формувати поведінку на основі колективного успіху.

Архітектура реалізованої моделі передбачала чітке розділення відповідальностей між агентами, об’єктами, контролером середовища та механізмом оцінки. Агент взаємодіє із середовищем через набір дискретних дій (рух уперед/назад, обертання, переміщення вбік), отримує спостереження та адаптує політику поведінки згідно з отриманими нагородами.

Як основний алгоритм навчання був обраний Proximal Policy Optimization (PPO), що відзначається стабільністю, ефективністю та здатністю до узагальнення. Цей метод добре підходить для складних середовищ, де є

безперервна взаємодія, а простір дій може бути як дискретним, так і неперервним. Особливу увагу приділено дизайну системи винагород, де були впроваджені:

1) негативні винагороди за рух у невірному напрямку
2) позитивні — за зменшення відстані до цілі
3) групова винагорода при успішному виконанні задачі
4) штрафи за бездіяльність або перевищення кількості кроків в епізоді

Результати навчання підтвердили ефективність такого підходу: графіки стабільності винагороди демонструють поступове зростання середньої винагороди з виходом на плато, що свідчить про досягнення стійкої політики поведінки агентів.

У рамках тестування було проведено як функціональний, так і нефункціональний аналіз роботи системи. Виявлено стабільність при масштабуванні (до 10 агентів), ефективну обробку дій у режимі реального часу, відповідність інтерфейсів взаємодії, а також зручну візуалізацію процесу навчання.

Окремо слід відзначити, що архітектурні рішення, такі як модульність середовища, розділення агентів і контролера, використання єдиного джерела налаштувань та параметрів навчання — позитивно вплинули на адаптивність системи, можливість її подальшої модифікації та повторного використання. З точки зору дослідження, отримана система: підтвердила працездатність концепції кооперативного навчання, продемонструвала потенціал масштабування та адаптації, може бути використана як основа для нових експериментів у сфері штучного інтелекту, робототехніки, ігрової розробки або колективного управління агентами.

Отже, основна частина роботи демонструє завершену розробку системи, яка поєднує теоретичні засади підкріпленого навчання, сучасні інструменти симуляції та ефективні інженерні рішення. Отримані результати підтверджують її прикладну та наукову цінність, відкриваючи широкі можливості для подальших досліджень.

4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. — 2nd edition. — Cambridge, MA: MIT Press, 2018.
2. Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach. — 4th edition. — Pearson, 2021.
3. Goodfellow I., Bengio Y., Courville A. Deep Learning. — Cambridge, MA: MIT Press, 2016.
4. Silver D., Schrittwieser J., Hubert T. et al. Mastering the game of Go without human knowledge // Nature. — 2017.
5. Liang Y., Machado M. C., Talvitie E., Bowling M. State of the Art Control of Atari Games Using Shallow Reinforcement Learning // Artificial Intelligence. — 2016.
6. Bishop C. M. Pattern Recognition and Machine Learning. — New York: Springer, 2006.
7. Millington I., Funge J. Artificial Intelligence for Games. — 2nd edition. — CRC Press, 2009.
8. Julian Togelius, Noor Shaker, Georgios N. Yannakakis. Artificial Intelligence and Games. — Springer, 2016.
9. Bonabeau E., Dorigo M., Theraulaz G. Swarm Intelligence: From Natural to Artificial Systems. — Oxford University Press, 1999.
10. Bergstra J., Yamins D., Cox D. D. Hyperparameter optimization in machine learning // Advances in Neural Information Processing Systems (NIPS). — 2013.
11. Unity Technologies. Unity ML-Agents: Documentation — Офіційна документація: https://docs.unity3d.com/Packages/com.unity.ml-agents@3.0/manual/index.html
    Chollet F. Deep Learning with Python. — 2nd edition. — Manning Publications, 2021.
12. Singh A., Lee Y. et al. Emergent coordination through competition. — 2017.