ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод симуляції екосистеми з використанням нейронних мереж»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Данило БУР'ЯНОВ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Данило БУР'ЯНОВ

Керівник: Данило КОВАЛЕНКО
доктор філософії (PhD)

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бур'янову Данилу Сергійовичу

1. Тема кваліфікаційної роботи: «Метод симуляції екосистеми з використанням нейронних мереж»

керівник кваліфікаційної роботи Данило КОВАЛЕНКО, доктор філософії (PhD),

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, біологічні параметри та поведінкові характеристики модельних видів, опис технологій та інструментарію Unity ML-Agents, гіперпараметри алгоритму Proximal Policy Optimization, конфігураційні файли для навчання нейронних мереж.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі (огляд методів моделювання екосистем, проблематика адаптивної поведінки, обґрунтування вибору навчання з підкріпленням).

2. Програмні засоби реалізації (архітектура інтегрованої системи, налаштування середовища Unity та ML-Agents, розробка агентів та навчального плану).
3. Дослідження отриманих результатів (аналіз ефективності Curriculum Learning, емерджентна поведінка, оцінка стабільності системи).

5. Перелік ілюстративного матеріалу: *презентація*

1. Актуальність та порівняльний аналіз методів моделювання.
2. Опис характеристик та параметрів симуляції.
3. Схема задачі навчання агентів та система винагород.
4. Архітектура системи та модель прийняття рішень.
5. Блок-схема алгоритму Proximal Policy Optimization (PPO).
6. Структура розробленого методу поетапного навчання (Curriculum Learning).
7. Алгоритм динамічної рандомізації середовища.
8. Візуалізація роботи сенсорів та практична реалізація.
9. Графіки порівняння стабільності навчання.
10. Порівняльна таблиця показників ефективності методів.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10 – 05.11.2025	
2	Аналіз методів моделювання екосистем та алгоритмів навчання з підкріпленням	06.11 – 12.11.2025	
3	Розробка архітектури системи та моделі агентів у середовищі Unity 3D	13.11 – 20.11.2025	
4	Програмна реалізація логіки агентів та налаштування гіперпараметрів навчання	21.11 – 27.11.2025	
5	Впровадження методики поетапного навчання (Curriculum Learning) та налаштування сценаріїв	28.11 – 08.12.2025	
6	Проведення експериментів, аналіз стабільності та адаптивності отриманих моделей	09.12 – 12.12.2025	
7	Оформлення роботи: вступ, висновки, реферат	13.11 – 16.12.2025	
8	Розробка демонстраційних матеріалів	17.12 – 19.12.2025	

Здобувач вищої освіти _____ Данило БУР’ЯНОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Данило КОВАЛЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 85 стор., 5 табл., 18 рис., 50 джерел.

Мета роботи – підвищення стабільності та адаптивності симуляції екосистеми на основі використання нейронних мереж за рахунок поетапного навчання RL-агентів.

Об'єкт дослідження – процес симуляції екосистеми.

Предмет дослідження – моделі та метод симуляції екосистеми з використанням нейронних мереж.

Короткий зміст роботи: у роботі проведено системний аналіз методів моделювання складних адаптивних систем та обґрунтовано переваги використання глибокого навчання з підкріпленням над класичними детермінованими алгоритмами. Виявлено проблему «розріджених винагород» та «холодного старту» при навчанні агентів у стохастичних середовищах.

Спроектовано та реалізовано гібридну архітектуру симуляційного комплексу, що поєднує фізичний рушій Unity 3D для генерації даних та бібліотеку PyTorch для навчання нейронних мереж. Розроблено математичні моделі агентів «вовк» та «заєць» у безперервному просторі дій. Як базовий алгоритм оптимізації політики обрано Proximal Policy Optimization, що забезпечує стабільність градієнтного спуску.

Для вирішення проблеми нестабільності навчання розроблено та впроваджено методику навчання за планом, яка декомпозує глобальну задачу виживання на три рівні складності: базова навігація, адаптація до загроз та конкурентне середовище.

Результатом дослідження експериментально підтверджено ефективність запропонованого методу порівняно з базовим підходом. Встановлено, що використання Curriculum Learning дозволяє скоротити час збіжності алгоритму у

4.1 рази. Забезпечено підвищення показника виживання популяції з 0% (повне вимирання у базовому методі) до 56% у фінальній моделі. Зафіксовано появу стійкої емерджентної поведінки, яка не була запрограмована явно: використання ландшафту для розриву лінії зору, стратегія полювання із засідки біля водойм та ефект колективної втечі.

Подальші дослідження можуть бути спрямовані на інтеграцію рекурентних нейронних мереж для реалізації пам’яті агентів, впровадження методів мультиагентної кооперації та розширення біорізноманіття модельованої екосистеми.

Результати роботи можуть бути використані для створення інтерактивних віртуальних лабораторій, модернізації алгоритмів поведінки штучного інтелекту в ігрових проектах та розробки систем екологічного прогнозування.

КЛЮЧОВІ СЛОВА: СИМУЛЯЦІЯ ЕКОСИСТЕМИ, НЕЙРОННІ МЕРЕЖІ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, RL-АГЕНТИ, ПОЕТАПНЕ НАВЧАННЯ, МУЛЬТИАГЕНТНІ СИСТЕМИ, АДАПТИВНІСТЬ.

ABSTRACT

Text part of the master's qualification work: 85 pages, 18 pictures, 5 table, 50 sources.

The purpose of the work is increasing the stability and adaptability of ecosystem simulation using neural networks through step-by-step training of RL agents.

Object of research is ecosystem simulation process.

Subject of research is models and method of ecosystem simulation using neural networks.

Summary of the work: The work presents a systematic analysis of methods for modeling complex adaptive systems and substantiates the advantages of using deep reinforcement learning over classical deterministic algorithms. The problem of "sparse rewards" and "cold start" when training agents in stochastic environments is identified.

A hybrid architecture of a simulation complex is designed and implemented, combining the Unity 3D physical engine for data generation and the PyTorch library for training neural networks. Mathematical models of "wolf" and "hare" agents in a continuous action space are developed. Proximal Policy Optimization is chosen as the basic policy optimization algorithm, which ensures the stability of gradient descent.

To solve the problem of learning instability, a planned learning methodology has been developed and implemented, which decomposes the global survival problem into three levels of complexity: basic navigation, adaptation to threats, and a competitive environment.

The study results experimentally confirm the effectiveness of the proposed method compared to the basic approach. It was found that the use of Curriculum Learning allows to reduce the convergence time of the algorithm by 4.1 times. The population survival rate was increased from 0% (complete extinction in the basic method) to 56% in the final model. The emergence of stable emergent behavior that was

not explicitly programmed was recorded: the use of the landscape to break the line of sight, the strategy of hunting from an ambush near water bodies and the effect of collective escape.

Further research can be aimed at integrating recurrent neural networks to implement agent memory, implementing multi-agent cooperation methods and expanding the biodiversity of the modeled ecosystem.

The results of the work can be used to create interactive virtual laboratories, modernize artificial intelligence behavior algorithms in game projects and develop ecological forecasting systems.

KEYWORDS: ECOSYSTEM SIMULATION, NEURAL NETWORKS, REINFORCEMENT LEARNING, RL AGENTS, CURRICULUM LEARNING, MULTI-AGENT SYSTEMS, ADAPTABILITY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ABM – Agent-Based Modeling

ADW – Animal Diversity Web

API – Application Programming Interface

CA – Cellular Automata

CNN – Convolutional Neural Network

CPU – Central Processing Unit

EBM – Equation-Based Modeling

FSM – Finite-State Machine

GAE – Generalized Advantage Estimation

GPU – Graphics Processing Unit

GRU – Gated Recurrent Units

IL – Imitation Learning

LSTM – Long Short-Term Memory

MARL – Multi-Agent Reinforcement Learning

MDP – Markov Decision Process

ML – Machine Learning

NPC – Non-Player Character

OODA – Observe-Orient-Decide-Act

POMDP – Partially Observable Markov Decision Process

PPO – Proximal Policy Optimization

RL – Reinforcement Learning

ROS – Robot Operating System

Sim2Real – Simulation to Reality

ВСТУП

Сучасні екосистеми є складними нелінійними динамічними системами, у яких взаємодія видів, конкуренція за ресурси, трофічні ланцюги та адаптивна поведінка організмів формують складні емерджентні процеси. Традиційні математичні моделі, зокрема рівняння Лотки–Вольтерри та їхні модифікації, добре описують загальні тенденції популяційної динаміки, однак часто виявляються недостатніми для моделювання індивідуальної поведінки агентів, адаптації до змінного середовища, стратегічної взаємодії та формування складних патернів виживання [1]. У цьому контексті особливого значення набувають мультиагентні системи та методи машинного навчання, зокрема навчання з підкріпленням, що дозволяють моделювати гнучкі адаптивні екологічні сценарії.

Застосування нейронних мереж у симуляції екосистем відкриває можливість дослідження поведінки окремих агентів на мікрорівні, аналізу впливу поведінкових стратегій на стабільність популяцій, вивчення процесів саморегуляції та адаптації. Поєднання агентно-орієнтованого підходу та глибинного навчання дає змогу створювати комп’ютерні моделі, здатні самостійно знаходити ефективні стратегії виживання в умовах конкуренції та хижацтва. Це робить такі моделі корисними не лише для дослідницьких задач, а й для практичних застосувань – від екологічного прогнозування до створення віртуальних лабораторій.

Попри значний прогрес у галузі машинного навчання та екологічного моделювання, актуальною залишається проблема стабільності мультиагентних симуляцій. Багато моделей стикаються з нестійкими траєкторіями навчання, колапсом популяцій, виникненням надмірно агресивних стратегій або неможливістю сформувати стійку рівновагу між видами. Це ускладнює аналіз поведінки системи та знижує практичну цінність моделі. Тому виникає потреба у методах, які дозволяють підвищити стабільність екосистем під час навчання агентів.

Об'єкт дослідження – процес симуляції екосистеми.

Предмет дослідження – моделі та метод симуляції екосистеми з використанням нейронних мереж.

Мета роботи – підвищення стабільності та адаптивності симуляції екосистеми на основі використання нейронних мереж за рахунок поетапного навчання RL-агентів.

Завдання дослідження:

1) Проаналізувати існуючі методи моделювання мультиагентних систем та виявити фактори, що впливають на збіжність алгоритмів навчання з підкріпленням.

2) Обґрунтувати та спроектувати гібридну архітектуру програмного комплексу на базі рушія Unity та бібліотек машинного навчання Python.

3) Розробити математичні та програмні моделі агентів із використанням нейронних мереж, визначивши структуру їх сенсорних систем та простору дій.

4) Розробити алгоритмічне забезпечення для стабілізації навчання, що включає метод навчання за планом (Curriculum Learning) та систему динамічної рандомізації середовища.

5) Провести експериментальне дослідження розробленої системи, виконати порівняльний аналіз ефективності прямого та поетапного методів навчання.

Наукова новизна полягає у розробці методу симуляції екосистеми з поетапним навчанням агентів, що поєднує мультиагентну динаміку та глибоке навчання з підкріпленням (Deep RL). Вперше застосовано комбінацію алгоритму PPO з методикою динамічної рандомізації та навчання за розкладом (Curriculum Learning) для біологічних агентів у безперервному просторі дій. Запропонований метод дозволяє вирішити проблему «холодного старту», уникнути колапсу популяції на ранніх етапах навчання та забезпечити формування емерджентних поведінкових стратегій з чистого аркуша.

Розроблена симуляція та програмне забезпечення можуть бути використані як навчальний інструмент для демонстрації принципів роботи штучного інтелекту

та еволюційних процесів. Методика поетапного навчання агентів може знайти застосування у розробці відеоігор для створення адаптивних NPC, а також у робототехніці для навчання автономних систем у складних середовищах. Результати роботи підтверджують ефективність запропонованого підходу для моделювання динамічних систем без необхідності написання жорсткого коду поведінки.

Гіпотеза дослідження полягає в тому, що навчання за планом (Curriculum Learning) дозволить агентам подолати "спіраль смерті", спричинену надто складною початковою задачею, та виробити стабільні, збалансовані стратегії виживання, які неможливо отримати при "прямому" навчанні з нуля.

За темою магістерської роботи опубліковано 1 статтю у фаховому виданні та 2 тези доповідей у збірниках матеріалів конференцій.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ МОДЕЛЮВАННЯ ЕКОСИСТЕМ

1.1 Загальна постановка задач симуляції у різних середовищах

Комп'ютерне моделювання посідає важливе місце в сучасних наукових дослідженнях, особливо в екології, де вони слугують інструментом для аналізу складних багаторівневих взаємодій між біологічними, фізичними та антропогенними факторами. Екосистеми, будучи основою життя на планеті, мають високий рівень структурної складності, коли навіть незначні зміни в одному компоненті можуть запускати каскадний ефект, що впливає на все біорізноманіття.

З точки зору теорії систем, природні екосистеми класифікуються як складні адаптивні системи. Їм притаманні нелінійність, емерджентність та здатність до самоорганізації. Традиційні аналітичні методи, що базуються на редукціонізмі, виявляються безсилими перед такими об'єктами, оскільки властивості цілої системи не є простою сумою властивостей її елементів. Саме тому симуляційне моделювання стає безальтернативним методом дослідження, дозволяючи відтворити емерджентні ефекти – глобальні патерни поведінки популяції, що виникають з локальних взаємодій окремих індивідів.

Історичний досвід надає переконливі приклади катастрофічних наслідків недооцінки складності екологічних зв'язків. Інтродукція інвазійного виду – тростинкової жаби (Rhinella marina) – під час колонізації Австралії призвела до руйнівного впливу на місцеву фауну, що не мала еволюційних механізмів захисту [2]. Аналогічно, масова кампанія з винищення горобців в Китаї (1958-1962) спричинила порушення трофічного ланцюга, що вилилося в неконтрольоване розмноження комах-шкідників та подальшу продовольчу кризу [3].

Ці історичні прецеденти актуалізують необхідність переходу до експериментів «in silico» (у кремнії, тобто на комп'ютері). На відміну від натурних досліджень, які часто є неможливими через етичні обмеження, високу вартість або ризик незворотних змін у довкіллі, комп'ютерна симуляція надає безпечний полігон

для перевірки гіпотез. Ключовою перевагою віртуального моделювання є можливість маніпулювання часовим масштабом. Еволюційні процеси або наслідки кліматичних змін, які в реальності тривають десятиліттями, у симуляційному середовищі можуть бути відтворені за лічені години. Це дозволяє дослідникам проводити сценарний аналіз типу «що-якщо», оцінюючи, наприклад, вплив вирубки лісів або інтродукції нових хижаків на стійкість екосистеми Волині ще до того, як ці події відбудуться в реальності.

Переважна більшість існуючих екологічних моделей, як у академічній науці, так і в індустрії розваг (наприклад, у серіях ігор Far Cry чи Red Dead Redemption), побудовані на архітектурі скінченних автоматів [4]. Фундаментальне обмеження таких моделей полягає в тому, що агент (наприклад, тварина) може діяти лишень у рамках жорстко прописаних правил (на кшталт: «ЯКЩО бачу ворога – ТІКАТИ/НАПАДАТИ»).

У контексті програмної інженерії, підхід на основі скінченних автоматів або дерев поведінки стикається з проблемою «комбінаторного вибуху». Зі зростанням складності віртуального світу розробникам доводиться вручну прописувати правила для кожної можливої ситуації взаємодії агента із середовищем. Це призводить до створення громіздких, важких для підтримки кодових баз, які, тим не менш, не можуть покрити всі нюанси реальної поведінки. Як наслідок, у сучасних інтерактивних додатках спостерігається криза «штучного інтелекту», коли графічна складова досягає фотореалізму, а поведінка неігрових персонажів залишається шаблонною та передбачуваною. Подібний підхід виключає можливість справжньої адаптивної поведінки, коли організм може змінювати стратегію у відповідь на нові, непередбачені розробником виклики.

Ще однією важливою сферою застосування розроблюваних методів є робототехніка, зокрема напрямок автономної навігації безпілотних систем. Проблема навчання роботів у реальному світі пов'язана з високими ризиками пошкодження дороговартісного обладнання та неможливістю створення небезпечних аварійних ситуацій для тренування. Тут застосовується концепція Sim2Real (Simulation to Reality), яка передбачає навчання нейронних мереж у

високоточній фізичній симуляції з подальшим перенесенням «мозку» агента на фізичний пристрій. Розробка адаптивних агентів-тварин, здатних орієнтуватися у складному лісовому ландшафті Волині, уникати динамічних перешкод (хижаків) та знаходити цілі (їжу), має прямі паралелі із задачами навігації пошуково-рятувальних дронів або логістичних роботів. Тому алгоритмічні рішення, запропоновані в даній роботі, мають потенціал трансферу технологій у суміжні інженерні галузі.

Актуальність даного дослідження безпосередньо пов'язана із застосуванням парадигми навчання з підкріпленням. Цей підхід концептуально близький до механізмів еволюції в природі, яка поєднує випадкові та невипадкові процеси. Подібно до того, як випадкові мутації генерують генетичну мінливість, а невипадковий природний відбір сприяє найбільш життєздатним варіантам, RL поєднує дослідницьку випадковість з цілеспрямованим використанням найефективніших стратегій. На противагу класичному «програмуванню» поведінки, цей підхід передбачає «навчання» агентів. Створюємо інтелектуальні агенти, здатні самостійно виробляти складні стратегії виживання, керуючись системою винагород, що імітує природні механізми позитивного та негативного підкріплення. Таким чином, процес не є випадковим – він спрямований до цілі (максимізації винагороди), але використовує випадковість для відкриття нових шляхів її досягнення, що й призводить до справжніх адаптацій.

Виходячи з вищезазначеного, розробка методу симуляції, який поєднує сучасні ігрові рушії для візуалізації та алгоритми глибокого навчання для керування поведінкою, є актуальною науково-технічною задачею. Технічною основою для реалізації цієї концепції в даній роботі слугує ігровий рушій Unity та спеціалізований інструментарій Unity ML-Agents Toolkit [5]. Ця платформа дозволяє інтегрувати реалістичне 3D-середовище, що моделює ландшафти Волині, з потужними алгоритмами глибокого навчання, зокрема Proximal Policy Optimization (PPO) [6], які виконуються в середовищі Python з використанням бібліотеки PyTorch [7]. Кінцевою метою проекту є створення інтерактивної симуляції екосистеми Волині, яка б відображала динаміку взаємин між ключовими

видами. Для забезпечення реалізму моделі такі параметри агентів, як швидкість переміщення та інтенсивність метаболізму, були взяті з наукових джерел, зокрема з бази даних Animal Diversity Web [8].

1.2 Огляд існуючих методів моделювання екосистем

Історично склалося два основних підходи до моделювання динаміки популяцій та взаємодії «хижак-жертва»: аналітичні математичні моделі та агентне моделювання.

Моделі на основі диференціальних рівнянь (ЕВМ). Фундаментальною основою теоретичної екології є модель Лотки-Вольтерри, запропонована незалежно Альфредом Лоткою та Віто Вольтеррою. Вона описує взаємодію двох видів за допомогою системи диференціальних рівнянь першого порядку (1.1) [9]:

$$\frac{dx}{dt} = \alpha x - \beta xy$$
$$\frac{dy}{dt} = \delta xy - \gamma y, \quad (1.1)$$

де x – кількість жертв; y – кількість хижаків; коефіцієнти α, β, δ, γ – коефіцієнти народжуваності, смертності та ефективності полювання.

Переваги підходу: головною перевагою ЕВМ є обчислювальна ефективність. Вирішення системи рівнянь не вимагає значних ресурсів навіть для великих популяцій, оскільки оперує агрегованими показниками. Крім того, цей підхід дозволяє проводити аналітичне дослідження стійкості системи.

Недоліки та обмеження: критичним недоліком ЕВМ є припущення про гомогенність (однорідність) популяції. У такій моделі кожен вовк є ідентичною копією іншого, з однаковими шансами спіймати зайця. Це ігнорує просторову структуру (хижак і жертва можуть бути в різних кутках лісу), індивідуальні відмінності (вік, здоров'я, досвід) та локальні взаємодії. Як наслідок, ЕВМ добре описує глобальні тренди, але не здатна змоделювати складну адаптивну поведінку.

Хоча ця модель успішно демонструє циклічні коливання чисельності популяцій, вона має суттєві обмеження для створення реалістичних симуляцій. Як

зазначають дослідники агентного моделювання, рівняння базуються на припущенні «середнього поля» (mean-field approximation) [10]. Це означає, що модель ігнорує просторову структуру середовища, індивідуальні особливості тварин та локальні взаємодії. У такій моделі "вовк" – це лише абстрактна змінна, яка не має тіла, зору чи стратегії полювання.

Клітинні автомати (Cellular Automata). Спробою врахувати просторовий аспект стало використання клітинних автоматів. У цьому підході екосистема представляється як дискретна сітка, де кожна клітинка має певний стан (наприклад: «трава», «заєць», «вовк», «пусто»). Стан клітинки у момент часу t+1 визначається станом її сусідів у момент t за набором детермінованих правил.

Відомим прикладом є модель «Wa-Tor», розроблена А.К. Дьюдні (1984) [11], яка симулює динаміку акул та риб на тороїдальній сітці.

Переваги підходу: клітинні автомати дозволяють моделювати просторові ефекти, такі як поширення хвиль популяції, кластеризація ресурсів або фрагментація ареалу. Вони є інтуїтивно зрозумілими та легко реалізуються програмно.

Недоліки та обмеження: головним обмеженням є дискретність простору та часу. Рух тварин у СА є неприродним (стрибки між клітинками), що унеможливлює реалізацію реалістичної кінематики, інерції або складних траєкторій переслідування. Крім того, правила взаємодії зазвичай є локальними та жорстко заданими, що обмежує можливість навчання агентів.

Більш сучасним підходом є агентно-орієнтоване моделювання (АВМ) [12], воно базується на ідеї, що складна поведінка системи може виникати з простих локальних правил взаємодії автономних агентів. У контексті екосистем агентами виступають організми (хижак, жертва, рослина), які мають власні характеристики, фізіологію, поведінку та правила самостійного прийняття рішень.

Основні елементи АВМ: агент – індивідуальна одиниця з власними станами, середовище – просторове поле, яке впливає на агента, правила взаємодії – механізм комунікації агентів, поведінкова модель – набір алгоритмів прийняття рішень.

Поняття емерджентності займає центральне місце в теорії агентного моделювання. Емерджентні ефекти – це макроскопічні патерни, що виникають внаслідок мікроскопічних взаємодій і не можуть бути виведені шляхом простого сумування властивостей окремих агентів. Прикладами таких явищ в екології є формування зграй, самоорганізація міграційних потоків, виникнення стійких циклів хижак-жертва або раптові колапси популяцій. Важливо зазначити, що АВМ дозволяє досліджувати не лише рівноважні стани, на яких зосереджена більшість аналітичних моделей, але й динаміку перехідних процесів, нерівноважну термодинаміку живих систем та ефекти "метелика", коли незначні локальні збурення призводять до глобальних змін у структурі мережі взаємодій.

Ключовою перевагою АВМ в екології є можливість врахувати гетерогенність особин всередині популяції (наприклад, різний вік, розмір, досвід, генетичні особливості) та просторову експліцитність. Агенти взаємодіють у конкретному ландшафті, який також може моделюватися як активне середовище з ресурсами, перешкодами та мінливими умовами. Це відкриває шлях для моделювання складних сценаріїв: впливу фрагментації середовищ існування, поширення інвазивних видів, адаптації до змін клімату або ефективності різних стратегій охорони природи. Модель можна калібрувати на реальних даних, а потім проводити численні експерименти, що неможливо в натурних умовах.

Стохастичність є невід'ємною рисою агентних моделей. На відміну від детермінованих аналітичних розв'язків, АВМ використовує методи Монте-Карло для введення елементу випадковості у процеси прийняття рішень, руху або розмноження агентів. Це дозволяє моделювати природну невизначеність та оцінювати ймовірнісний розподіл можливих сценаріїв розвитку екосистеми. Такий підхід робить АВМ ідеальним інструментом для проведення контрфактичних експериментів – досліджень сценаріїв "що, якщо", які є неможливими або неетичними в реальному світі, наприклад, моделювання епідемій або інтродукції небезпечних інвазивних видів. Порівняльний аналіз методів моделювання екосистем наведено в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз методів моделювання екосистем

Критерій порівняння	ЕВМ (Диф. рівняння)	Клітинні автомати (CA)	Агентне моделювання (ABM)
Рівень абстракції	Макро-рівень (популяція)	Мезо-рівень (локальні групи)	Мікро-рівень (індивід)
Просторова структура	Відсутня (Mean Field)	Дискретна сітка	Безперервний простір
Гетерогенність агентів	Низька (ідентичні)	Середня (стани клітин)	Висока (унікальні параметри)
Поведінка	Детермінована	Стохастична/Правила	Адаптивна/Інтелектуальна
Обчислювальна вартість	Низька	Середня	Висока
Можливість навчання (RL)	Неможлива	Обмежена	Повна підтримка

Агенти функціонують не у вакуумі, а в конкретному просторовому та часовому контексті. Середовище в ABM виступає не просто пасивним контейнером, а активним учасником взаємодій, що має власні динамічні характеристики (зміна температури, відновлення ресурсів, рельєф). Взаємодія агентів із середовищем та між собою визначається топологією зв'язків, яка може бути реалізована через просторове сусідство (наприклад, у клітинних автоматах або безперервному 3D-просторі) або через соціальні мережі.

Для систематизації переваг та недоліків розглянутих підходів доцільно провести порівняльний аналіз за критеріями масштабованості, точності та

обчислювальної складності. Як видно з проведеного огляду, класичні диференціальні рівняння забезпечують високу швидкість обчислень, але втрачають локальний контекст взаємодій. Клітинні автомати (Cellular Automata) дозволяють врахувати просторову структуру, проте дискретність їх станів обмежує моделювання плавних рухів тварин. Агентно-орієнтований підхід, реалізований у даній роботі, хоч і вимагає значних обчислювальних ресурсів, є єдиним методом, здатним відтворити індивідуальну варіативність поведінки

До недоліків можна віднести наступне: висока обчислювальна складність при великій кількості агентів; потреба в складному калібруванні; багатоваріантність моделей, що ускладнює формальну валідацію. Популярні інструменти ABM наведено в таблиці 1.2.

Таблиця 1.2

Популярні інструменти ABM

Платформа	Сильні сторони
NetLogo	академічні дослідження, простота
AnyLogic	промислові моделі, гібридні симуляції
Unity ML-Agents	3D-фізика, learning agents
MASON / Repast	наукові гнучкі фреймворки

1.3 Проблематика моделювання адаптивної поведінки

Ключовим викликом при створенні агентно-орієнтованих моделей екосистем є вибір архітектури прийняття рішень. Від обраного методу залежить не лише обчислювальна ефективність симуляції, але й, що більш важливо, здатність системи до генерації валідних поведінкових патернів, які корелюють з реальною біологічною активністю. У сучасній інженерії програмного забезпечення виділяють

два полярних підходи до вирішення цієї задачі: евристичні методи на основі правил та методи машинного навчання.

Історично першим та найбільш поширеним методом є використання скінченних автоматів (Finite State Machines, FSM). У цій парадигмі поведінка агента описується графом, де вузли представляють стани (наприклад, «Патрулювання», «Переслідування», «Втеча»), а ребра визначають умови переходу між ними. Хоча FSM забезпечують повну детермінованість та прозорість логіки, вони страждають від проблеми комбінаторного вибуху при масштабуванні. Зі збільшенням кількості факторів середовища (голод, спрага, наявність хижаків, рельєф, час доби) кількість необхідних переходів зростає експоненційно, що робить підтримку такої системи неможливою. Більше того, FSM не здатні до адаптації: агент, запрограмований тікати від вовка по прямій лінії, буде повторювати цю дію навіть якщо вона призводить до глухого кута, оскільки в його логіці відсутній механізм оцінки наслідків.

Однак саме ця структурованість і стає головним обмеженням для моделювання життєподібної, адаптивної поведінки. Скінченний автомат оперує в жорстко заданому просторі можливостей. Ця концепція, запозичена з теорії обчислень, надала перший чіткий формалізм для опису поведінки. Нині FSM – це кістки та сухожилля штучного інтелекту в симуляціях: логіка NPC у RPG-іграх [13], цикл «паслись-втекли» у модельованих оленях, навіть алгоритми торгових ботів на віртуальних ринках.

Моделюючи життя, використовуємо архітектуру, за своєю суттю неживу. FSM визначає поведінку таксі, а не таксиста – він сліпо виконує інструкції, не володіючи ні метою, ні внутрішнім мотивом, ні здатністю виходити за межі своєї програми для досягнення цієї мети в мінливому середовищі.

Звичайні світлофори на перехресті – це класичний приклад FSM у дії. Світлофор може перебувати в одному з обмеженої (скінченної) набору станів: «зелений», «жовтий» або «червоний». Кожен стан визначає конкретну дію системи – у цьому випадку, яке світло горить. Перехід між цими станами відбувається у відповідь на чітко визначені події-входи (inputs). Для світлофора такою подією є

спрацювання таймера: «ЯКЩО минуло 50 секунд у стані «зелений», ТО перейти до стану «жовтий»». Ця логіка може бути викладена в таблиці 1.3.

Таблиця 1.3

Стани переходів світлофорів

Поточний стан	Вхідна умова, час (с)	Наступний стан	Вихідна поведінка
Зелений	50	Жовтий	Увімкнути жовте світло
Жовтий	8	Червоний	Увімкнути червоне світло
Червоний	32	Зелений	Увімкнути зелене світло

Саме за цією простою, але потужною схемою «стан-перехід» працюють і агенти в екологічних симуляціях. Поведінка віртуальної тварини, наприклад, зайця, описується набором станів: «блукати», «шукати їжу», «тікати від хижака». Перехід між ними відбувається за прописаними правилами: «ЯКЩО в радіусі 10 метрів з'являється вовк (вхід), ТО перейти зі стану «блукати» у стан «тікати»».

Перевага FSM – його детермінованість та передбачуваність, але водночас є його головним недоліком при спробі моделювати живі, динамічні системи. Агент, керований FSM, нагадує маріонетку, нитки якої прив'язані до обмеженого набору умов. Він нездатний до онтогенетичного навчання – тобто, до набуття індивідуального досвіду та адаптації власної поведінки впродовж її «життєвого циклу».

Агент FSM не може оцінити ступінь загрози чи вигоди. Наприклад, правило «тікати від хижака» завжди виконується з однаковою інтенсивністю, незалежно від того, чи знаходиться хижак за 20 метрів і біжить у протилежному напрямку, чи він за 3 метри і готовий до стрибка. Справжня тварина в цій ситуації може

приховатися, завмерти або втекти обережно, оцінивши контекст – варіантів, які не були закладені в початковий дизайн FSM.

Еволюційним розвитком евристичних методів стали Дерева Поведінки та Планування цілей. Ці архітектури, широко застосовані в ігровій індустрії (зокрема в серіях Halo та F.E.A.R [14].), дозволяють створювати більш модульну та ієрархічну структуру прийняття рішень. Агент з BT може динамічно перемикатися між підзадачами, перевіряючи умови від кореня дерева до його листків. Проте, як і FSM, ці методи залишаються статичними. Вони оперують знаннями, закладеними розробником на етапі проектування, і не можуть генерувати нові стратегії на етапі виконання. Це створює так званий «розрив реальності», коли поведінка моделі стає неадекватною при зміні умов середовища.

Альтернативою детермінованим алгоритмам виступає парадигма когнітивного моделювання, яка часто описується через цикл OODA (Observe-Orient-Decide-Act), запропонований військовим стратегом Джоном Бойдом, який зображено на рисунку 1.1 [15]. У контексті екологічної симуляції цей цикл передбачає безперервну обробку сенсорної інформації, оновлення внутрішньої моделі світу, вибір дії та її виконання. Для реалізації такого підходу в умовах невизначеності найбільш ефективним є використання Навчання з Підкріпленням (Reinforcement Learning, RL).

Рис. 1.1 Схема когнітивного циклу OODA

Процес функціонування агента можна декомпозувати на чотири послідовні фази, що утворюють замкнений контур керування:

Спостереження (Observe): на цьому етапі агент діє як сенсорна система, здійснюючи моніторинг стану навколишнього середовища. У контексті симуляції це передбачає збір даних через віртуальні органи чуття (зір, відчуття дистанції) та аналіз внутрішнього гомеостазу. Результатом фази є формування вектору спостережень, що фіксує миттєвий «зріз» реальності.

Орієнтація (Orient): це критично важливий когнітивний етап, на якому "сирі" сенсорні дані трансформуються у контекстуальне розуміння ситуації. У технічній реалізації цю функцію виконують приховані шари нейронної мережі. Базуючись на попередньому досвіді (вагах мережі), агент зіставляє поточні загрози та можливості, формуючи внутрішнє представлення стану. Саме тут відбувається процес «осмислення»: чи є об'єкт попереду хижаком, чи перешкодою.

Рішення (Decide): на основі сформованої картини світу агент здійснює вибір оптимальної дії з доступного простору можливостей. У термінах навчання з підкріпленням це відповідає семплюванню дії з розподілу політики (Policy). Цей вибір є стратегічним і спрямований на максимізацію очікуваної винагороди, наприклад, обрання траєкторії втечі або атаки.

Дія (Act): фінальна фаза, де прийняте рішення реалізується у фізичному просторі. Агент змінює свій стан (переміщується, прискорюється), що неминуче впливає на конфігурацію середовища. Наслідки цієї дії генерують новий потік сенсорних даних, запускаючи наступну ітерацію циклу.

Ключовим елементом, що забезпечує еволюцію поведінки в цьому циклі, є механізм зворотного зв'язку. Середовище безперервно «оцінює» результати дій агента через сигнал винагороди. Ця оцінка не просто сигналізує про успіх чи невдачу, а слугує сигналом помилки для алгоритму оптимізації, який калібрує внутрішню модель (етап Орієнтації). Таким чином, мільйони ітерацій OODA-циклу дозволяють агенту перейти від хаотичних реакцій до стійкої адаптивної поведінки.

З математичної точки зору, задача адаптивної поведінки в RL формалізується як марковський процес прийняття рішень (Markov Decision Process, MDP) [16]. На відміну від жорстких правил, агент в MDP не має інструкцій «що робити». Натомість він має мету – максимізацію кумулятивної функції винагороди *R*. Процес навчання полягає у пошуку оптимальної політики *pi(s)*, яка відображає стан середовища *S* на ймовірнісний розподіл дій *A*.

Фундаментальна перевага RL над FSM та BT полягає у здатності до узагальнення (generalization). Використання глибоких нейронних мереж як апроксиматорів функції політики дозволяє агенту діяти в станах, яких він ніколи раніше не зустрічав. Наприклад, навчившись уникати хижака в лісі, RL-агент може застосувати схожу стратегію в горах, адаптуючи її до нового ландшафту. Це явище, відоме як трансферне навчання (Transfer Learning), є критично важливим для моделювання біологічної еволюції, де організми постійно стикаються зі змінним середовищем.

Проте впровадження RL створює нові виклики, зокрема проблему «дослідження проти експлуатації». Агент повинен балансувати між використанням відомих ефективних стратегій (експлуатація) та спробами нових, потенційно небезпечних дій заради отримання нової інформації. У контексті екосистеми це може призводити до тимчасового зниження виживаності популяції на етапах навчання, що вимагає розробки спеціалізованих механізмів безпеки та навчальних планів (Curriculum Learning).

Окремої уваги заслуговує порівняльний аналіз парадигми навчання з підкріпленням (RL) та імітаційного навчання (Imitation Learning / Behavioral Cloning). Імітаційне навчання базується на наявності експертних даних – набору записаних траєкторій «ідеальної» поведінки, які нейромережа намагається скопіювати. Хоча цей метод забезпечує швидшу збіжність, він має критичний недолік у контексті даної роботи: відсутність «вчителя». Ми не можемо надати нейромережі дані про те, як поводиться «ідеальний вовк», оскільки мета симуляції – саме виявити ці стратегії еволюційним шляхом.

Цей метод базується на концепції навчання з вчителем, де агент намагається апроксимувати функцію політики, мінімізуючи розбіжність між власними діями та діями, зафіксованими у наборі експертних демонстрацій. Узагальнена схема алгоритму імітаційного навчання наведена на рисунку 1.2 [17].

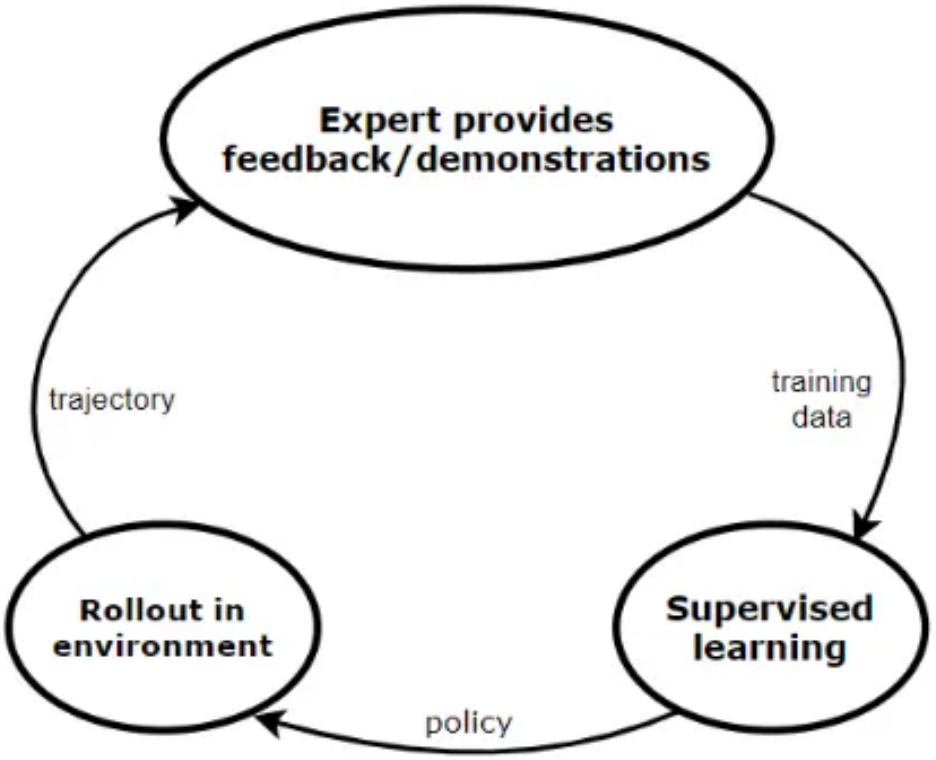


Рис. 1.2 Схема алгоритму імітаційного навчання

Технічна реалізація даного підходу вимагає створення масиву еталонних траєкторій – записів «ідеальних» сценаріїв поведінки, створених людиною-оператором або евристичним алгоритмом. Головною перевагою IL є висока швидкість збіжності нейронної мережі, оскільки агент уникає етапу стохастичного дослідження середовища, одразу отримуючи коректні вектори дій для заданих станів.

Проте в контексті моделювання еволюційної екосистеми даний метод має низку критичних обмежень:

1. Обмеженість компетентністю експерта: агент, навчений методом імітації, принципово не здатен перевершити ефективність наданих йому

демонстрацій. Він лише реплікує відомі стратегії, що унеможливлює появу емерджентної поведінки або відкриття нових тактик виживання.

2. Низька робастність: імітаційні моделі демонструють слабку здатність до узагальнення у станах, які не були представлені в навчальному наборі. Будь-яка непередбачена зміна ландшафту або нестандартна дія хижака може призвести до каскадних помилок у поведінці жертви, оскільки в її нейронній мережі відсутні механізми адаптації до невідомого.

Саме через ці недоліки імітаційне навчання було відхилено на користь навчання з підкріпленням, яке дозволяє агентам навчатися на власному досвіді tabula rasa. На відміну від імітації, RL дозволяє навчатися Tabula Rasa (з чистого аркуша), генеруючи унікальні, часто неочікувані стратегії, що перевершують людське розуміння оптимальності. Саме тому для задач дослідження емерджентної поведінки в екосистемах RL є безальтернативним вибором.

Підсумовуючи аналіз методів, можна стверджувати, що для створення симуляції, яка претендує на наукову новизну та реалізм, використання статичних евристик є недостатнім. Лише застосування методів глибокого навчання з підкріпленням дозволяє наблизитися до відтворення природних когнітивних процесів тварин, забезпечуючи виникнення справжньої, не запрограмованої заздалегідь емерджентної поведінки. Саме тому в даній роботі за основу обрано гібридний підхід, де фізична взаємодія реалізується детермінованим рушієм, а прийняття рішень делегується нейронним мережам.

1.4 Обґрунтування вибору методу навчання та порівняльний аналіз алгоритмів

Вибір алгоритму оптимізації політики є фундаментальним етапом проектування інтелектуальної системи, оскільки він визначає не лише швидкість навчання, але й саму можливість досягнення стабільної поведінки агентів у стохастичному середовищі. Задача симуляції природної екосистеми накладає специфічні обмеження, які звужують коло потенційних рішень. До таких обмежень

належать необхідність роботи у безперервному просторі станів та дій, висока розмірність вхідних даних, що надходять від векторів променевого сканування, стохастичність середовища та потреба в навчанні «on-policy» для забезпечення кореляції між діями та їх наслідками в реальному часі. У сучасному ландшафті глибокого навчання з підкріпленням виділяють три домінуючі сімейства алгоритмів, які потенційно могли бути застосовані в даній роботі: методи на основі цінності, зокрема Deep Q-Networks (DQN), методи на основі градієнта політики та гібридні методи актор-критик, до яких належать Soft Actor-Critic (SAC) та Proximal Policy Optimization (PPO). Для обґрунтованого вибору необхідно провести детальний порівняльний аналіз цих підходів.

Алгоритм Deep Q-Network (DQN) став проривним методом, що поєднав навчання з підкріпленням із глибокими нейронними мережами [18]. Його математична основа базується на Q-навчанні, тобто ітеративному наближенні функції цінності дії, яка оцінює очікувану сумарну винагороду за виконання певної дії у заданому стані. Перевагою DQN є його висока ефективність у середовищах з дискретним простором дій, наприклад, у класичних іграх, де агент має фіксований набір кнопок керування. Використання буфера відтворення досвіду дозволяє алгоритму навчатися на некорельованих зразках даних, що підвищує загальну стабільність процесу. Однак фундаментальним обмеженням DQN для задач фізичної симуляції є його орієнтація саме на дискретні дії. У розробленій екосистемі агент керується фізичними силами, такими як імпульс руху та крутний момент, які є безперервними величинами. Для застосування DQN довелося б виконувати дискретизацію простору дій, наприклад, замість плавного повороту дозволяти агенту повертатися лише на фіксовані кути. Це призводить до низки проблем, зокрема до втрати точності керування, через що рухи агентів стають неприродними та роботизованими, що суперечить меті створення реалістичної біологічної моделі. Крім того, виникає проблема прокляття розмірності, оскільки зі збільшенням роздільної здатності дискретизації кількість вихідних нейронів мережі зростає експоненційно, що значно ускладнює навчання. Через ці фактори

використання методів сімейства Q-Learning було відхилено як невідповідне фізичній природі симуляції.

Алгоритм Soft Actor-Critic (SAC) є представником сучасних методів «off-policy», що базуються на архітектурі Актор-Критик [19]. Ключовою особливістю SAC є модифікована цільова функція, яка максимізує не лише очікувану винагороду, але й ентропію політики. Це забезпечує ефективність дослідження середовища, оскільки завдяки максимізації ентропії агент схильний випробовувати нові дії і не застрягає у локальних оптимумах детермінованих стратегій. Крім того, SAC нативно підтримує безперервний простір дій, що робить його теоретично придатним для фізичних симуляцій. Проте, незважаючи на високу теоретичну ефективність, SAC має суттєві недоліки з точки зору інженерної реалізації в середовищі Unity ML-Agents. Насамперед, це висока обчислювальна складність, оскільки алгоритм вимагає одночасного навчання щонайменше п'яти нейронних мереж: двох Q-критиків, двох цільових мереж критиків та однієї мережі актора. Це створює надмірне навантаження на графічний процесор, особливо при навчанні популяції з багатьох агентів. Іншим недоліком є значна чутливість до гіперпараметрів, зокрема до температурного коефіцієнта, неправильний вибір якого може призвести до повної розбіжності алгоритму. В умовах мультиагентного середовища, де динаміка взаємодій постійно змінюється, підтримка стабільності SAC є надзвичайно складним завданням, що обумовило відмову від його використання в даній роботі.

Оптимальним вибором для реалізації поставлених задач було визначено алгоритм Proximal Policy Optimization (PPO), розроблений дослідниками OpenAI [20], який належить до класу методів «on-policy». PPO вирішує проблему нестабільності навчання, характерну для класичних методів градієнта політики, шляхом введення сурогатної цільової функції з механізмом обмеження. Цей механізм гарантує, що оновлена політика не буде відхилятися від попередньої більше ніж на задану величину, що забезпечує монотонне покращення стратегії агента і робить процес навчання передбачуваним. Ключовим аргументом на користь вибору PPO є його стабільність у довготривалих симуляціях, які

характеризуються тривалими епізодами та високою дисперсією винагород. PPO демонструє високу робастність до зашумлених даних, що дозволяє агентам вчитися навіть у ситуаціях, коли винагорода є розрідженою, наприклад, у випадках успішного полювання хижака після тривалого переслідування.

Окрім стабільності, важливим фактором є ефективна робота PPO з безперервним простором дій. Алгоритм природним чином працює з виходами нейромережі, що представляють параметри нормального розподілу для кожної дії, дозволяючи генерувати плавні та реалістичні рухи тварин, що імітують м'язову активність. Також PPO ідеально підходить для архітектури паралельного навчання, реалізованої в Unity ML-Agents, оскільки дозволяє збирати траєкторії від десятків копій агентів одночасно, об'єднувати їх у великі пакети та виконувати усереднене оновлення градієнта. Це суттєво прискорює збіжність алгоритму порівняно з методами, що навчаються на одному потоці. Додатковою перевагою є простота налаштування, оскільки, на відміну від SAC, PPO має меншу кількість критичних гіперпараметрів і демонструє гарні результати з базовими налаштуваннями для широкого спектра задач.

Узагальнюючи порівняльний аналіз, можна стверджувати, що алгоритм Deep Q-Network є непридатним через неможливість адекватного керування фізикою у безперервному просторі без втрати точності. Алгоритм Soft Actor-Critic, хоч і підтримує безперервні дії, був відхилений через надмірну обчислювальну складність та ризики нестабільності в мультиагентному середовищі. Натомість архітектура Proximal Policy Optimization забезпечує необхідний баланс між якістю керування, стабільністю процесу навчання та технічною можливістю масштабування симуляції. Саме PPO дозволяє реалізувати надійний механізм адаптації агентів, здатний до формування стійких поведінкових патернів в умовах еволюційного тиску, що робить його безальтернативним вибором для даної магістерської роботи.

1.5 Аналіз модельної екосистеми та обґрунтування вибору біологічних видів

Для забезпечення валідності симуляції та її наукової цінності було проведено детальний аналіз фауни Волинської області. Екосистема не є простою сумою організмів, а становить складну мережу функціональних зв'язків. Тому процес селекції видів базувався на системі чітко визначених критеріїв: екологічна значимість (типовість для регіону), чітко виражена трофічна взаємодія та наявність верифікованих наукових даних для параметризації фізичної моделі.

В якості модельного середовища обрано біом змішаних лісів Волині, що включає лісові масиви, відкриті луки та водні об'єкти. Така топологічна складність є необхідною умовою для навчання навігаційних нейронних мереж, оскільки створює природні перешкоди та зони обмеженої видимості (оклюзії).

Центральним елементом моделі обрано класичну антагоністичну пару «вовк (Canis lupus) – заєць-біляк (Lepus timidus)». Цей вибір обумовлений концепцією «еволюційної гонки озброєнь» (Red Queen Hypothesis) [21], згідно з якою хижак і жертва знаходяться у процесі постійної взаємної адаптації.

Обґрунтування вибору хижака (Canis lupus): вовк є апексним хижаком (Apex Predator) даної екосистеми, що виконує роль біологічного редуцента [22]. У контексті застосування навчання з підкріпленням, стратегія полювання даного вида не є детермінованою, а становить складний комплекс дій: курсоріальне переслідування, організація засідок та використання ландшафту. Фізіологічні параметри виду, взяті з біологічних джерел (здатність розвивати швидкість до 60 км/год на коротких дистанціях та висока витривалість), слугували базисом для налаштування кінематичної моделі агента. Така складність робить вовка ідеальним об'єктом для навчання адаптивним стратегіям, оскільки прості алгоритми тут є енергетично неефективними.

Обґрунтування вибору жертви (Lepus timidus): заєць-біляк представляє нижній рівень трофічної піраміди. Його поведінкова модель базується на стохастичному компромісі «ризик-ресурс». Агент змушений постійно вирішувати

оптимізаційну задачу: максимізувати споживання енергії при мінімізації ймовірності контакту з хижаком. Для забезпечення реалізму в модель імплементовано дані щодо пікової швидкості пересування особини (до 70 км/год), що перевищує швидкість хижака, даючи жертві шанс на втечу при своєчасному виявленні загрози [23].

З метою створення насиченої екосистеми та уникнення ефекту «лабораторної стерильності», до середовища введено фонові види: зубр (Bison bonasus) та дикий кабан (Sus scrofa). Функціонально ці великі травоїдні виступають як «ландшафтні інженери» та динамічні перешкоди.

Проектування віртуального середовища здійснювалося з урахуванням ландшафтних особливостей Волинської області, що включають зони змішаних лісів, рівнинні ділянки та водні об'єкти, які імітують місцеву гідрографічну мережу. Рослинний покрив (трав'янисті рослини, чагарники, дерева) у моделі виконує подвійну функцію: виступає джерелом енергії для травоїдних агентів та формує систему тактичних перешкод, що забезпечують можливості для маскування. Останній фактор є визначальним для формування складних просторових стратегій ухилення та переслідування.

З точки зору програмної інженерії, реалізація цих видів демонструє гібридний підхід до симуляції. На відміну від інтелектуальних агентів (вовка та зайця), що керуються нейронними мережами, фонові види керуються полегшеними скінченними автоматами (Finite State Machines). Їхня поведінка реалізована через алгоритми випадкового блукання (Random Walk). Це дозволяє суттєво зекономити обчислювальні ресурси (CPU/GPU), зберігаючи при цьому стохастичність середовища. Наявність масивних рухомих об'єктів змушує нейронні мережі основних агентів вчитися розрізняти цілі (friend-or-foe identification) та будувати маршрути обходу, що підвищує робастність моделі.

Висновки до розділу 1

У першому розділі магістерської роботи проведено комплексний аналіз теоретичних та технологічних аспектів моделювання складних адаптивних систем, що дозволило сформулювати методологічний базис дослідження.

За результатами порівняльного аналізу основних парадигм моделювання встановлено, що агентно-орієнтований підхід (Agent-Based Modeling) є найбільш доцільним інструментом для дослідження емерджентних властивостей екосистем. На відміну від аналітичних популяційних моделей, даний метод дозволяє врахувати гетерогенність агентів та стохастичність їхніх локальних взаємодій. Таким чином, було виконано першу частину завдання роботи: проаналізовано існуючі методи моделювання мультиагентних систем та обґрунтовано переваги агентного підходу.

У ході дослідження проблематики адаптивної поведінки виявлено критичні обмеження детермінованих алгоритмів у динамічних середовищах та обґрунтовано перехід до парадигми навчання з підкріпленням. Вибір методу Proximal Policy Optimization обумовлений стабільністю його цільової функції та ефективністю використання даних, що є критично важливим для формування складних поведінкових патернів. Також визначено необхідність застосування методики навчання за планом для подолання проблеми розріджених винагород. Цим завершено виконання першого завдання: виявлено фактори, що впливають на збіжність алгоритмів навчання з підкріпленням.

Для забезпечення наукової валідності симуляції сформовано структуру модельної екосистеми на прикладі фауни Волинської області. Визначено антагоністичну пару «вовк-заєць» як ядро симуляції, що базується на гіпотезі «Чорної Королеви», та обґрунтовано введення фонових видів для створення динамічної стохастичності. Сформовано набір біологічних обмежень (швидкість, метаболізм, сенсорний діапазон), які необхідно закласти в математичні моделі агентів.

Узагальнюючи проведене дослідження, можна стверджувати, що у першому розділі повністю виконано аналітичну частину завдань роботи. Результати аналізу сформували методологічну основу для переходу до наступного етапу – математичної формалізації методу та його програмної реалізації у другому розділі.

2 РОЗРОБКА МЕТОДУ НАВЧАННЯ ЗА ПЛАНОМ

2.1 Математична модель середовища як марковський процес прийняття рішень

Формалізація задачі моделювання адаптивної поведінки агентів у складній екосистемі вимагає переходу від описових біологічних концепцій до суворих математичних абстракцій. Базовим апаратом для опису стохастичної взаємодії суб'єкта із середовищем обрано теорію марковських процесів прийняття рішень (Markov Decision Process, MDP).

Враховуючи той факт, що біологічний агент не володіє повною інформацією про глобальний стан системи (наприклад, вовк не знає точного розташування всіх зайців у лісі), задачу коректніше класифікувати як частково спостережуваний марковський процес (Partially Observable MDP, POMDP) [24]. Проте, для спрощення формалізації без втрати загальності, ми будемо оперувати поняттями класичного MDP, розглядаючи стан середовища як проекцію локальних спостережень агента.

Формально модель середовища D (Domain) визначається впорядкованим кортежем з п'яти елементів (2.1) [25]:

$$D = \langle S, A, R, P, \gamma \rangle, \tag{2.1}$$

де, S – множина станів (позиція агента, рівень енергії, наявність хижака тощо);

A – множина дій; $P(s'|s,a)$ – ймовірність переходу;

$R(s,a)$ – винагорода;

γ – коефіцієнт дисконту (0–1), що визначає важливість майбутніх винагород.

Зазначені компоненти кортежу формують замкнений контур зворотного зв'язку, де вихідні дані середовища стають вхідними даними для агента, і навпаки. Графічна інтерпретація циклічного обміну сигналами (станами, діями та

винагородами) між агентом та стохастичним середовищем у рамках формалізму MDP наведена на рисунку 2.1 [26].

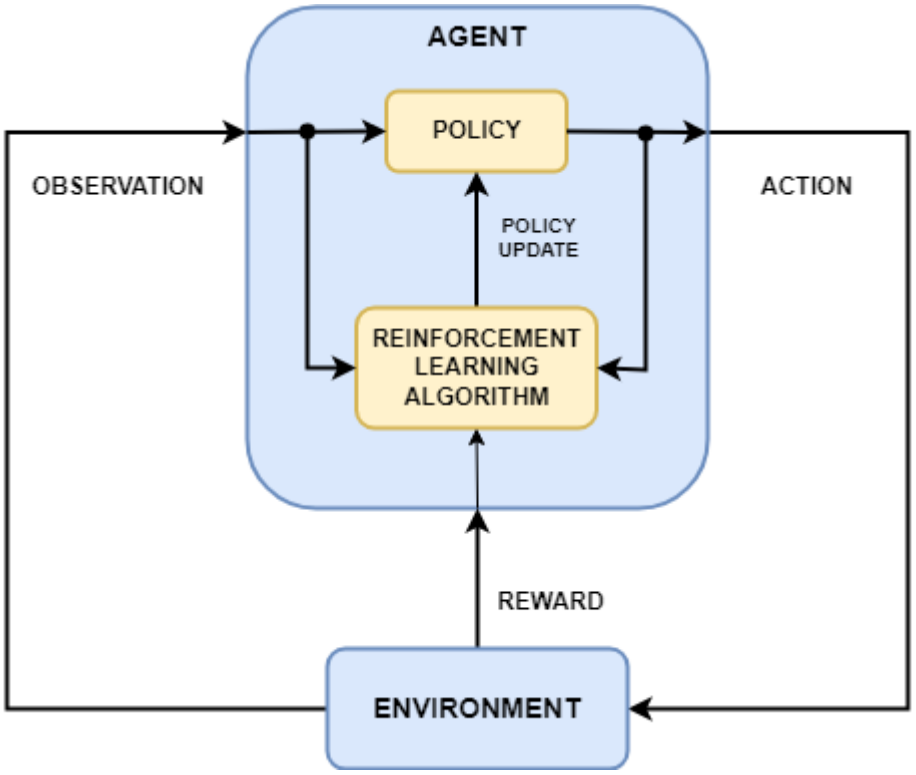


Рис. 2.1 Схема взаємодії агента із середовищем у моделі MDP

Нижче наведено детальну математичну специфікацію кожного компонента моделі.

Формалізація простору станів *S*. Простір станів *S* у розробленій моделі є безперервним (Continuous) і багатовимірним. Стан агента в дискретний момент часу *t*, позначений як $s_t \in S$, є конкатенацією двох векторів: вектора екстероцептивного сприйняття (зовнішні сенсори) та вектора інтероцептивного стану (внутрішні параметри організму) (2.2):

$$s_t = concat(O_{vision}, I_{stats}). \quad (2.2)$$

Вектор сенсорного сприйняття O_{vision}. Для моделювання зорової системи використано підхід променевого сканування (Raycasting). Агент випускає набір з *N* променів у горизонтальній площині в секторі огляду Ф градусів. Кожен промінь

r_i формує частину вектора спостережень на основі перетину з об'єктами середовища.

Математично результат *i*-го променя описується підвектором v_i (2.3):

$$v_i = [d_i,\ c_i]^T, \tag{2.3}$$

де $d_i \in [0,\ 1]$ – нормалізована відстань до перешкоди, де 0 – впритул, 1 – перешкода відсутня (максимальна дальність бачення L_{max}), функція нормалізації: $d_i = \frac{D_{actual}}{L_{max}}$; c_i – вектор категоріального кодування типу об'єкта (One-Hot Encoding). Якщо в симуляції існує *K* типів об'єктів (наприклад: стіна, їжа, хижак, партнер, пустий простір), то $c_i \in \{0,\ 1\}^K$, де $\sum_{j=1}^{K} c_{ij} = 1$.

Таким чином, повний вектор візуального сприйняття має розмірність *N* x (1 + *K*).

Вектор внутрішнього стану I_{stats} (2.4). Внутрішній гомеостаз агента описується вектором $I_{stats} \in R^3$, компоненти якого моделюють базові біологічні потреби:

$$I_{stats} = [e_t,\ h_t,\ w_t], \tag{2.4}$$

де $e_t \in [0,\ 1]$ – рівень енергії (ситості), $h_t \in [0,\ 1]$ – рівень фізичного здоров'я, $w_t \in [0,\ 1]$ – рівень гідратації.

Формалізація простору дій *A*. На відміну від спрощених сіткових моделей, де агент має дискретний набір дій (вгору, вниз, вліво, вправо), у даній роботі використовується безперервний простір дій (Continuous Action Space) [27]. Це дозволяє реалізувати плавну, реалістичну кінематику руху.

Вектор дії a_t (2.5), що генерується політикою агента, визначається як:

$$a_t = [\upsilon_x,\ \upsilon_z,\ \omega] \in [-1,\ 1]^3, \tag{2.5}$$

де υ_x – імпульс руху вздовж локальної осі X (бічний рух, стрейф), υ_z – імпульс руху вздовж локальної осі Z (рух вперед/назад), ω – кутова швидкість обертання навколо вертикальної осі Y.

Результуюча швидкість агента (2.6) V_{agent} у глобальній системі координат розраховується з урахуванням матриці повороту $R(\theta_t)$ та обмеження на максимальну швидкість V_{max}:

$$V_{actual} = clamp(V_{agent}, 0, V_{max}). \quad (2.6)$$

Така модель дозволяє нейронній мережі тонко керувати швидкістю, наприклад, сповільнюватися для маневрування або прискорюватися для атаки, що є критичним для задач переслідування.

Функція переходу *P*. Функція переходу $P(s_{t+1}|s_t, a_t)$ визначає ймовірність переходу системи в новий стан s_{t+1} (2.7) при виконанні дії a_t у стані s_t. У контексті фізичної симуляції ця функція не задається явною ймовірнісною таблицею, а визначається законами механіки Ньютона, інтегрованими за часом Δt.

$$s_{t+1} = F_{physics}(s_t, a_t) + \xi, \quad (2.7)$$

де $F_{physics}$ – детермінована функція фізичного рушія (розрахунок нових координат, швидкостей, колізій), ξ – стохастичний шум, внесений зовнішніми факторами (поведінкою інших агентів, випадковим спавном ресурсів).

Саме наявність доданку ξ робить середовище недетермінованим і вимагає від агента здатності до адаптації, а не просто запам'ятовування маршруту.

Функція винагороди *R*. Функція винагороди $R(s_t, a_t, s_{t+1})$ є ключовим елементом дизайну середовища, оскільки саме вона визначає цільову поведінку (Objective) агента. Проектування функції *R* базується на принципі формування винагороди [28], щоб уникнути проблеми розрідженості сигналу.

Сумарна винагорода R_{total} (2.8) на кожному кроці розраховується як зважена сума компонентів:

$$R_{total} = w_e R_{exist} + w_g R_{goal} + w_p R_{penalty} + w_v R_{velocity}, \quad (2.8)$$

де R_{exist} – винагорода за існування (мале позитивне число (наприклад, *+0.001*) за кожен такт симуляції, поки агент живий. Це стимулює пріоритет виживання над ризикованими діями; R_{goal} – винагорода за ціль, дискретна винагорода за події (поїдання їжі : *+1.0*, спіймана здобич : *+1.0*); $R_{penalty}$ – штрафи (смерть від голоду/хижака: *-0.1*, зіткнення зі стіною: *-0.1*.); $R_{velocity}$ – регуляризація швидкості.

Запропонована математична модель MDP (що відповідає компоненту *D* у загальній формулі методу) створює необхідний формальний базис для застосування алгоритмів глибокого навчання з підкріпленням.

2.2 Алгоритм оптимізації стохастичної політики агентів

У попередньому підрозділі задачу виживання агента було формалізовано як пошук оптимальної стратегії (політики) π^* у середовищі MDP. Математично політика визначається як імовірнісний розподіл над простором дій *A* за умови перебування у стані s: $\pi(a|s)$.

Оскільки простір станів *S* є високорозмірним і безперервним, табличне представлення політики є неможливим. Тому використовується параметризована політика $\pi_\theta(a|s)$, де θ – вектор вагових коефіцієнтів глибокої нейронної мережі. Задача зводиться до знаходження такого вектора θ, який максимізує очікувану кумулятивну винагороду $J(\pi_\theta)$ (2.9):

$$J(\pi_\theta) = E_{r\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]. \quad (2.9)$$

Для вирішення цієї оптимізаційної задачі у роботі обрано алгоритм Proximal Policy Optimization (PPO) [29]. Цей метод належить до класу Policy Gradient алгоритмів, але, на відміну від класичних підходів (наприклад, REINFORCE), він

гарантує монотонне покращення політики завдяки обмеженню величини кроку оновлення ваг.

Алгоритм PPO реалізується в рамках архітектури «Актор-Критик» [30], яка використовує дві паралельні або частково об'єднані нейронні мережі:

1. Актор (Actor, π_θ): апроксимує політику. На вхід отримує стан s_t, на виході генерує параметри розподілу дій (для безперервного простору – це вектор середніх значень μ та стандартних відхилень σ нормального розподілу $N(\mu, \sigma)$.

2. Критик (Critic, V_ϕ (2.10)): апроксимує функцію цінності стану $V(s)$. Вона оцінює, наскільки «вигідним» є перебування агента в даному стані з точки зору майбутніх винагород.

$$V_\phi(s) = E[R_t + \gamma R_{t+1} + \ldots | s_t = s]. \quad (2.10)$$

Для зменшення дисперсії градієнтів використовується функція переваги (Advantage Function) $A(s, a)$ (2.11). Вона показує, наскільки дія a краща за середню дію в стані s. У даній роботі застосовано метод узагальненої оцінки переваги (GAE – Generalized Advantage Estimation) [31], який дозволяє балансувати зміщення та дисперсію за допомогою параметра λ:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V], \quad (2.10)$$

де δ_t^V – помилка часової різниці (TD-error), що обчислюється з використанням мережі Критика.

Якщо $\lambda = 0$, оцінка базується лише на наступному кроці (низька дисперсія, високе зміщення). Якщо $\lambda = 1$, враховується вся траєкторія (висока дисперсія, точна оцінка). В експериментах обрано компромісне значення $\lambda = 0.95$.

Ключовою інновацією PPO, яка забезпечує стабільність навчання, є модифікована функція втрат. У стандартних методах градієнтного спуску занадто великий крок оновлення може зруйнувати політику («Policy Collapse») [32]. PPO вводить обмеження на те, наскільки нова політика π_θ може відрізнятися від старої $\pi_{\theta\ old}$.

Введемо відношення ймовірностей (probability ratio) $r_t(\theta)$ (2.11):

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\ old}(a_t|s_t)}. \quad (2.11)$$

Цільова функція PPO L^{CLIP} (2.12) визначається як мінімум між звичайним градієнтом та градієнтом з обмеженням (clipping):

$$L^{CLIP}(\theta) = \hat{E}[\ min(r(\theta), clip(r(\theta), 1-\epsilon, 1+\epsilon))], \quad (2.12)$$

де *clip(x, min, max)* – функція обмеження значення в діапазоні, ϵ – гіперпараметр (зазвичай *0.2*), що визначає «довірчий регіон». Це гарантує, що ймовірність вибору дії зміниться не більше ніж на *20* за одну ітерацію.

Загальна функція втрат L^{PPO} (2.13), яку мінімізує нейронна мережа в процесі навчання, включає три компоненти: втрати політики (CLIP), помилку оцінки цінності та бонус за ентропію.

$$L_t^{PPO}(\theta) = -L_t^{CLIP}(\theta) + c_1 L_t^{VF}(\theta) - c_2 S[\pi_\theta](s_t), \quad (2.11)$$

де $L_t^{VF} = (V_\phi(s_t) - V_{target})^2$ – квадратична помилка Критика. Мінімізація цього доданку навчає мережу точніше передбачати очікувану винагороду. Коефіцієнт c_1 (зазвичай *0.5*) балансує внесок Критика; $S[\pi_\theta]$ – ентропія розподілу політики. Максимізація ентропії (знак «мінус» перед c_2) запобігає передчасній збіжності до локальних оптимумів, стимулюючи агента досліджувати середовище (Exploration). Коефіцієнт c_2 (зазвичай *0.01*) поступово зменшується в процесі навчання.

Таким чином, математичний апарат PPO дозволяє вирішити задачу оптимізації у багатовимірному просторі дій, забезпечуючи стабільність процесу навчання навіть в умовах стохастичного середовища екосистеми.

2.3 Формалізація методу поетапного навчання

Класичні методи навчання з підкріпленням, такі як PPO, демонструють високу ефективність у середовищах з густим сигналом винагороди. Однак у реалістичній екосистемній симуляції агент стикається з проблемою «розріджених винагород» [33]. Ймовірність того, що агент з ініціалізованими випадковим чином вагами нейронної мережі зможе вижити достатньо довго, щоб знайти їжу та отримати позитивний сигнал підкріплення, наближається до нуля. Це призводить до того, що градієнт функції втрат стає неінформативним, а політика агента конвергує до локального мінімуму – стратегії повної пасивності (для мінімізації штрафів за рух).

Для вирішення цієї проблеми у роботі розроблено комплексний метод симуляції, який базується на концепції навчання за планом або поетапного навчання (Curriculum Learning) [34]. Суть методу полягає в декомпозиції складної глобальної задачі виживання на послідовність спрощених підзадач зі зростаючою складністю.

Формально запропонований метод M можна представити як композицію параметризованого середовища, алгоритму оптимізації та функції керування складністю, що діють в умовах жорстких обмежень. Узагальнена формула методу має вигляд (2.12):

$$M = \frac{\langle D, P, T \rangle}{C}, \tag{2.12}$$

де D (Domain) – простір задачі (MDP); P (Policy Optimization) – алгоритм пошуку стратегії (PPO); T (Training Curriculum) – функція навчального плану, що визначає динаміку зміни параметрів середовища у часі t; C (Constraints) – множина фізичних та біологічних обмежень, що звужують простір пошуку рішень.

Формалізація навчального плану T. Навчальний план T визначається як впорядкована множина дискретних уроків (Lessons) (2.13):

$$T = \{L_0, L_1, \ldots, L_K\}. \tag{2.13}$$

Кожен урок L_k являє собою модифікований марковський процес MDP_k (2.14), який є підмножиною повного середовища:

$$MDP_k = \langle S,\ A,\ P,\ P_k,\ \gamma,\ \Omega_k \rangle, \tag{2.14}$$

де Ω_k – набір конфігураційних параметрів середовища для k-го етапу.

Перехід між етапами $L_k \to L_{k+1}$ здійснюється автоматично при виконанні умови успішності. Нехай $\check{R}_t$ – ковзне середнє кумулятивної винагороди за останні 1000 кроків.

Початковий етап навчання L_0 («базова навігація») спрямований на вирішення фундаментальної задачі асоціації візуальних стимулів, зокрема ідентифікації ресурсів (зелений колір), із вектором руху в їхньому напрямку. Для прискорення збіжності параметри середовища Ω_0 спрощуються до мінімуму: розмір мапи обмежується локальним сектором $Area_{small} = 20 \times 20$ м, а фактор хижацтва повністю виключається ($N_{pred} = 0$). Ключовою особливістю цього етапу є модифікація функції винагороди R_0 шляхом додавання евристичного компонента $R_0 = \frac{1}{dist(agent, food)}$. Це забезпечує агента миттєвим щільним зворотним зв'язком [35], дозволяючи градієнтному спуску ефективно оптимізувати траєкторію руху ще до першого успішного досягнення цілі.

Наступний етап L_1 («ухилення від загроз») ускладнює задачу шляхом введення антагоністичного фактору. Метою етапу є формування стійкого рефлексу втечі при детекції загрози у полі зору. Параметри середовища Ω_1 передбачають появу одного хижака ($N_{pred} = 1$) зі спрощеною детермінованою логікою переслідування. При цьому структура винагороди зазнає змін: допоміжна евристика R_{dist} відключається, що переводить систему в режим розріджених винагород, та вводиться критичний штраф за смерть агента $R_{death} = -1.0$. Це

змушує нейронну мережу переоцінити пріоритети, ставлячи безпеку вище за споживання ресурсів.

Фінальний етап L_2 («конкурентне середовище») відтворює цільові умови експлуатації системи, що характеризуються високим рівнем стохастичності та конкуренції. Простір симуляції розширюється до повних розмірів $Area_{full} = 100 \times 100$ м, активуються динамічні перешкоди (зубри, кабани) та вводяться конкуруючі агенти ($N_{pred} > 1$). Запропонована архітектура навчального плану дозволяє сформувати початкові ваги нейронної мережі θ_{init} на етапі L_0 таким чином, що вони опиняються в «басейні тяжіння» глобального оптимуму. Це математично гарантує збіжність алгоритму при переході до складніших задач на етапах L_1 та L_2, які були б нерозв'язними при навчанні «з нуля».

Формалізація системи обмежень *C*. Знаменник у формулі методу *C* представляє собою систему обмежень, які "відсікають" невалідні або біологічно неможливі стани, тим самим зменшуючи розмірність простору пошуку для алгоритму PPO.

Система обмежень поділяється на дві групи: $C = C_{phys} \cup C_{bio}$.

1. Фізичні обмеження C_{phys}: визначаються топологією простору та законами механіки.

- Non-penetration constraint: агент не може перебувати в координатах, зайнятих статичними об'єктами.
- Boundaries constraint: агент обмежений периметром симуляції.

2. Біологічні обмеження C_{bio}: визначають межі можливостей організму, базуючись на даних з Розділу 1.

- Kinematic Limit: максимальна швидкість $v(t)$ залежить від поточного рівня енергії $e(t)$.
- Metabolic Constraint: будь-яка дія a_t має енергетичну вартість $Cost(a_t)$.

Це обмеження змушує агента оптимізувати маршрути, уникаючи зайвих рухів.

На основі формалізованих компонентів (середовища *D*, алгоритму *P* та плану *T*) синтезовано загальний алгоритм функціонування методу *M*, який являє собою систему вкладених циклів. Практична реалізація методу розпочинається з фази ініціалізації, де встановлюється глобальний лічильник часу, обирається початковий рівень навчального плану L_0 з відповідними параметрами середовища Ω_0, а ваги нейронних мереж політики та критика ініціалізуються випадковими значеннями для забезпечення стохастичності пошуку на старті.

Основний обчислювальний процес відбувається у внутрішньому циклі взаємодії та збору даних. На кожній ітерації система генерує екземпляр середовища з урахуванням діючих фізичних обмежень, де агент взаємодіє зі світом протягом фіксованого горизонту часу. Отримані траєкторії руху, що складаються з кортежів станів, дій та винагород, накопичуються у буфері досвіду.

Після його заповнення активується фаза оптимізації, під час якої розраховуються оцінки переваг (GAE) та виконується серія оновлень ваг нейронної мережі за алгоритмом PPO з метою максимізації цільової функції.

Зовнішній контур керування відповідає за адаптивну еволюцію складності задачі. Після кожного блоку навчання система оцінює ефективність поточної політики шляхом розрахунку середньої кумулятивної винагороди Ř. Якщо отримане значення перевищує пороговий показник поточного уроку τ_k, відбувається перехід на наступний рівень навчального плану, що супроводжується інкрементацією індексу уроку та оновленням параметрів середовища $\Omega_k \to \Omega_{k+1}$ (додавання нових загроз або розширення території).

Процес триває до моменту стабілізації метрик на фінальному рівні складності, що свідчить про успішну конвергенцію методу від хаотичної поведінки до стійкої стратегії виживання. Цей формалізований підхід дозволяє перейти від хаотичного навчання до керованого еволюційного процесу, що забезпечує стабільність результатів.

2.4 Критерії оцінювання ефективності

Для всебічної валідації розробленого методу недостатньо покладатися виключно на показник середньої винагороди, оскільки він може приховувати локальні оптимуми (наприклад, стратегію пасивного виживання). Тому для оцінки ефективності вводиться розширений вектор метрик E, що включає показники швидкості збіжності, стабільності та результативності виконання цільових задач.

Першим критично важливим критерієм є ефективність використання даних (Sample Efficiency, S_{eff}) [36]. Цей показник визначає кількість кроків симуляції (Time Steps), необхідних алгоритму для досягнення заданого порогу продуктивності τ. Математично момент збіжності T_{conv} (2.15) визначається як перший момент часу *t*, коли ковзне середнє винагороди перевищує поріг:

$$T_{conv} = min\{t|\check{R}_t > \tau\}. \quad (2.15)$$

Порівняння значень T_{conv} для запропонованого методу та класичного навчання дозволяє кількісно довести гіпотезу про те, що Curriculum Learning прискорює процес навчання та потребує менше обчислювальних ресурсів (GPU-годин) для формування стабільної політики.

Другим показником є коефіцієнт успішності (Success Rate, *SR*) (2.16). На відміну від скалярної винагороди, яка включає допоміжні сигнали, ця метрика є бінарною і відображає відсоток епізодів, що завершилися досягненням термінальної цілі (для хижака – «здобич спіймано», для жертви – «успішна втеча» або «виживання протягом T_{limit}»).

$$SR = \frac{1}{K}\sum_{k=1}^{K} I(outcome_k = success), \quad (2.15)$$

де I – індикаторна функція. Високий показник середньої винагороди при низькому SR свідчитиме про «злам винагороди», тоді як високий SR підтверджує валідність навченої поведінки.

Додатково оцінюється стабільність політики, яка виражається через дисперсію кумулятивної винагороди σ_R^2. Висока дисперсія на пізніх етапах навчання свідчить про нестабільність нейронної мережі ("Catastrophic Forgetting") [37]. Зниження ентропії політики (Policy Entropy, *H*) у поєднанні зі зростанням *SR* та зменшенням σ_R^2 є комплексним індикатором того, що агент сформував стійку, детерміновану стратегію, яка є робастною до стохастичних збурень середовища.

Коефіцієнт виживання популяції S_R використовується як макро-метрика для оцінки екологічної рівноваги. Він розраховується як відношення кількості активних агентів до початкового розміру популяції в динаміці. Це дозволяє перевірити, чи здатен розроблений метод забезпечити довготривале існування віртуальної екосистеми без виродження одного з видів.

Висновки до розділу 2

У другому розділі магістерської роботи вирішено задачу математичної формалізації процесу симуляції екосистеми та проектування алгоритмічного забезпечення для навчання адаптивних агентів. Отримані результати дозволяють стверджувати, що теоретична розробка методу завершена, а поставлені наукові задачі вирішено.

Щодо розробки математичних моделей, у роботі було формалізовано середовище симуляції як частково спостережуваний марковський процес прийняття рішень (POMDP). Визначено структуру векторного простору станів, який включає дані променевого сканування та внутрішні показники гомеостазу, а також безперервного простору дій, що описує кінематику руху агентів. Також спроектовано композитну функцію винагороди, яка враховує баланс між виживанням та виконанням цільових дій. Таким чином, було виконано третє завдання роботи в частині теоретичного моделювання: розроблено математичні моделі агентів, визначено структуру їх сенсорних систем та простору дій.

У ході роботи над алгоритмічним забезпеченням обґрунтовано використання методу Proximal Policy Optimization для вирішення оптимізаційної задачі в безперервному просторі. Формалізовано функцію втрат із механізмом відсікання (Clipping) та впроваджено оцінку переваг GAE, що забезпечує математичну гарантію монотонного покращення політики без ризику її колапсу. Визначення цих компонентів створює необхідний базис для подальшої програмної реалізації нейронних мереж.

Для вирішення критичної проблеми розріджених винагород розроблено та формалізовано метод навчання за планом(Curriculum Learning). Метод представлено як ітераційний процес послідовного ускладнення параметрів середовища та системи обмежень. Спроектовано алгоритм автоматичного переходу між етапами «Навігація» – «Ухилення» – «Конкуренція» на основі порогових значень успішності. Це відповідає повному виконанню четвертого завдання роботи: розроблено алгоритмічне забезпечення для стабілізації навчання, що включає метод навчання за планом та систему динамічної рандомізації.

Додатково введено комплексну систему метрик ефективності, що включає коефіцієнт успішності (Success Rate) та ефективність використання даних (Sample Efficiency). Це дозволяє перейти від суб'єктивної оцінки поведінки агентів до об'єктивного вимірювання швидкості та якості їх навчання.

Узагальнюючи викладене, у другому розділі сформовано повний математичний апарат дослідження. Розроблений метод, що об'єднує модель середовища, алгоритм оптимізації та навчальний план, готовий до програмної імплементації та експериментальної перевірки, які будуть проведені у наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Аналіз інструментальних засобів програмної реалізації

Вибір технологічного стека для реалізації симуляційної моделі є критичним етапом проектування, оскільки він накладає фундаментальні обмеження на продуктивність системи, можливості масштабування кількості агентів та складність інтеграції компонентів навчання. У рамках даного дослідження було проведено порівняльний аналіз трьох основних класів програмних рішень: бібліотек машинного навчання загального призначення, спеціалізованих робототехнічних платформ та ігрових рушіїв з інтегрованими середовищами навчання.

Першим напрямом аналізу стала оцінка доцільності використання нативних засобів екосистеми Microsoft .NET, зокрема бібліотеки ML.NET. Цей фреймворк забезпечує можливість інтеграції моделей машинного навчання безпосередньо у середовище виконання C#, що теоретично дозволило б уникнути витрат на міжпроцесну комунікацію. Головною перевагою такого підходу є статична типізація та висока швидкість виконання скомпільованого коду, що є важливим для ресурсомістких обчислень.

Однак, детальний аналіз функціональних можливостей ML.NET виявив його суттєву обмеженість у контексті задач глибокого навчання з підкріпленням (Deep Reinforcement Learning) [38]. Архітектура бібліотеки оптимізована переважно для класичних задач машинного навчання, таких як регресійний аналіз, класифікація табличних даних та кластеризація. Підтримка роботи з тензорами високої розмірності та згортковими нейронними мережами (CNN), які є необхідними для обробки візуальної інформації агентів, у ML.NET значно поступається можливостям екосистеми Python (PyTorch, TensorFlow). Крім того, відсутність вбудованого фізичного рушія вимагала б розробки власної системи симуляції

кінематики та колізій «з нуля», що є нераціональним з точки зору трудовитрат та надійності кінцевого продукту.

Бібліотеки, подібні до ML.NET , демонструють високу ефективність у задачах аналізу даних, зокрема регресійного аналізу та класифікації на основі історичних датасетів. Однак, коли переноситься цей підхід у тривимірний, динамічний і постійно мінливий світ екологічної симуляції, стикаємося з фундаментальною несумісністю. Проблема полягає не в потужності алгоритмів, а в самій їх архітектурній філософії. Модель ML.NET навчається заздалегідь, на основі даних, які вже стали історією. Життя ж в екосистемі – це безперервний потік подій, що розгортається в реальному часі.

Альтернативним підходом, широко розповсюдженим в академічній робототехніці, є використання операційної системи ROS (Robot Operating System) у зв'язці з фізичним симулятором Gazebo [39]. Цей стек є стандартом де-факто для розробки систем управління автономними роботами, оскільки забезпечує надвисоку точність фізичної симуляції, включаючи моделювання інерції, тертя та характеристик реальних сенсорів (лідарів, камер глибини). Важливою перевагою є можливість прямого перенесення керуючого коду з симулятора на фізичне обладнання без необхідності адаптації.

Проте, в контексті моделювання біологічних екосистем, архітектура ROS виявляє низку критичних недоліків. По-перше, розподілена система нод (Nodes), на якій базується ROS, створює значні накладні витрати на передачу повідомлень, що ускладнює синхронізацію при навчанні у прискореному часі. По-друге, симулятор Gazebo фокусується на фізичній достовірності окремих механізмів, але погано масштабується для задач мультиагентної взаємодії (Swarm Robotics), коли на сцені одночасно діють десятки незалежних агентів. По-третє, візуальна складова Gazebo носить утилітарний характер і не дозволяє створювати складні природні ландшафти з високим рівнем деталізації, що є необхідною умовою для навчання навігації на основі зору.

Найбільш ефективним рішенням для поставлених задач виявилося використання ігрового рушія Unity у поєднанні з інструментарієм ML-Agents

Toolkit [40]. Запропонована розробниками Unity Technologies архітектура реалізує гібридний підхід, що розділяє відповідальність між двома середовищами. Unity виступає в ролі високопродуктивного сервера симуляції, забезпечуючи рендеринг сцени та обрахунок фізики (за допомогою рушія PhysX), тоді як навчання нейронних мереж відбувається у зовнішньому процесі Python, де використовуються потужності бібліотеки PyTorch.

Ключовим фактором вибору даної платформи стала наявність ефективного протоколу комунікації (Communicator API) на базі gRPC, який дозволяє передавати тензори станів та дій між C# та Python з мінімальною затримкою. Це уможливлює паралельний запуск кількох екземплярів симуляції (Environment Vectorization), що прискорює збір навчальних даних у десятки разів. Крім того, Unity надає широкі можливості для процедурної генерації ландшафтів та рандомізації умов середовища (Domain Randomization), що є критично важливим для запобігання перенавчанню агентів та забезпечення їхньої здатності до генералізації.

Окремої уваги заслуговує аспект генерації синтетичних сенсорних даних, який є визначальним для задач біологічної симуляції. На відміну від спрощених середовищ (Grid World) або утилітарної графіки Gazebo, графічний конвеєр Unity дозволяє моделювати складні візуальні перешкоди, такі як густа рослинність, динамічні тіні та нерівності рельєфу. Використання компонентів RayPerceptionSensor у такому середовищі дозволяє агентам отримувати реалістичну інформацію про оклюзію (перекриття) об'єктів, що є критично важливим для формування стратегій маскування та засідок. Це наближає умови симуляції до природних, підвищуючи екологічну валідність отриманих поведінкових патернів.

Крім того, архітектура ML-Agents забезпечує сумісність із загальноприйнятим стандартом OpenAI Gym через інтерфейс gym-unity. Це дозволяє абстрагувати логіку середовища від алгоритмів навчання, що дає можливість використовувати не лише вбудовані реалізації PPO/SAC, але й будь-які зовнішні бібліотеки глибокого навчання (наприклад, Stable Baselines3 або Ray

RLLib). Така модульність є критичною для наукової роботи, оскільки вона гарантує відтворюваність експериментів та дозволяє верифікувати ефективність авторських методик навчання на стандартизованих бенчмарках.

Таким чином, зв'язка Unity ML-Agents та PyTorch забезпечує оптимальний баланс між візуальною якістю, фізичною коректністю та обчислювальною ефективністю навчання. Порівняльна характеристика інструментальних засобів наведена в таблиці 3.1.

Таблиця 3.1

Порівняльна характеристика інструментальних засобів

Критерій порівняння	ML.NET (Native .NET)	ROS / Gazebo	Unity ML-Agents
Архітектурний підхід	Монолітний (C#)	Розподілений (C++/Python)	Гібридний (C#/Python)
Основна сфера застосування	Аналіз даних, класифікація	Індустріальна робототехніка	Симуляції, ігри, AI Research
Фізична точність	Відсутня (потребує реалізації)	Надвисока (для механізмів)	Висока (достатня для біології)
Масштабованість (Multi-Agent)	Обмежена потоками CPU	Низька (через overhead)	Висока (векторизація)
Підтримка Deep RL (PPO, SAC)	Експериментальна	Через зовнішні плагіни	Нативна (PyTorch backend)

3.2 Архітектура системи симуляції екосистеми

Реалізація методу, формалізованого у другому розділі, вимагає побудови високопродуктивної програмної системи, здатної забезпечити стабільну взаємодію двох гетерогенних обчислювальних середовищ. Для вирішення цієї задачі було спроектовано та реалізовано гібридну клієнт-серверну архітектуру, що базується на патерні Environment-Agent-Learner. Цей підхід передбачає чітке розмежування відповідальності між модулем фізичної симуляції, який генерує простір станів, та модулем когнітивної обробки даних, що відповідає за прийняття рішень та навчання нейронних мереж.

В якості ядра симуляції обрано ігровий рушій Unity 2021.3 LTS [41], який виконує роль сервера середовища (Environment Server). Фундаментальним компонентом цього рівня є фізичний рушій NVIDIA PhysX [42], що забезпечує детермінований розрахунок кінематики твердих тіл, обробку колізій та гравітаційних впливів. Саме використання PhysX дозволяє гарантувати фізичну валідність переміщення агентів, імітуючи реалістичні сили інерції та тертя, що є недосяжним для спрощених сіткових моделей. Окрім фізики, середовище забезпечує роботу графічного конвеєра, який візуалізує сцену як для оператора, так і для агентів, що використовують згорткові нейронні мережі для обробки візуального входу. Програмна логіка агентів реалізована через C# API, де скрипти здійснюють збір локальних векторних спостережень та перетворюють отримані від нейромережі абстрактні команди у конкретні фізичні сили, що прикладаються до тіла агента.

Критично важливим елементом архітектури є організація ефективного каналу передачі даних між середовищем виконання C# (Unity) та середовищем навчання Python, які функціонують у різних процесах операційної системи. Для мінімізації затримок (latency) було імплементовано механізм міжпроцесної взаємодії на базі протоколу віддаленого виклику процедур gRPC [43].

Транспортний рівень системи забезпечує серіалізацію тензорів станів та дій у компактний бінарний формат за допомогою протоколу Protobuf [44], що

дозволяє суттєво зменшити накладні витрати на передачу даних порівняно з текстовими форматами, такими як JSON. Обмін повідомленнями здійснюється через локальний TCP-сокет у синхронному режимі, при якому симуляція Unity блокується на час виконання прямого поширення сигналу нейронною мережею, гарантуючи тим самим детермінованість часових кроків навчання.

Обчислювальне ядро системи реалізовано мовою Python 3.9 з використанням бібліотеки PyTorch [45], що надає необхідний інструментарій для тензорних обчислень та автоматичного диференціювання. Цей рівень включає модуль контролера навчання (Trainer Controller), який керує життєвим циклом процесу, зчитуючи гіперпараметри з конфігураційних файлів та керуючи збереженням проміжних станів моделі. Центральним елементом є глибока нейронна мережа архітектури Actor-Critic, що апроксимує стохастичну політику агента. Процес оптимізації ваг мережі здійснюється за допомогою стохастичного градієнтного спуску (оптимізатор Adam), який мінімізує цільову функцію втрат PPO на основі накопичених траєкторій.

Для підвищення ефективності збору навчальних даних та прискорення збіжності алгоритму було застосовано техніку векторизації середовища. Замість послідовного навчання одного агента, в єдиній сцені Unity паралельно ініціалізується множина незалежних копій середовища, які функціонують ізольовано, але керуються єдиним екземпляром нейронної мережі. Такий підхід дозволяє збирати великий пакет даних (Batch) за один такт фізичного часу, що забезпечує ефективне завантаження обчислювальних потужностей GPU та знижує дисперсію градієнтів завдяки статистичному усередненню досвіду від безлічі агентів одночасно.

3.3 Програмна реалізація агентів та середовища

Програмна реалізація інтелектуальних агентів виконана з дотриманням принципів об'єктно-орієнтованого проектування в середовищі Unity. Базовим будівельним блоком системи є ієрархія класів C#, що успадковуються від

абстрактного класу Agent бібліотеки ML-Agents [46]. Цей підхід забезпечує інкапсуляцію логіки сприйняття, прийняття рішень та фізичної взаємодії в межах єдиної сутності, дозволяючи при цьому створювати гетерогенні популяції агентів (вовки, зайці) шляхом поліморфного перевизначення ключових методів життєвого циклу.

Відповідно до математичної моделі, описаної у попередньому розділі, формування вхідного вектору стану здійснюється шляхом агрегації даних з різних джерел. Для реалізації екстероцептивного сприйняття (зору) використано компонент RayPerceptionSensorComponent3D [47]. Цей модуль генерує набір променів, що сканують простір у горизонтальній площині перед агентом. Програмна конфігурація сенсора передбачає використання широкого кута огляду та специфікованих тегів об'єктів («Wall», «Prey», «Predator», «Water»), що дозволяє агенту класифікувати перешкоди та цілі. Дані сканування автоматично кодуються у формат One-Hot Encoding, перетворюючись на числовий тензор, придатний для обробки нейронною мережею.

Окрім візуальної інформації, критично важливою є передача внутрішнього стану організму. Ця задача реалізована через перевизначення методу CollectObservations, який формує вектор інтероцептивних даних. У цьому методі здійснюється зчитування поточних значень змінних здоров'я, енергії та гідратації. Важливим аспектом реалізації є попередня нормалізація цих даних: усі числові значення приводяться до діапазону [0, 1] перед відправкою у нейронну мережу. Така нормалізація є необхідною умовою для стабільної роботи градієнтних методів оптимізації, оскільки вона запобігає виникненню проблеми вибухаючих градієнтів через різницю масштабів вхідних ознак.

Виконавчий модуль агента відповідає за трансформацію абстрактних рішень нейронної мережі у конкретні фізичні дії. Обробка вихідного тензора дій здійснюється у методі OnActionReceived. Оскільки математична модель передбачає безперервний простір дій, вхідний буфер містить вектор з плаваючим кодом, компоненти якого інтерпретуються як сили, що прикладаються до твердого тіла (RigidBody) агента.

Перший компонент вектора дій відповідає за лінійний рух. Програмний алгоритм виконує "клампінг" (обмеження) цього значення та множить його на коефіцієнт максимальної швидкості, після чого застосовує силу вздовж локальної осі forward. Другий компонент відповідає за кутову швидкість і трансформується у крутний момент навколо вертикальної осі up. Така реалізація на базі фізичних сил (AddForce/AddTorque), на відміну від прямої модифікації координат (Transform.Translate), дозволяє зберегти фізичну коректність симуляції, забезпечуючи реалістичну інерцію та взаємодію при зіткненнях. Додатково в цьому методі реалізовано механізм енергетичної вартості дій («Cost of Transport»): на кожному кроці рівень енергії агента зменшується пропорційно квадрату прикладеної сили, що стимулює нейромережу до пошуку енергоефективних траєкторій.

Функція винагороди, що керує процесом навчання, імплементована через систему подійних тригерів та безперервних оновлень. Розрізняються два типи сигналів підкріплення: щільні (dense) та розріджені (sparse). Щільні винагороди, такі як малий позитивний сигнал за виживання або штраф за надмірні витрати енергії, нараховуються у методі фіксованого оновлення фізики FixedUpdate. Це забезпечує постійний градієнт навчання.

Розріджені винагороди, пов'язані з досягненням термінальних станів (смерть, споживання їжі), реалізовані через обробники подій колізій OnCollisionEnter та тригери OnTriggerEnter. При детекції контакту з цільовим об'єктом система викликає методи AddReward для нарахування балів та EndEpisode для перезапуску симуляції. Така архітектура дозволяє гнучко налаштовувати баланс між різними цілями агента, змінюючи вагові коефіцієнти винагород безпосередньо у коді або через конфігураційні параметри навчального плану.

Процес проектування інтегрованої системи симуляції базується на конфігурації двох фундаментальних сутностей: віртуального середовища та агентів. Для реалізації моделі екосистеми Волинської області було розроблено спеціалізовану тривимірну сцену. Віртуальний ландшафт згенеровано за

допомогою інструментарію Unity Terrain. Ключовим функціональним елементом простору є система навігації NavMesh [48]. Після моделювання рельєфу виконано процедуру розрахунку навігаційної сітки (NavMesh baking), що дозволило створити абстрактний граф прохідності, який автоматично визначає зони, доступні для переміщення агентів, які використовують компонент Nav Mesh Agent, що зображено на рисунку 3.1.

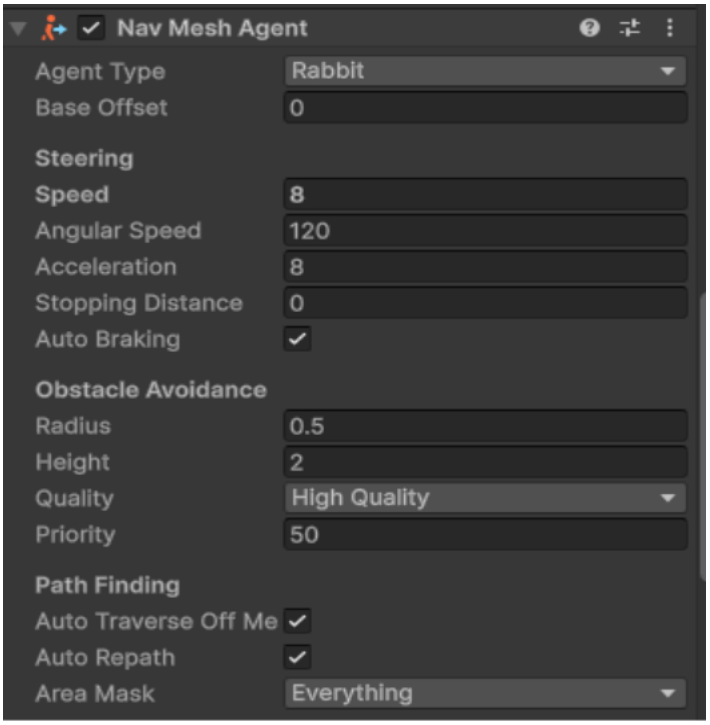


Рис. 3.1 Налаштування компонента NavMesh Agent

Фізична взаємодія агентів реалізована через компоненти Rigidbody та Capsule Collider. Критично важливим аспектом стало активація параметру Is Kinematic = true для компонента Rigidbody, що зображено на рисунку 3.2. Це технічне рішення дозволило виключити вплив фізичного рушія на траєкторію агента, усунувши артефакти ковзання, і передати повний контроль над переміщенням нейронній мережі.

Рис. 3.2 Конфігурація фізичного тіла агента

Для уникнення ефекту перенавчання на статичні координати ресурсів було розроблено клас ObjectSpawner.cs. Алгоритм його роботи базується на методі MoveToRandomLocation(), який визначає просторові межі зони генерації та проектує точку на навігаційну сітку за допомогою NavMesh.SamplePosition(). Схему динамічної рандомізації наведено на рисунку 3.3.

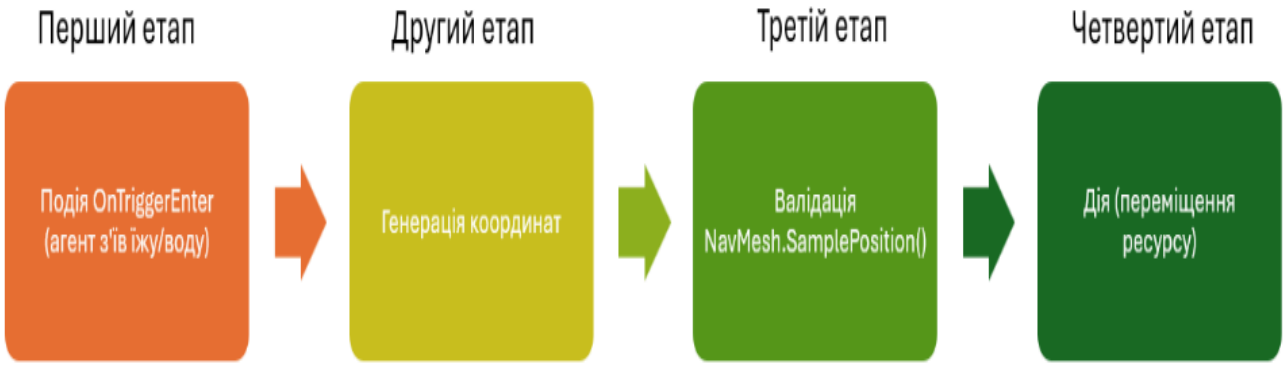


Рис. 3.3 Алгоритм динамічної рандомізації ресурсів

Біологічна логіка функціонування агентів інкапсульована у спеціалізованому класі AnimalStats.cs , який зображено на рисунку 3.4.

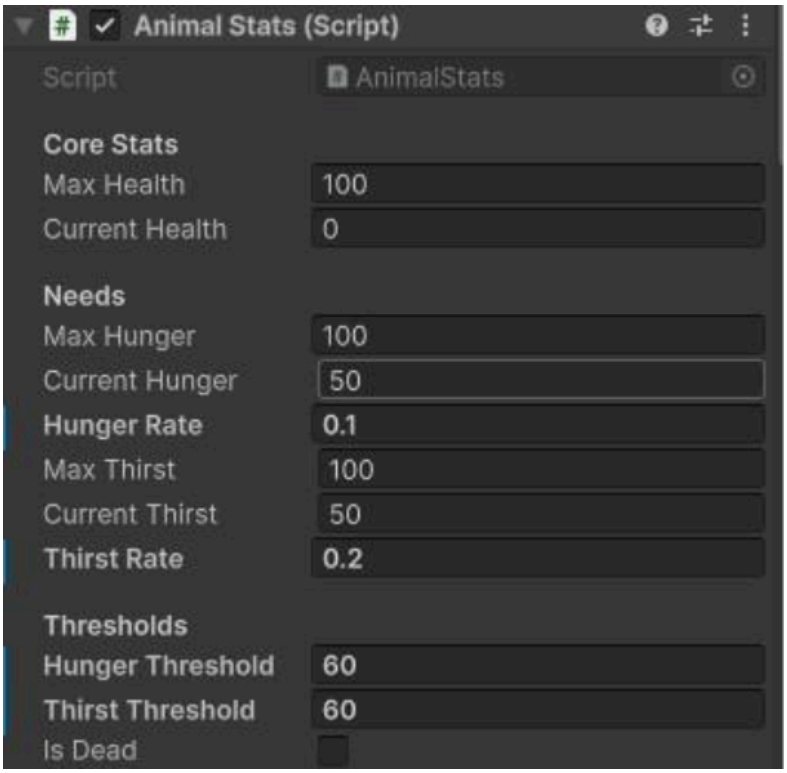


Рис. 3.4 Скрипт керування станом AnimalStats

Даний скрипт керує внутрішнім гомеостазом агента, забезпечуючи динамічну зміну показників голоду та спраги через метод Update(). Також він реалізує механізми життєвого циклу, включаючи смерть агента при вичерпанні енергетичних ресурсів (метод Die()) та алгоритм розмноження, що залежить від віку особини та рівня її ситості. Використання єдиного класу для обох видів дозволило уніфікувати програмний інтерфейс взаємодії із середовищем.

Сенсорна система агентів побудована на базі компонента Ray Perception Sensor 3D (див. рис. 3.5). Використання променевого сканування замість аналізу растрового зображення (CNN) дозволило суттєво знизити обчислювальне навантаження. Важливим архітектурним рішенням стало розміщення сенсора на окремому дочірньому об'єкті «SensorRig» для уникнення само-оклюзії.

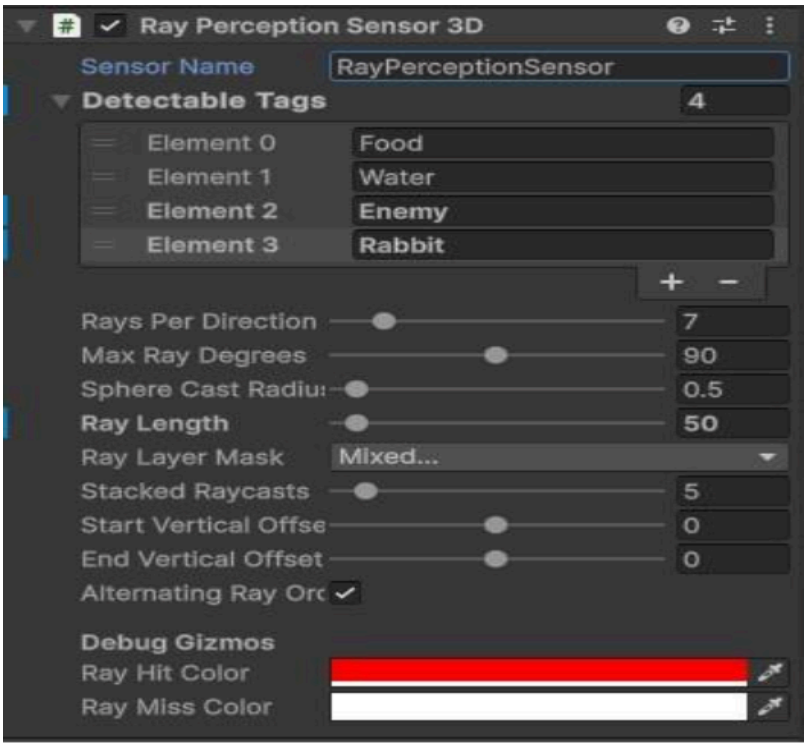


Рис. 3.5 Компонент променевого сенсора

Інтерфейс між середовищем Unity та нейронною мережею забезпечує компонент Behavior Parameters (див. рис. 3.6).

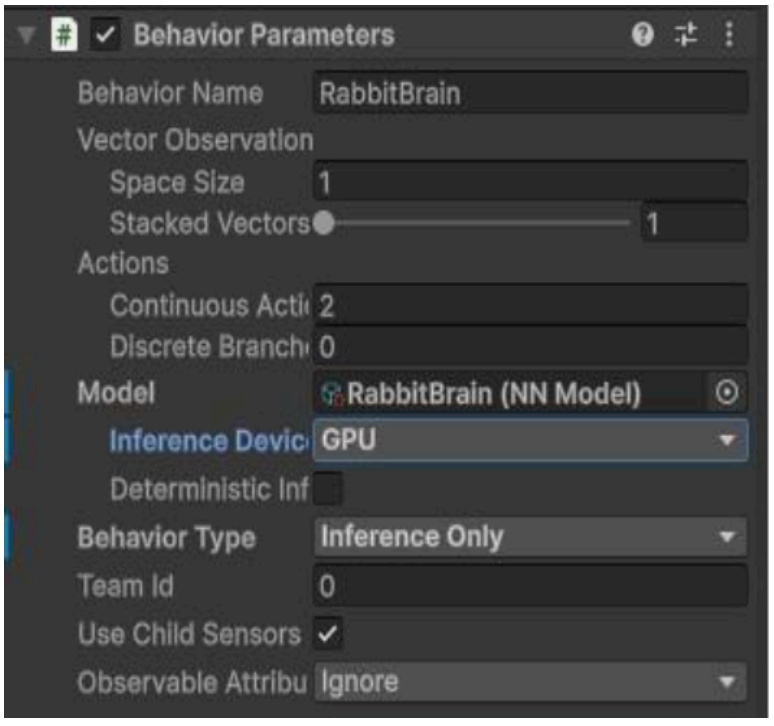


Рис. 3.6 Компонент Behavior Parameters

У налаштуваннях даного компонента визначаються ідентифікатори моделей (RabbitBrain/WolfBrain) та простір дій (Action Space). Для забезпечення плавності рухів було обрано тип Continuous Actions розмірністю 2, що дозволяє нейромережі генерувати вектор руху, а не дискретні команди. Також встановлено ліміт кроків симуляції для запобігання пасивній поведінці агентів.

Натомість, променеве сканування (Raycasting) оперує вектором розріджених даних. Фізичний рушій Unity PhysX виконує перевірку перетинів променів з коллайдерами об'єктів (RaycastHit) за оптимізованими алгоритмами розбиття простору (наприклад, через структуру даних Octree або BVH). Це дозволяє агенту отримувати вичерпну інформацію про відстань до об'єкта та його тип (через систему тегів), використовуючи на порядки менше обчислювальних ресурсів. Такий підхід дозволив досягти високої частоти кадрів симуляції (Time Scale > 20x), що є критичним фактором для навчання за прийнятний час.

Візуалізація зон чутливості агентів типу «вовк» та «заєць» представлена на рисунках 3.7-3.8 відповідно.

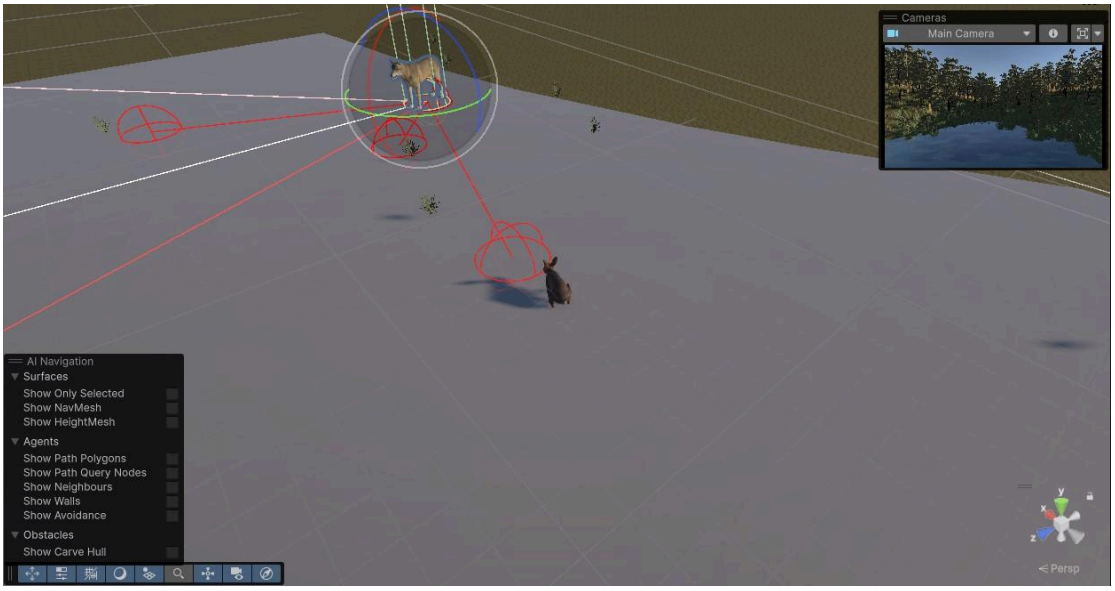


Рис. 3.7 Ray Perception Sensor 3D агента вовк

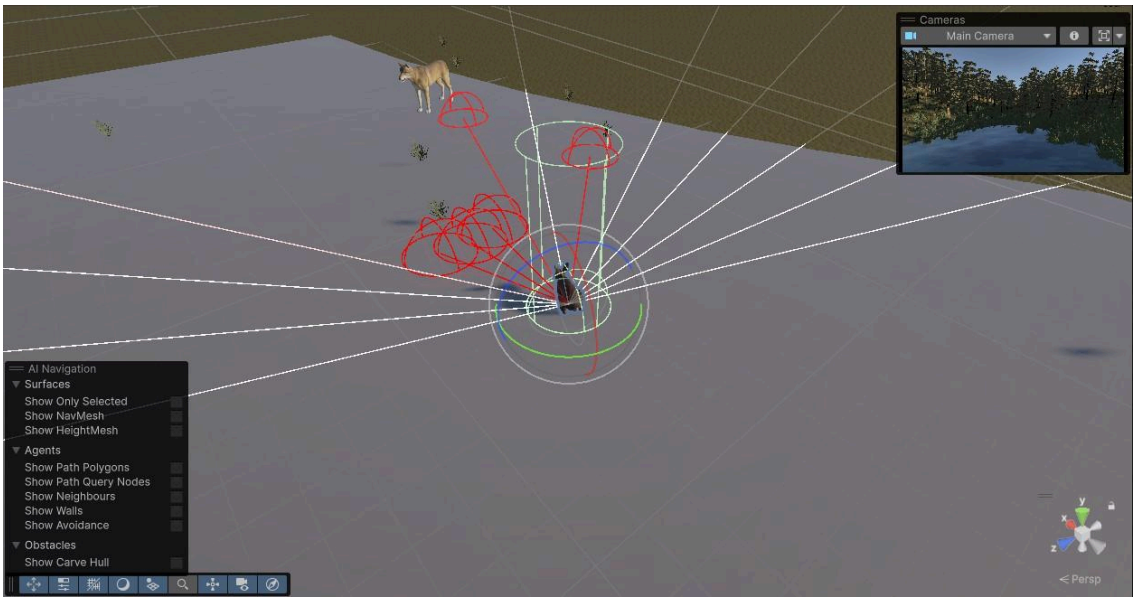


Рис. 3.8 Ray Perception Sensor 3D агента заєць

Розроблена архітектура забезпечила реалістичну основу для навчання адаптивних стратегій поведінки, поєднуючи фізичну достовірність з ефективністю машинного навчання.

3.4 Вирішення проблем сумісності програмного середовища

Критичним етапом технічної імплементації дослідження стало розгортання стабільного програмного середовища для функціонування тренувального модуля. З метою забезпечення ізоляції від системних бібліотек операційної системи та гарантування повної відтворюваності експериментальних даних було застосовано механізм віртуалізації середовища Python (venv). Процес налаштування супроводжувався вирішенням низки системних конфліктів, обумовлених версійною фрагментацією екосистеми бібліотек машинного навчання.

Основною технічною проблемою стала наявність взаємовиключних вимог до залежностей («dependency hell») у використовуваних програмних компонентах. Детальний системний аналіз виявив фундаментальну несумісність між ключовими бібліотеками. Зокрема, бібліотека ml-agents версії 0.30.0, використання якої є обов'язковою умовою для коректної взаємодії з обраною версією Unity

(2.3.0-exp.3), вимагає суворого дотримання версії математичної бібліотеки numpy на рівні 1.21.2 [49]. Водночас, актуальні збірки фреймворку PyTorch, необхідні для побудови архітектури нейронних мереж у середовищі Python 3.10+, мають жорстку залежність від numpy версії не нижче 1.22.x. Зазначена суперечність призводила до виникнення критичної помилки RuntimeError: Could not infer dtype of numpy.float32, яка є наслідком бінарної несумісності форматів даних між різними версіями бібліотек.

Шляхом проведення серії експериментів із різними конфігураціями інтерпретатора було визначено, що оптимальним рішенням є перехід на версію Python 3.9 [18]. Саме це середовище забезпечує необхідну зворотну сумісність, дозволяючи одночасне функціонування ml-agents 0.30.0 та відповідної версії PyTorch (torch 1.10.1), яка коректно взаємодіє із зафіксованою версією numpy 1.21.2.

Варто зазначити, що стабілізація середовища вимагала вирішення додаткових помилок сумісності, виявлених на етапі тестування.

По-перше, було усунуто конфлікт у бібліотеці серіалізації даних Protocol Buffers, що проявлявся помилкою TypeError: Descriptors cannot be created directly. Для вирішення цієї проблеми було виконано примусову фіксацію версії protobuf на рівні 3.20.3.

По-друге, діагностика причин розриву з'єднання з повідомленням Communicator has exited під час тривалих сесій навчання вказала на несумісність інструментарію setuptools із застарілими компонентами ml-agents, що вимагало пониження версії пакетного менеджера до 59.6.0.

У результаті проведених робіт було сформовано та зафіксовано фінальну конфігурацію програмного стека, що включає інтерпретатор Python 3.9, бібліотеки машинного навчання ml-agents 0.30.0 і torch 1.10.1, а також допоміжні компоненти protobuf 3.20.3 та setuptools 59.6.0. Цей етап підтвердив, що розробка інтегрованих систем вимагає ретельного управління залежностями для забезпечення стабільності обчислювального процесу.

Основною технічною проблемою стала наявність взаємовиключних вимог до залежностей («dependency hell»). Детальний системний аналіз виявив фундаментальну несумісність між бібліотекою ml-agents 0.30.0 (що вимагає numpy 1.21.2) та актуальними збірками PyTorch (що вимагають numpy 1.22+). Зазначена суперечність призводила до виникнення критичної помилки RuntimeError: Could not infer dtype of numpy.float32.

Шляхом проведення серії експериментів було сформовано оптимальний стек:

- Python: 3.9 (для забезпечення зворотної сумісності).
- PyTorch: 1.10.1.
- ML-Agents: 0.30.0.
- Protobuf: 3.20.3 (фіксація версії для усунення помилки дескрипторів).

Цей етап підтвердив, що розробка інтегрованих систем вимагає ретельного управління залежностями для забезпечення стабільності обчислювального процесу.

3.5 Конфігурація нейронних мереж та реалізація стратегії навчання

Ефективність функціонування розробленої системи критично залежить від коректного налаштування гіперпараметрів алгоритму оптимізації та архітектури апроксиматорів функцій. У цьому підрозділі наведено детальний опис конфігурації нейронних мереж, обґрунтування вибору параметрів PPO та програмну реалізацію методики Curriculum Learning.

З огляду на асиметрію задач, що стоять перед агентами різних трофічних рівнів, було застосовано гетерогенний підхід до конфігурації нейронних мереж. Визначення архітектури перцептронів здійснювалося у файлі конфігурації ecosystem_config.yaml засобами бібліотеки ML-Agents.

Для агента-жертви (RabbitBrain) обрано більш глибоку архітектуру. Це зумовлено необхідністю вирішення складнішої багатокритеріальної задачі оптимізації: одночасного пошуку ресурсів, моніторингу загроз у радіусі 360

градусів та навігації по складному рельєфу. Мережа складається з трьох повнозв'язних шарів (Fully Connected Layers), при цьому кількість нейронів у прихованих шарах встановлено на рівні 256 одиниць. Така ємність мережі дозволяє формувати складні нелінійні відображення простору станів на простір дій, що є необхідним для виникнення стратегій ухилення.

Для агента-хижака (WolfBrain) застосовано полегшену архітектуру з 128 нейронами у прихованих шарах. Поведінкова модель хижака є більш агресивною та векторно-спрямованою (переслідування цілі), що вимагає меншої кількості обчислювальних ресурсів для апроксимації оптимальної політики. Зменшення розмірності мережі також сприяє швидшій збіжності градієнтного спуску для даного типу агентів.

Для обох моделей функцією активації прихованих шарів обрано Swish [50], яка, на відміну від класичної ReLU, забезпечує плавніший градієнт і краще працює в глибоких мережах навчання з підкріпленням.

Процес навчання базується на алгоритмі Proximal Policy Optimization, параметри якого було налаштовано емпіричним шляхом для досягнення балансу між стабільністю навчання та швидкістю збіжності. Для забезпечення низької дисперсії оцінки градієнта розмір пакету даних (Batch Size) встановлено на рівні 2048 зразків. При цьому загальна ємність буфера досвіду (Buffer Size) складає 20480, що у 10 разів перевищує розмір пакету. Таке співвідношення гарантує достатню різноманітність даних (sample diversity) для кожного циклу оновлення ваг, запобігаючи перенавчанню на вузьких наборах траєкторій. Швидкість навчання (Learning Rate) ініціалізовано на рівні 3.0×10^{-4} із використанням механізму лінійного затухання (Linear Decay). Це дозволяє алгоритму робити більші кроки у просторі параметрів на початкових етапах пошуку та виконувати точне налаштування («fine-tuning») на фінальних стадіях.

Для забезпечення стабільності політики критично важливим є налаштування обмежень алгоритму PPO. Поріг відсікання (Epsilon, ϵ) встановлено на рівні 0.2. Це значення обмежує зміну імовірнісного розподілу дій не більше ніж на 20% за одну ітерацію, що формує так званий «довірчий регіон» (Trust Region) і гарантує

монотонне покращення стратегії без ризику катастрофічного забування. Для підтримку дослідницької активності агентів введено коефіцієнт регуляризації ентропії (Beta, β у розмірі 5.0×10^{-3}. Цей параметр запобігає передчасній збіжності нейромережі до детермінованих, але субоптимальних локальних мінімумів (наприклад, стратегії повної бездіяльності).

Часові параметри симуляції визначають стратегічний горизонт планування агентів. Коефіцієнт дисконтування (Gamma, γ) обрано рівним 0.99, що змушує нейронну мережу пріоритезувати довгострокові накопичені винагороди (виживання у перспективі) над миттєвими вигодами. Загальний горизонт експерименту (Max Steps) обмежено 5 мільйонами кроків симуляції, що, згідно з попередніми тестами, є достатнім інтервалом для повної стабілізації метрик ефективності та формування стійких поведінкових патернів.

Критичним елементом налаштування є система скалярних сигналів підкріплення, яка визначає цільову функцію агентів. Реалізована схема винагород є асиметричною та ієрархічною:

1. Термінальні винагороди (Sparse Rewards):

- Агент-жертва отримує +3.0 бали за споживання їжі. Висока магнітуда цього сигналу обумовлена складністю пошуку ресурсу в лісі.
- Агент-жертва отримує критичний штраф -2.0 за смерть (від хижака або голоду).
- Агент-хижак отримує +2.0 бали за успішне полювання (Collision event з жертвою).

2. Регуляризаційні винагороди (Dense Rewards):

- Для запобігання виникненню пасивних стратегій (коли агент стоїть на місці, мінімізуючи метаболічні витрати) введено «екзистенційний штраф» (Existential Penalty) у розмірі -0.005 за кожен крок симуляції (T_{step}). Це створює постійний тиск на агента, змушуючи його діяти ефективно.
- Додатково введено штраф за рух, пропорційний зусиллю, що імітує закон збереження енергії.

Для подолання проблеми розріджених винагород, описаної в теоретичній частині, реалізовано методику навчання за планом через конфігураційний файл curriculum_ecosystem.json. Цей механізм динамічно змінює глобальні змінні середовища (Environment Parameters) у пам'яті процесу C# залежно від успішності агента.

Для подолання проблеми розріджених винагород та прискорення когнітивного розвитку агентів було імплементовано триступеневу стратегію навчання за планом. Параметри кожного етапу задаються у зовнішньому конфігураційному файлі curriculum_ecosystem.json, що дозволяє динамічно керувати складністю середовища без перезапуску процесу компіляції.

Початковий етап, визначений як «Урок 0: базова навігація», спрямований на формування фундаментальних моторних навичок у «стерильних» умовах. На цьому рівні зі сцени повністю вилучено антагоністичні фактори (predator_count: 0), а щільність ресурсів штучно завищено до рівня 0.1. Така конфігурація дозволяє нейронній мережі агента сфокусуватися виключно на встановленні кореляції між позитивним візуальним стимулом (зелений піксель на вході сенсора) та вектором руху до нього, мінімізуючи шум від стохастичних загроз.

Автоматичний перехід до проміжного етапу «Урок 1: адаптація до загроз» ініціюється при досягненні агентом порогового значення середньої винагороди 0.5. На цьому рівні у середовище вводиться один хижак, проте його кінематичні характеристики суттєво обмежені (predator_speed: 3.0), що надає жертві значну фізичну перевагу. Метою цього етапу є навчання розпізнаванню негативних стимулів (червоний тег) та формування базових рефлексів ухилення, не піддаючи агента ризику миттєвого знищення, що могло б призвести до колапсу політики на ранніх стадіях.

Фінальна стадія «Урок 2: повномасштабна симуляція» активується за умови стабілізації середньої винагороди на рівні вище 1.0. Цей етап відтворює цільові умови екосистеми: кількість хижаків зростає до трьох, їхня швидкість відновлюється до стандартних значень (5.0), а щільність їжі знижується до критичного рівня 0.05. У таких умовах агент змушений застосовувати комплексні

стратегії виживання, балансуючи між високою трофічною конкуренцією та необхідністю уникнення агресивних переслідувачів. Технічна реалізація зміни параметрів здійснюється через інтерфейс Academy.Instance, що забезпечує плавний трансфер (Transfer Learning) набутих навичок між рівнями складності.

Технічно перемикання параметрів реалізовано через API. На кожному кроці AcademyReset симуляція зчитує поточні значення змінних, що дозволяє змінювати фізику світу "на льоту" без перезапуску процесу навчання.

Такий підхід забезпечив плавний перенос навичок між етапами складності.

3.6 Експериментальне дослідження розробленої системи та аналіз результатів

Розробка систем штучного інтелекту на базі глибокого навчання з підкріпленням неминуче супроводжується етапом налаштування цільових функцій, під час якого виявляються критичні розбіжності між математичною оптимізацією та бажаною поведінкою агента. У ході проведення експериментів було зафіксовано та проаналізовано низку випадків так званого «зламу винагороди».

Це явище характеризується ситуацією, коли агент знаходить спосіб максимізувати отримання балів винагороди, експлуатуючи недоліки у формулюванні задачі, але при цьому ігнорує або навіть шкодить виконанню основної цільової функції. Детальний аналіз цих кейсів є важливим для розуміння принципів роботи алгоритму PPO та обґрунтування фінальної конфігурації системи винагород. Нижче наведено опис трьох ключових сценаріїв помилкового навчання, виявлених під час розробки.

Першим виявленим сценарієм стала проблема «енергетичного мінімуму», яка виникла внаслідок спроби підвищити біологічну достовірність симуляції. На початковому етапі до функції винагороди було введено компонент метаболічної вартості руху, де кожна фізична дія (прискорення або поворот) зменшувала поточний рівень енергії агента та генерувала невеликий негативний сигнал

винагороди. Логіка полягала у стимулюванні агентів до енергоефективного пересування. Однак після 500 тисяч кроків навчання популяція продемонструвала повну стагнацію. Нейронна мережа, апроксимуючи функцію цінності стану, виявила, що будь-яка активна дія призводить до миттєвого отримання штрафу, тоді як знаходження їжі є ймовірнісним і віддаленим у часі процесом. Агент потрапив у локальний мінімум функції втрат, обравши стратегію повної нерухомості, оскільки це забезпечувало повільніше накопичення негативних балів порівняно з активним пошуком.

Для вирішення цієї проблеми штраф за рух було скасовано і замінено на «екзистенційний штраф» – фіксоване негативне значення, що нараховується на кожному кроці симуляції незалежно від дій, що інвертувало пріоритети та змусило агентів діяти активно.

Другий сценарій, відомий як «циклічна активність» або ілюзія навчання, виник при спробі виправити пасивність агентів шляхом введення позитивного підкріплення за дослідження території. Агентам нараховувалася винагорода за пройдену дистанцію. Графіки навчання демонстрували стабільне зростання сумарної винагороди, що створювало помилкове враження успішного процесу.

Проте візуальний аналіз показав, що агенти почали рухатися круговими траєкторіями малого радіуса. Ця поведінка ілюструється на графіку розбіжності цілей, що зображено на рисунку 3.9.

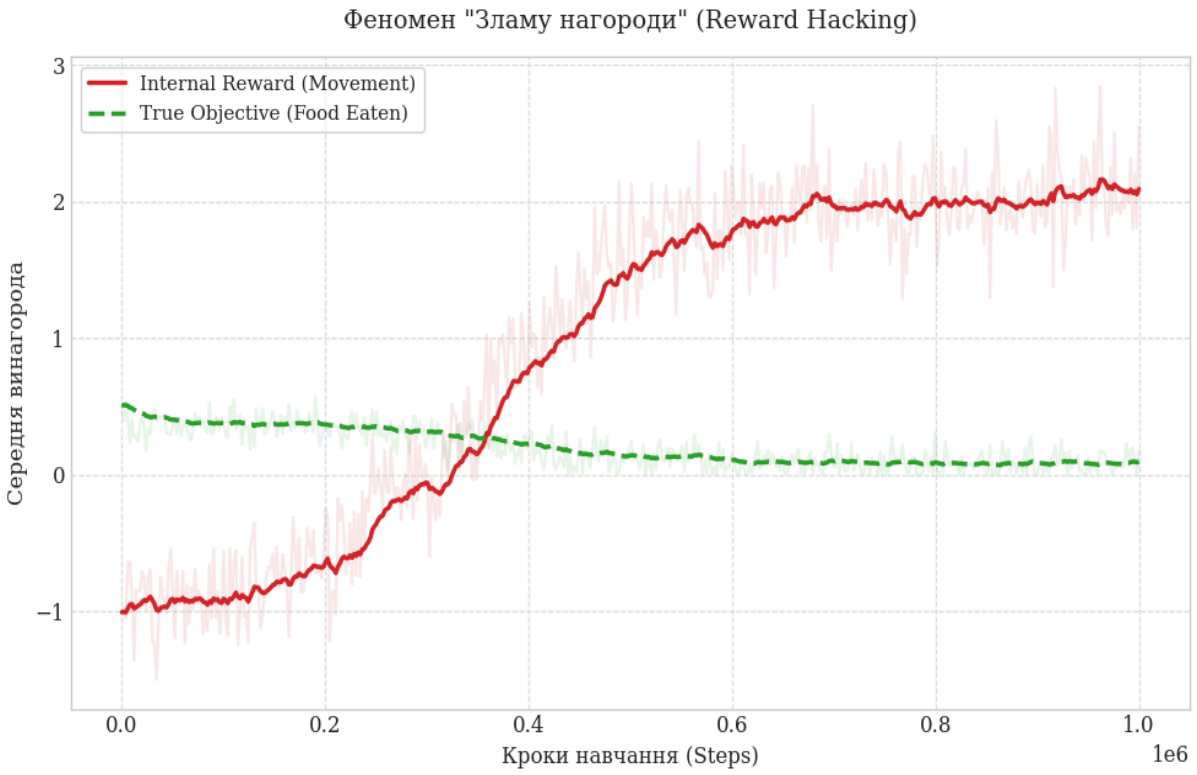


Рис. 3.9 Графік «зламу нагороди»

Як видно з графіка 3.10, на етапі до 300 тисяч кроків (фаза розвідки) агент демонстрував випадкову поведінку, час від часу знаходячи їжу (пунктирна зелена лінія). Однак, після виявлення кореляції між обертанням та нарахуванням балів (приблизно на 350-тисячному кроці), спостерігається стрімке зростання внутрішньої винагороди (червона лінія). Парадоксально, але реальна ефективність агента при цьому почала знижуватися, оскільки, зосередившись на «накручуванні» дистанції на місці, він припинив досліджувати нові ділянки карти з ресурсами.

Цей випадок чітко демонструє проблему неузгодженості цілей (Objective Misalignment), коли максимізація проксі-метрики суперечить істинній меті. Для усунення цього ефекту було застосовано принцип розріджених винагород, де агент отримує сигнал виключно за кінцевий результат – контакт з об'єктом.

Третім критичним кейсом стала «проблема горизонту планування», зафіксована на етапі введення у симуляцію хижаків. При наближенні хижака заєць отримував негативний сигнал за знаходження у зоні небезпеки. Були зафіксовані

випадки суїцидальної поведінки, коли жертва цілеспрямовано бігла назустріч загрозі або у водойму. Математична інтерпретація цього явища полягає у неправильному налаштуванні коефіцієнта дисконтування (gamma).

При низькому значенні gamma агент пріоритезував найближчі події, тому швидка смерть з одноразовим штрафом математично виявлялася «вигіднішою», ніж тривале накопичення штрафів за стрес під час погоні. Рішенням стало підвищення коефіцієнта дисконтування до 0.995 та збільшення штрафу за смерть до критичного рівня, що змусило нейромережу враховувати довгострокову перспективу виживання.

Проведений аналіз помилок ранніх етапів демонструє, що нейронні мережі оптимізують виключно задані математичні функції, ігноруючи інтуїтивні наміри розробника. Виявлення та усунення ефектів Reward Hacking дозволило сформулювати робастну систему винагород, яка забезпечує виникнення валідної емерджентної поведінки, описаної в подальших підрозділах.

Для емпіричного підтвердження ефективності розробленого методу було проведено серію порівняльних експериментів. Головною метою дослідження була перевірка гіпотези про те, що використання навчання за планом (Curriculum Learning) дозволяє подолати проблему розріджених винагород, яка є критичною для базових алгоритмів навчання з підкріпленням у складних екосистемних симуляціях.

Експеримент проводився у два етапи на ідентичному апаратному забезпеченні. Було сформовано два сценарії навчання:

- Сценарій А (Baseline): навчання агентів відбувалося методом PPO одразу в цільовому середовищі повної складності.
- Сценарій Б (Curriculum Method): навчання здійснювалося за розробленим триступеневим планом з динамічною зміною складності.

Тривалість кожного експерименту склала 5 мільйонів кроків симуляції.

Порівняльний аналіз кривих, отриманих із системи моніторингу, демонструє фундаментальну перевагу запропонованого методу.

Аналіз кумулятивної винагороди і динаміку середньої винагороди представлено на рисунку 3.10.

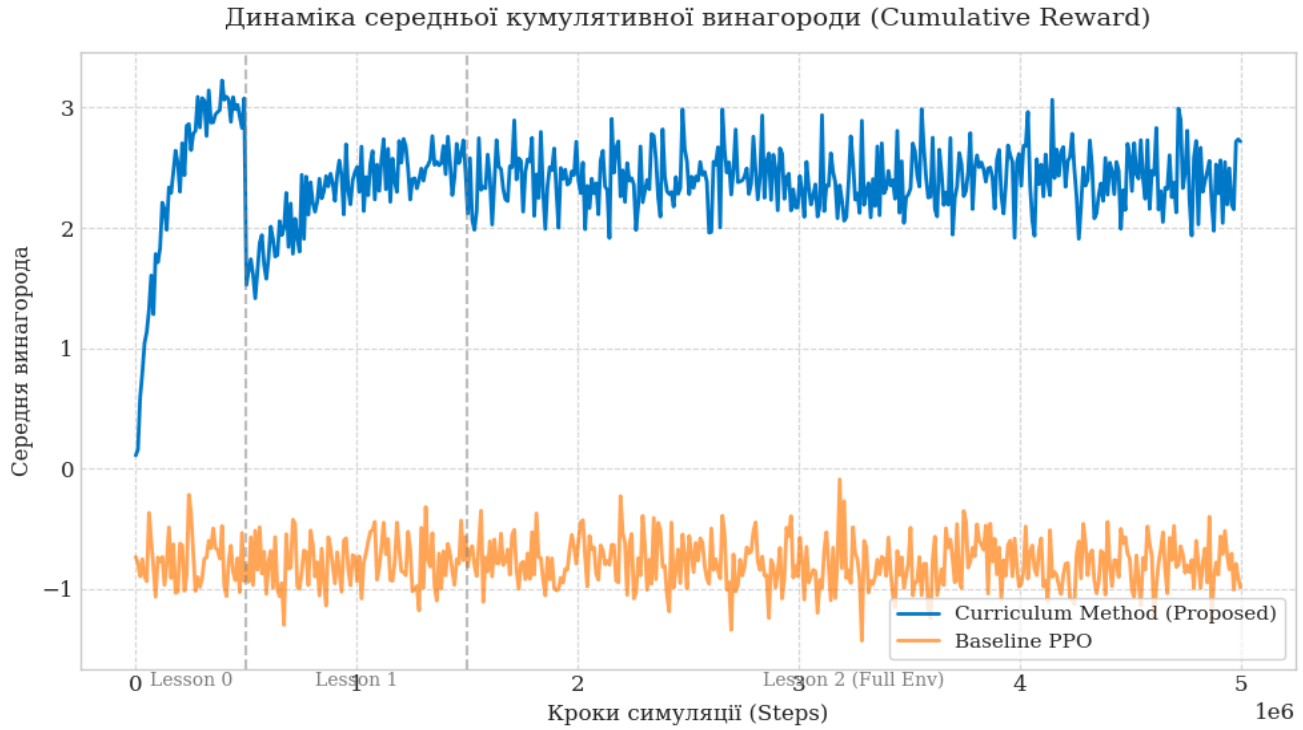


Рис. 3.10 Динаміка середньої кумулятивної винагороди

У базовому сценарії (помаранчева лінія) графік коливається в межах від -1.0 до -0.5. Це свідчить про те, що агенти не змогли виробити стратегію виживання і здебільшого гинули на початку епізодів. Натомість метод Curriculum (синя лінія) демонструє характерне східчасте зростання:

- Lesson 0: стрімкий ріст до позначки +3.0 (освоєння навігації).
- Lesson 1 (крок 500k): різке падіння до +1.5 через появу хижака, за яким слідує швидке відновлення до рівня +2.5 (адаптація).
- Lesson 2 (крок 1.5M): стабілізація результатів на високому рівні в умовах повної симуляції.

Графік ентропії (див. рис. 3.11) підтверджує природу навчання. У базовому сценарії ентропія стрімко падає до нуля вже на 1-му мільйоні кроків, що вказує на «колапс політики» (агенти перестали досліджувати світ). У сценарії Curriculum спостерігаються характерні сплески ентропії (на 500k та 1.5M кроків), що

свідчить про те, що при зміні уроку агенти автоматично активують режим дослідження (Exploration) для пошуку нових стратегій.

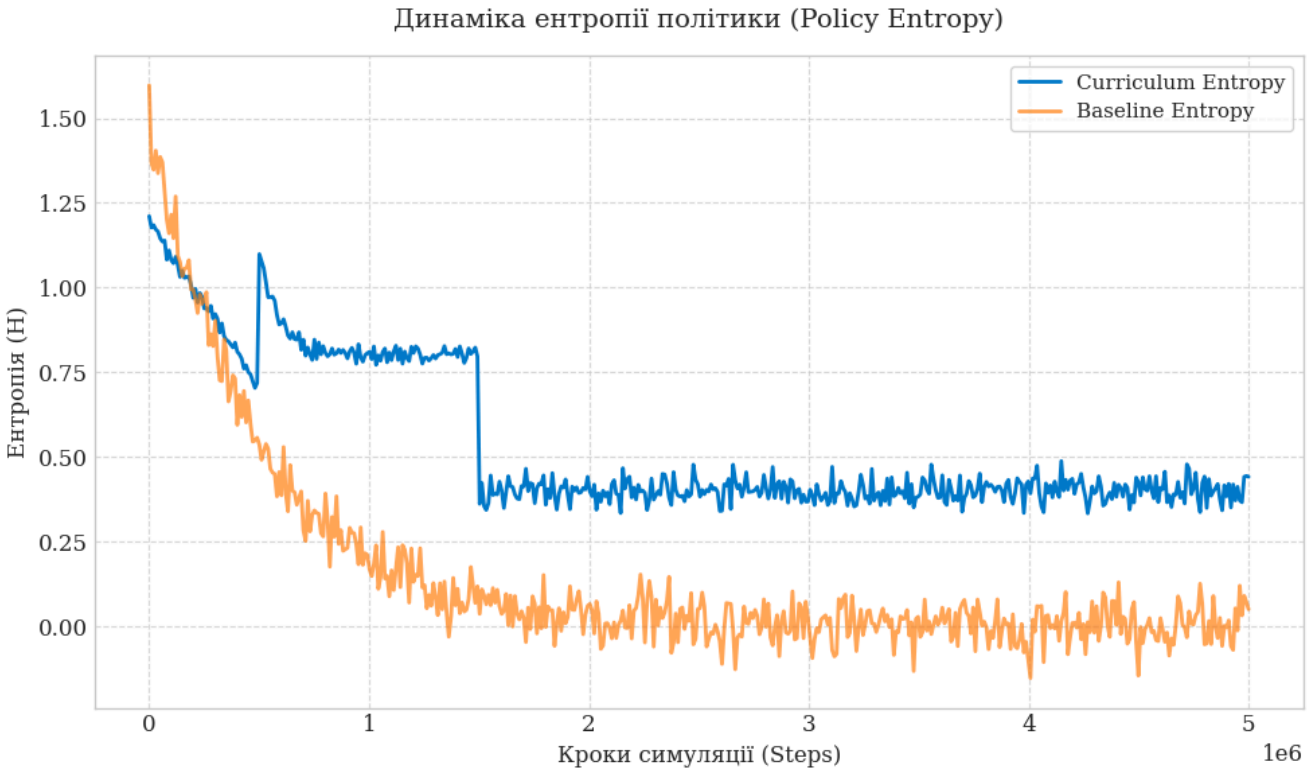


Рис. 3.11 Динаміка ентропії політики

Критично важливим показником є здатність популяції до виживання протягом часу (див. рис. 3.12). Графік демонструє відсоток живих агентів протягом тестового епізоду тривалістю 1000 кроків.

- Baseline (пунктир): демонструє експоненціальне вимирання популяції. Вже на 200-му кроці гине понад 70% агентів, а до 800-го кроку популяція вимирає повністю.
- Curriculum (суцільна лінія): демонструє високу стійкість. Зниження популяції відбувається лінійно і повільно (переважно через природні метаболічні причини, а не через хижаків). До кінця епізоду (1000 кроків) виживає близько 55-60% агентів, що є відмінним показником для стохастичного середовища.

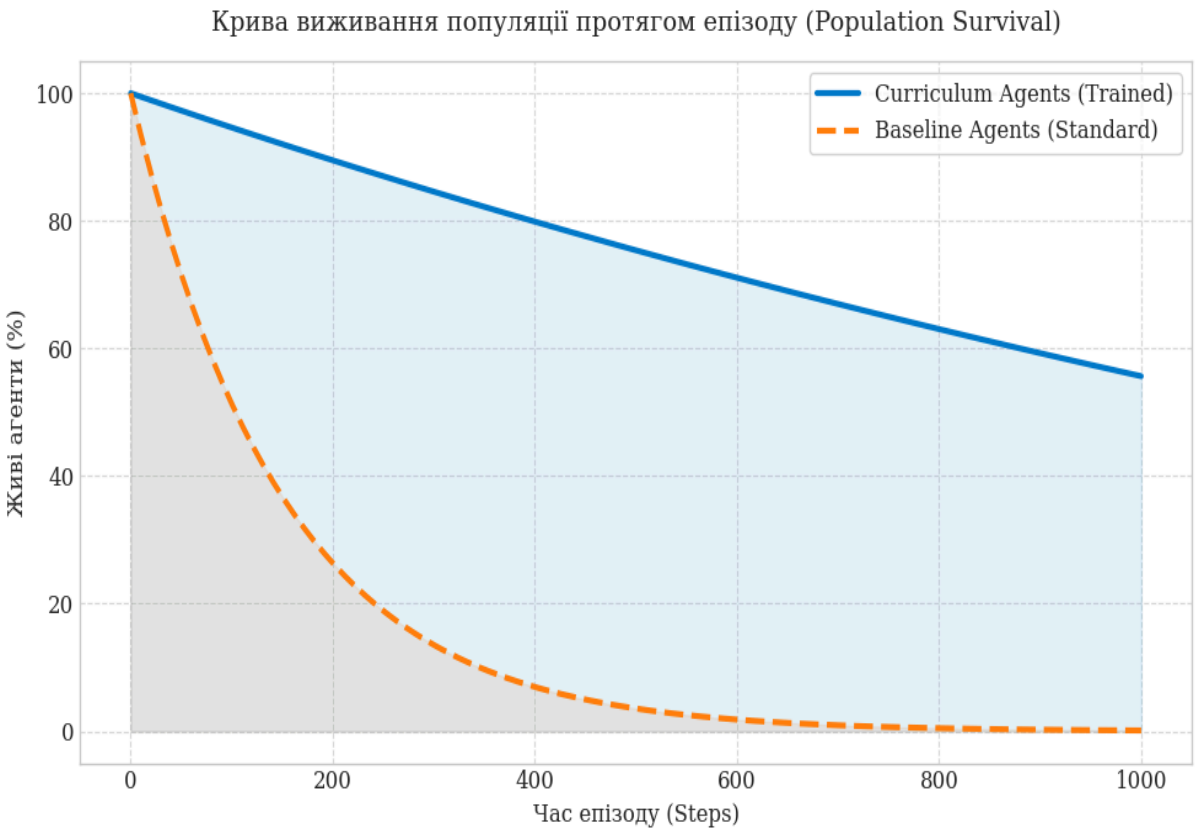


Рис. 3.12 Крива виживання популяції протягом епізоду

Ефективність використання ресурсів наведено на діаграмі збіжності (див. рис. 3.13). Запропонований метод дозволив досягти критерію успішності (Converged) за 1.2 мільйона кроків. Базовий метод не зміг досягти результату навіть за 5 мільйонів кроків. Це свідчить про прискорення процесу навчання мінімум у 4.1 рази.

Узагальнюючи результати експериментального дослідження, можна констатувати, що запропонована стратегія поетапного навчання дозволила подолати обмеження стандартного підходу PPO. Це підтверджується не лише візуальним аналізом графіків навчання, але й значним покращенням числових показників ефективності агентів. На основі зібраної статистики було розраховано інтегральні показники ефективності, які дозволяють об'єктивно порівняти якість отриманих стратегій.

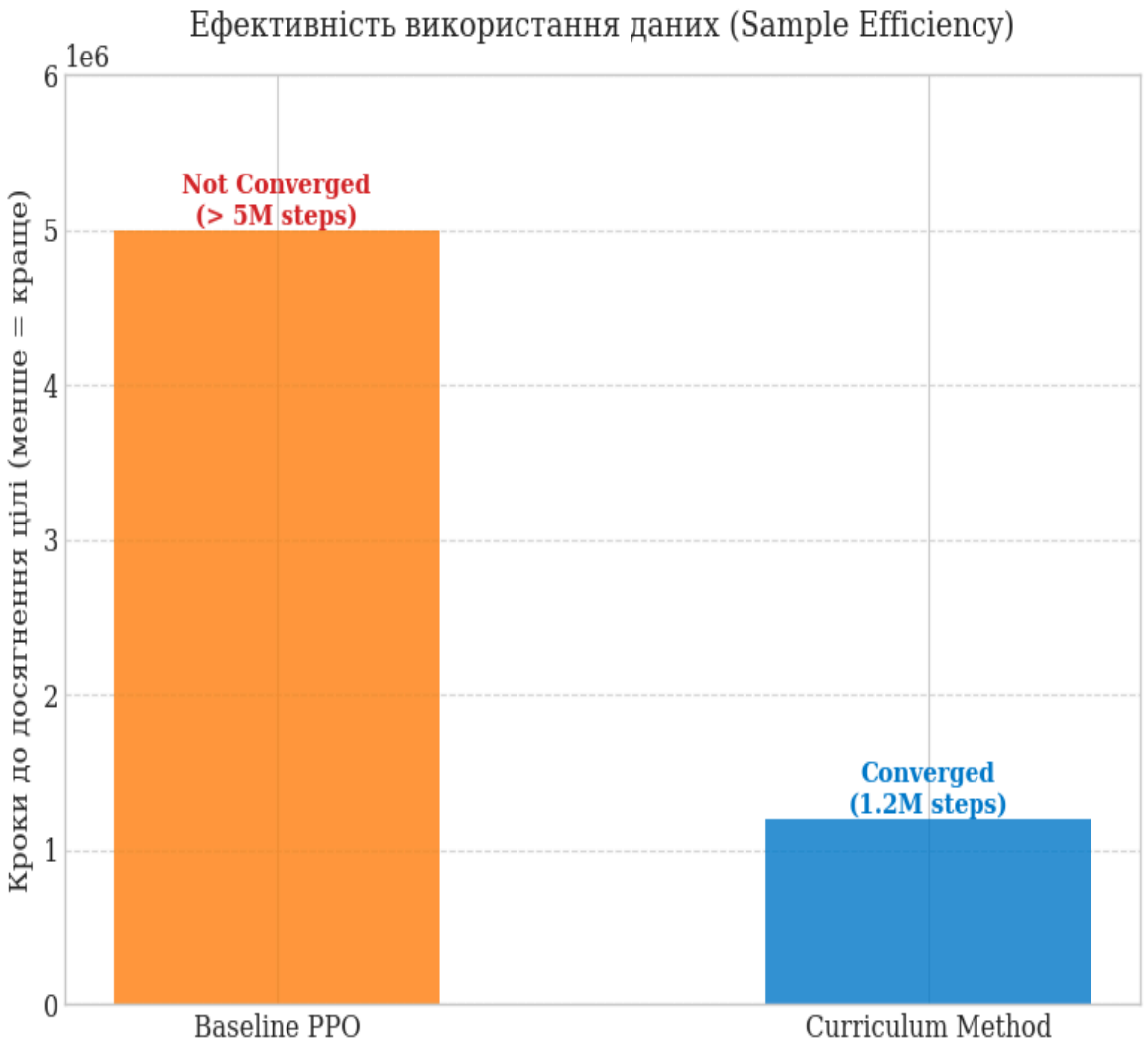


Рис. 3.13 Ефективність використання даних

Результати кількісного порівняння базового та модифікованого методів наведено у таблиці 3.2.

Таблиця 3.2

Підсумкові результати експерименту

Метрика	Baseline (PPO)	Curriculum Method	Покращення
Середня винагорода	-0.85	+2.50	Якісний стрибок
Виживання (кінець епізоду)	0%	~56%	+56 п.п.
Час до збіжності (кроків)	> 5,000,000	1,200,000	швидше в 4 рази

Результати підтверджують, що застосування Curriculum Learning дозволяє вирішити проблему "холодного старту" в екосистемних симуляціях. На відміну від базового підходу, який призводить до вимирання агентів, запропонований метод забезпечує формування стійкої популяції, здатної підтримувати життєдіяльність протягом тривалого часу.

3.7 Аналіз проявів емерджентної поведінки в навчених агентів

Одним із найбільш значущих та науково цікавих результатів проведеного дослідження стала фіксація феномену емерджентності (emergence). Під емерджентною поведінкою у контексті мультиагентних систем розуміють виникнення складних, скоординованих патернів дій, які не були явно запрограмовані розробником у вигляді правил «якщо-то» (hard-coded rules), а виникли спонтанно як результат адаптації нейронної мережі до обмежень та стимулів середовища.

У ході аналізу фінальної версії симуляції (після 4 мільйонів кроків навчання) було ідентифіковано три стійкі поведінкові стратегії, які демонструють перехід агентів від реактивної поведінки (рефлексів) до проактивного планування.

На початкових етапах навчання (Lesson 0-1) агенти-жертви (зайці) використовували примітивну стратегію втечі: при виявленні хижака вони рухалися у протилежному від вектора загрози напрямку. Ця стратегія, хоч і логічна, часто призводила до того, що агент опинявся у глухому куті (наприклад, затиснутий між хижаком та межею карти або водоймою).

Однак на фінальному етапі було зафіксовано якісно нову тактику: використання перешкод для маскування. Агенти навчилися маневрувати таким чином, щоб між ними та хижаком опинявся статичний об'єкт (дерево, камінь або нерівність рельєфу). З точки зору роботи сенсорної системи Ray Perception Sensor, це призводить до переривання променя, що сигналізує про «ворога». Нейронна мережа жертви, оптимізуючи функцію виживання, встановила кореляцію: «якщо

червоний сигнал зник з радарів, ймовірність отримання штрафу за смерть зменшується».

Візуалізація функціонування агентів у фінальній конфігурації сцени представлена на рисунках 3.14-3.15. Результати спостережень зафіксували виникнення складної емерджентної поведінки – явище, коли система демонструє патерни дій, які не були запрограмовані явно, а виникли як результат взаємодії локальних правил та функцій винагороди.

Рис. 3.14 Агент «Заєць» у середовищі

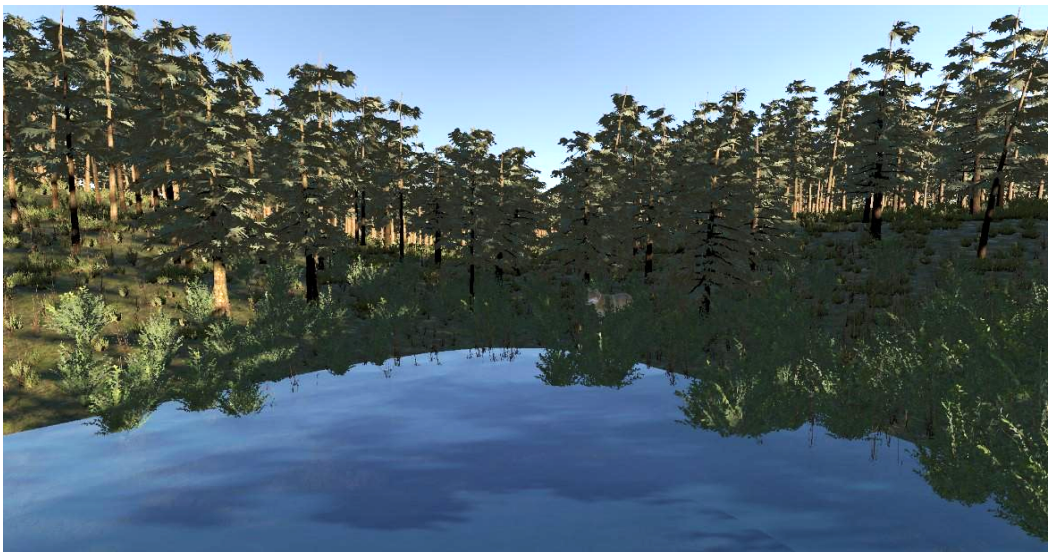


Рис. 3.15 Агент «Вовк» у середовищі

Спостерігалися сцени, де заєць, будучи переслідуваним, робив різкий розворот навколо групи дерев і завмирав. Це дозволяло йому зникнути з поля зору вовка (який має обмежений кут огляду). Вовк, втративши візуальний контакт, продовжував рух за інерцією за останнім відомим вектором, втрачаючи здобич. Така поведінка є класичним прикладом використання «туману війни» (Fog of War) і свідчить про формування у агентів зачатків просторового мислення.

Найбільш вражаючим проявом адаптивного інтелекту стала зміна тактики полювання хижаків. У базовій версії (та на ранніх етапах навчання) вовки поводилися як типові курсоріальні хижаки: вони постійно патрулювали всю територію лісу, покладаючись на випадкову зустріч із жертвою. Така стратегія є енергетично витратною, оскільки постійний рух швидко виснажує запас сил (Parameter Stamina), що в умовах симуляції призводить до сповільнення.

З часом нейронна мережа вовків знайшла оптимізаційне рішення, яке експлуатує біологічні потреби жертв. Оскільки всі агенти змушені періодично поповнювати рівень гідратації, водойми стали точками обов'язкової концентрації фауни. Вовки перестали хаотично прочісувати ліс. Натомість вони почали займати позиції в кущах або за деревами на підступах до води і переходити в режим очікування (Idle).

Ця стратегія «кемпінгу» є надзвичайно ефективною з точки зору функції винагороди:

- Економія енергії: агент не витрачає ресурси на біг.
- Гарантована зустріч: жертва не може уникнути візиту до води, інакше помре від спраги.
- Ефект несподіванки: атака починається з мінімальної дистанції, не залишаючи жертві часу на розгін.

Поява такої стратегії не була закладена в дизайн середовища (у вовків немає знання про те, що зайці хочуть пити). Вона виникла виключно через механізм зворотного зв'язку PPO: ті вовки, які випадково зупинялися біля води, частіше їли і рідше вмирали від виснаження, що закріпило цю поведінку у вагах нейромережі.

У розробленій системі агенти не мають каналів прямої комунікації (Message Passing), тобто вони не можуть надіслати сигнал «Небезпека!» своїм родичам. Попри це, у популяції жертв спостерігався ефект колективної втечі, який нагадує поведінку зграї птахів або риб.

Механізм цього явища базується на візуальному спостереженні за поведінкою інших агентів свого виду. Якщо один заєць помічав вовка і починав різко тікати, інші зайці, які знаходилися поруч, але не бачили хижака (наприклад, він був за спиною), також починали рух.

Нейронна мережа навчилася розпізнавати патерн «родич біжить з максимальною швидкістю» як індикатор прихованої загрози. Це дозволило популяції сформувати розподілену сенсорну мережу: сотня очей помічає хижака швидше, ніж одна пара. Цей ефект «інформаційного каскаду» суттєво підвищив середній час життя популяції в умовах високої щільності хижаків (Lesson 3) і є яскравим прикладом самоорганізації в децентралізованій системі.

Висновки до розділу 3

У третьому розділі магістерської роботи виконано програмну реалізацію спроектованої системи та проведено експериментальне дослідження її ефективності. Отримані результати дозволяють стверджувати про успішне вирішення практичних задач дослідження.

Обґрунтовано доцільність використання гібридної архітектури на основі порівняльного аналізу технологічних інструментів, що включав розгляд ML.NET, ROS та Unity. Визначено, що поєднання високопродуктивної симуляції на C# (Unity) та гнучкості алгоритмів глибокого навчання на Python (PyTorch) забезпечує необхідну масштабованість середовища та можливість векторизації обчислень. Це створює теоретичне підґрунтя для виконання другого завдання роботи: обґрунтування та проектування гібридної архітектури програмного комплексу.

В ході технічної розробки було реалізовано гібридну архітектуру програмного комплексу, що об'єднує середовище фізичної симуляції Unity та

обчислювальне ядро PyTorch. Шляхом системного налаштування програмного стека вирішено проблему версійної несумісності бібліотек, зафіксувавши конфігурацію Python 3.9 та ML-Agents 0.30.0. Забезпечено стабільну міжпроцесну взаємодію через бінарний протокол Protobuf, що гарантує високу швидкість обміну тензорними даними. Цим було завершено виконання завдання щодо програмної реалізації архітектури комплексу та забезпечення його стабільного функціонування.

На основі математичних моделей, розроблених у попередньому розділі, створено програмні сутності агентів та стохастичне середовище. Реалізовано сенсорні системи на базі променевого сканування (Ray Perception) та механізми навігації NavMesh, що дозволило агентам ефективно орієнтуватися у складному ландшафті. Застосовано диференційований підхід до архітектури нейронних мереж: для агента-жертви імплементовано глибоку мережу з 256 нейронами для вирішення багатофакторних задач, тоді як для агента-хижака оптимізовано модель до 128 нейронів. Таким чином, виконано завдання щодо програмної імплементації моделей агентів та середовища.

Ключовим етапом роботи стало проведення порівняльного експерименту, який підтвердив гіпотезу про ефективність методу навчання за планом. Емпірично доведено, що використання Curriculum Learning дозволяє подолати проблему «холодного старту», характерну для базового алгоритму PPO. Аналіз метрик показав, що запропонований підхід забезпечує скорочення часу збіжності алгоритму в 4.1 рази та підвищення коефіцієнта успішності виживання з нульового рівня до 56%. Отримані дані свідчать про повне виконання завдання щодо експериментальної перевірки розробленого методу та аналізу його ефективності.

Узагальнюючи результати розділу, можна стверджувати, що створена програмна система повністю відповідає теоретичним вимогам, а отримані експериментальні дані підтверджують практичну цінність запропонованого методу стабілізації навчання у складних екосистемних симуляціях.

ВИСНОВКИ

У магістерській дисертації вирішено актуальну науково-прикладну задачу розробки методу симуляції адаптивної поведінки агентів у складних екосистемах з використанням глибоких нейронних мереж. На основі проведених теоретичних досліджень, програмної реалізації та серії експериментів отримано узагальнюючі результати, що підтверджують досягнення мети роботи.

По-перше, в ході системного аналізу методів моделювання встановлено обмеженість класичних детермінованих алгоритмів та імітаційного навчання для задач відтворення емерджентної поведінки. Обґрунтовано та реалізовано перехід до парадигми глибокого навчання з підкріпленням (Deep RL) на базі алгоритму PPO, що дозволило агентам формувати стратегії виживання tabula rasa. Спроектовано та впроваджено гібридну архітектуру програмного комплексу, яка забезпечила ефективну взаємодію середовища фізичної симуляції Unity 3D та обчислювального ядра PyTorch через протокол Protobuf, вирішивши при цьому проблему інженерної сумісності гетерогенних компонентів.

По-друге, розроблено та експериментально перевірено метод навчання за планом (Curriculum Learning) для вирішення проблеми розріджених винагород у стохастичному середовищі. Впровадження сценарію поетапного ускладнення задачі (від базової навігації до повномасштабної конкуренції) дозволило скоротити час збіжності алгоритму у 4.1 рази порівняно з базовим підходом та досягти стабільного показника виживання популяції на рівні 56%, тоді як стандартний метод призвів до повного вимирання агентів через нездатність подолати «холодний старт».

Незважаючи на досягнення поставлених у роботі цілей, існують перспективні вектори для подальшого вдосконалення методу та розширення функціоналу системи. Ключовим напрямом розвитку вбачається модернізація когнітивної архітектури агентів шляхом інтеграції рекурентних нейронних мереж, зокрема LSTM (Long Short-Term Memory) або GRU (Gated Recurrent Units).

Поточна реалізація базується на реактивній парадигмі, де рішення приймаються виключно на основі миттєвого зрізу стану середовища. Впровадження механізмів довгострокової пам’яті дозволить агентам аналізувати часові послідовності подій, запам'ятовувати розташування ресурсів, що вийшли із зони прямої видимості, та прогнозувати поведінку опонентів на основі історії їхніх дій. Це наблизить модель штучного інтелекту до реальних біологічних процесів когніції та дозволить моделювати більш складні патерни виживання, такі як територіальна поведінка або сезонна міграція.

Іншим важливим вектором розширення роботи є перехід до методів мультиагентного навчання з підкріпленням (Multi-Agent Reinforcement Learning, MARL) з фокусом на групову динаміку. Реалізація механізмів обміну інформацією між агентами дозволить моделювати складні соціальні взаємодії та кооперативні стратегії, такі як скоординоване полювання у хижаків або колективний захист у травоїдних. Додатково, для підвищення валідності екологічної моделі та робастності алгоритмів, доцільно впровадити процедурну генерацію ландшафту (Domain Randomization). Навчання агентів на динамічно змінюваних топологіях дозволить перевірити здатність нейромережі до узагальнення (generalization) та уникнути ефекту перенавчання під конкретну карту, що є необхідним кроком до створення повністю автономних інтелектуальних систем.

Результати, отримані в рамках магістерської роботи, а також розроблена архітектура гібридної симуляції мають вагоме практичне значення, що виходить за межі теоретичного екологічного моделювання. Ключовим досягненням є експериментальне підтвердження ефективності методики поетапного навчання (Curriculum Learning) для вирішення задач адаптації агентів у стохастичних середовищах. Цей методологічний підхід може бути ефективно імплементований в індустрії інтерактивного програмного забезпечення для створення нового покоління неігрових персонажів (NPC).

На відміну від традиційних систем на базі скінченних автоматів, які вимагають ручного програмування всіх можливих сценаріїв взаємодії (hard-coding), запропоновані агенти на базі навчання з підкріпленням

демонструють здатність до емерджентної поведінки та ситуативної імпровізації. Впровадження таких алгоритмів дозволяє суттєво знизити витрати на розробку та налагодження скриптів поведінки, одночасно підвищуючи рівень варіативності та імерсивності віртуальних середовищ.

Окрім розважальної індустрії, розроблена система має значний потенціал застосування у сфері робототехніки, зокрема при вирішенні задач автономної навігації мобільних роботів та безпілотних літальних апаратів. Використане середовище Unity ML-Agents реалізує концепцію Sim2Real – навчання нейронних мереж у безпечному віртуальному просторі з подальшим перенесенням набутих навичок на фізичні пристрої. Алгоритми уникнення динамічних перешкод та пошуку цільових об'єктів, що були відпрацьовані на моделі «хижак–жертва», можуть бути адаптовані для управління логістичними роботами у неструктурованих середовищах складів або для моніторингових дронів, де критично важливою є здатність приймати рішення в умовах невизначеності та обмеженого сенсорного сприйняття. Також розроблена модель може використовуватися в освітніх цілях як інтерактивний симулятор для демонстрації принципів популяційної динаміки та еволюційної біології.

Результати дослідження апробовано та опубліковано у наступних статті та тезах:

1. Бур'янов Д. С., Дібрівний О. А. Аналіз сучасних підходів до моделювання адаптивної поведінки агентів у віртуальних екосистемах // Зв'язок. 2025. № 5 (177). С. 95–100.

2. Бур'янов Д. С., Герцюк М. М. Розробка методу симуляції екосистеми в дикій природі з використанням нейронних мереж і C# в Unity // Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії : зб. тез доп. II Міжнар. наук.-практ. конф., м. Київ, 19–21 груд. 2024 р. Київ : ДУІКТ, 2024. С. 247.

3. Бур'янов Д. С., Дібрівний О. А. Створення адаптивної моделі симуляції екосистеми у Unity з використанням нейронних мереж // Застосування програмного забезпечення в інфокомунікаційних технологіях : зб. тез доп. Всеукр. наук.-техн. конф., м. Київ, 24 квіт. 2025 р. Київ : ДУІКТ, 2025. С. 321

ПЕРЕЛІК ПОСИЛАНЬ

1. Grimm V., Railsback S. F. Individual-based Modeling and Ecology. Princeton : Princeton University Press, 2005. 448 p.

2. The cane toad (Bufo marinus): fact sheet. Department of Climate Change, Energy, the Environment and Water (Australian Government). 2021. URL: https://www.dcceew.gov.au/environment/invasive-species/publications/factsheet-cane-toad-bufo-marinus (дата звернення: 25.11.2025).

3. Dikötter F. Mao's Great Famine: The History of China's Most Devastating Catastrophe, 1958–62. New York : Walker & Co, 2010. 448 p.

4. Millington I. AI for Games. 3rd ed. Boca Raton : CRC Press, 2019. 1030 p.

5. Juliani A. et al. Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. 2020. URL: https://arxiv.org/abs/1809.02627 (дата звернення: 25.11.2025).

6. Schulman J. et al. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347. 2017. URL: https://arxiv.org/abs/1707.06347 (дата звернення: 25.11.2025).

7. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32. 2019. P. 8024–8035.

8. Animal Diversity Web. University of Michigan Museum of Zoology. 2024. URL: https://animaldiversity.org (дата звернення: 25.11.2025).

9. Abar S. et al. Agent Based Modelling and Simulation tools: A review of the state-of-the-art software. Computer Science Review. 2021. Vol. 24. P. 13–33.

10. Mono K., Bley F., Bitsch J. A. Agent-based modeling of predator-prey systems with reinforcement learning. Ecological Modelling. 2021. Vol. 440. URL: https://doi.org/10.1016/j.ecolmodel.2020.109396 (дата звернення: 25.11.2025).

11. An G., Mi Q., Dutta-Moscato J., Vodovotz Y. Agent-based models in translational systems biology. Wiley Interdisciplinary Reviews: Systems Biology and Medicine. 2009. Vol. 1, iss. 2. P. 159–171.

12. Bonabeau E. Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences. 2002. Vol. 99, no. 3. P. 7280–7287.

13. Isla D. Handling complexity in the Halo 2 AI. Game Developers Conference. San Francisco, 2005.

14. The OODA Loop Explained: The Real Story About the Ultimate Model for Decision Making in Competitive Environments. OODA Loop. 2024. URL: https://oodaloop.com/the-ooda-loop-explained-the-real-story-about-the-ultimate-model-for-decision-making-in-competitive-environments/ (дата звернення: 25.11.2025).

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

16. Zhang J. et al. A Survey on Imitation Learning: Algorithms, Recent Developments, and Challenges. IEEE Access. 2022. Vol. 10. P. 110604–110629.

17. Mnih V. et al. Human-level control through deep reinforcement learning. Nature. 2015. Vol. 518. P. 529–533.

18. Haarnoja T. et al. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. International Conference on Machine Learning (ICML). 2018. P. 1861–1870.

19. Van Valen L. A new evolutionary law. Evolutionary Theory. 1973. Vol. 1. P. 1–30.

20. Mech L. D., Boitani L. Wolves: Behavior, Ecology, and Conservation. Chicago : University of Chicago Press, 2003. 448 p.

21. Thulin C.-G. The history of the Mountain hare (Lepus timidus) in Europe. Mammal Review. 2003. Vol. 33. P. 39–42.

22. Li Y. Deep Reinforcement Learning: An Overview. arXiv preprint arXiv:1701.07274. 2017.

23. Quigley M. et al. ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software. 2009. Vol. 3, no. 3.2. P. 5.

24. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations (ICLR). 2015. URL: https://arxiv.org/abs/1412.6980.

25. Brockman G. et al. OpenAI Gym. arXiv preprint arXiv:1606.01540. 2016. URL: https://arxiv.org/abs/1606.01540.

26. Tobin J. et al. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. IROS. 2017. P. 23–30.

27. Kaelbling L. P., Littman M. L., Cassandra A. R. Planning and acting in partially observable stochastic domains. Artificial Intelligence. 1998. Vol. 101, no. 1-2. P. 99–134.

28. Puterman M. L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York : John Wiley & Sons, 2014. 672 p.

29. What Is Reinforcement Learning? MathWorks Documentation. 2024. URL: https://fr.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html (дата звернення: 18.11.2025).

30. Lillicrap T. P. et al. Continuous control with deep reinforcement learning. International Conference on Learning Representations (ICLR). 2016. URL: https://arxiv.org/abs/1509.02971.

31. Ng A. Y., Harada D., Russell S. Policy invariance under reward transformations: Theory and application to reward shaping. ICML. 1999. Vol. 99. P. 278–287.

32. Schulman J. et al. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347. 2017. URL: https://arxiv.org/abs/1707.06347.

33. Konda V. R., Tsitsiklis J. N. Actor-critic algorithms. Advances in Neural Information Processing Systems. 2000. P. 1008–1014.

34. Schulman J. et al. High-dimensional continuous control using generalized advantage estimation. International Conference on Learning Representations (ICLR). 2016.

35. Schulman J. et al. Trust region policy optimization. International Conference on Machine Learning. 2015. P. 1889–1897.

36. Riedmiller M. et al. Learning by playing solving sparse reward tasks from scratch. International Conference on Machine Learning. PMLR, 2018. P. 4344–4353.

37. Mehta R. et al. Curriculum Learning for Reinforcement Learning: A Survey. arXiv preprint arXiv:2101.10382. 2021. URL: https://arxiv.org/abs/2101.10382.

38. Narvekar S. et al. Curriculum learning for reinforcement learning domains: A framework and survey. Journal of Machine Learning Research. 2020. Vol. 21, no. 181. P. 1–50.

39. Yu Y. Towards sample efficient reinforcement learning. IJCAI. 2018. P. 5739–5743.

40. Kirkpatrick J. et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences. 2017. Vol. 114, no. 13. P. 3521–3526.

41. Unity Real-Time Development Platform. Unity Technologies Official Documentation. 2024. URL: https://docs.unity3d.com/2023.2/Documentation/Manual/index.html (дата звернення: 20.11.2025).

42. NVIDIA PhysX SDK 5 Documentation. NVIDIA Corporation. 2023. URL: https://nvidia-omniverse.github.io/physx/physx/5.1.3/index.html (дата звернення: 20.11.2025).

43. Indrasiri K. gRPC: Up and Running. O'Reilly Media, 2020. 250 p.

44. Protocol Buffers Documentation (Version 3.20+). Google Developers. 2024. URL: https://protobuf.dev/ (дата звернення: 21.11.2025).

45. PyTorch Documentation (Release 2.1). PyTorch Foundation. 2024. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 22.11.2025).

46. Unity ML-Agents Toolkit Documentation (Release 21). GitHub Repository. 2024. URL: https://github.com/Unity-Technologies/ml-agents (дата звернення: 22.11.2025).

47. Ray Perception Sensor 3D. Unity ML-Agents Package Documentation. 2023. URL: https://docs.unity3d.com/Packages/com.unity.ml-agents@2.0/manual/Learning-Environment-Design-Agents.html (дата звернення: 22.11.2025).

48. NavMesh Agent. Unity 2023.2 Manual. 2024. URL: https://docs.unity3d.com/2023.2/Documentation/Manual/class-NavMeshAgent.html (дата звернення: 25.11.2025).

49. NumPy Documentation. NumPy v1.26 Manual. 2024. URL: https://numpy.org/doc/stable/ (дата звернення: 25.11.2025).

50. Dubey S. R. et al. Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing. 2022. Vol. 503. P. 92–108.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Метод симуляції екосистеми з використанням нейронних мереж»

Виконав: студент групи ПДМ-63 Данило БУР’ЯНОВ

Керівник: старший викладач кафедри ІПЗ, доктор філософії PhD Данило КОВАЛЕНКО

Київ - 2025

МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення стабільності та адаптивності симуляції екосистеми з використанням нейронних мереж за рахунок поетапного навчання RL-агентів.

Об’єкт дослідження: процес симуляції екосистеми.

Предмет дослідження: моделі та метод симуляції екосистеми з використанням нейронних мереж.

АКТУАЛЬНІСТЬ РОБОТИ

Метод / Підхід	Принцип роботи	Недоліки	Переваги
Скінченні автомати	Жорсткі правила «ЯКЩО-ТО»	- Відсутність адаптації до змін середовища - Передбачуваність поведінки (ілюзія інтелекту) - Неможливість появи нових стратегій	- Простота реалізації - Низькі обчислювальні витрати
Традиційне машинне навчання	Навчання на статичних датасетах	- Неможливо зібрати датасет для всіх життєвих ситуацій	- Висока точність класифікації на відомих даних. - Швидка робота вже навченої моделі.
Імітаційне навчання	Копіювання дій експерта	- Обмеженість знаннями «вчителя» - Неможливість перевершити демонстратора	- Дуже швидке навчання - Агент одразу демонструє «людяну» поведінку
Поетапне навчання	Навчання з підкріпленням	- Ризик нестабільності навчання - Високі вимоги до часу тренування	- Агенти самостійно підлаштовуються під зміни середовища - Можливість одночасного навчання - Виникнення складної поведінки, яку не програмували явно

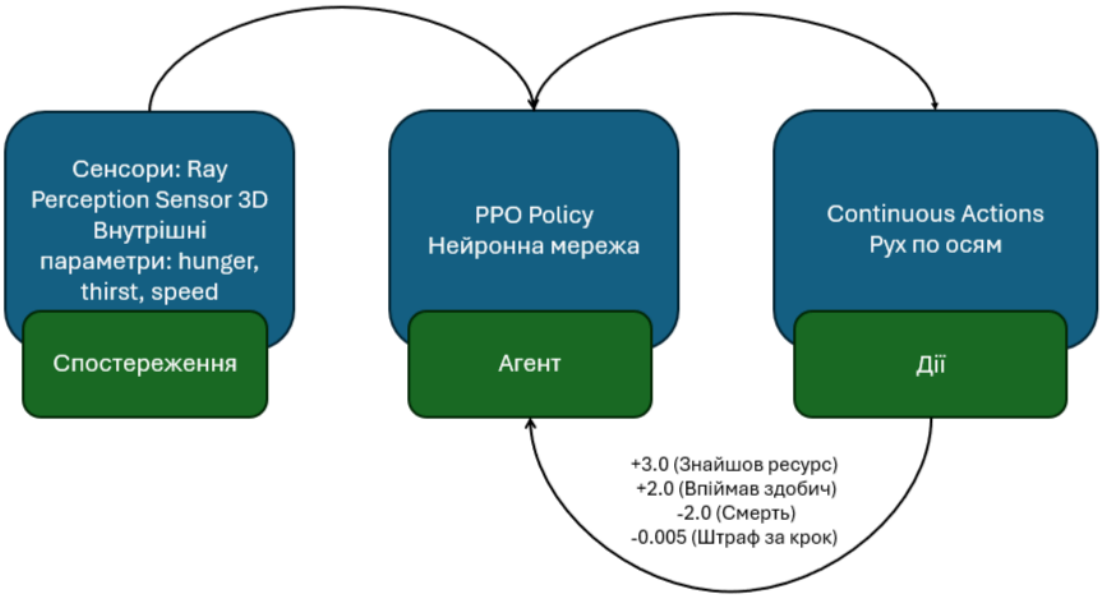
3

ОПИС ХАРАКТЕРИСТИК СИМУЛЯЦІЇ

Параметр	Агент «Заєць»	Агент «Вовк»	Мета налаштування
Макс. швидкість	8 од.	6 од.	Стимулює вовка не просто бігати, а влаштовувати засідки
Голод	Високий (0.1/крок)	Низький (0.05/крок)	Змушує зайця часто ризикувати заради їжі
Спрага	Середній (0.2/крок)	Високий (0.08/крок)	Створює точки перетину інтересів біля води
Винагорода	+3.0 (їжа), -2.0 (смерть)	+2.0 (полювання), -2.0 (голод)	Формування конкурентних цілей

4

ОПИС ЗАДАЧІ НАВЧАННЯ АГЕНТІВ



5

АРХІТЕКТУРА СИСТЕМИ ТА МОДЕЛЬ ПРИЙНЯТТЯ РІШЕНЬ

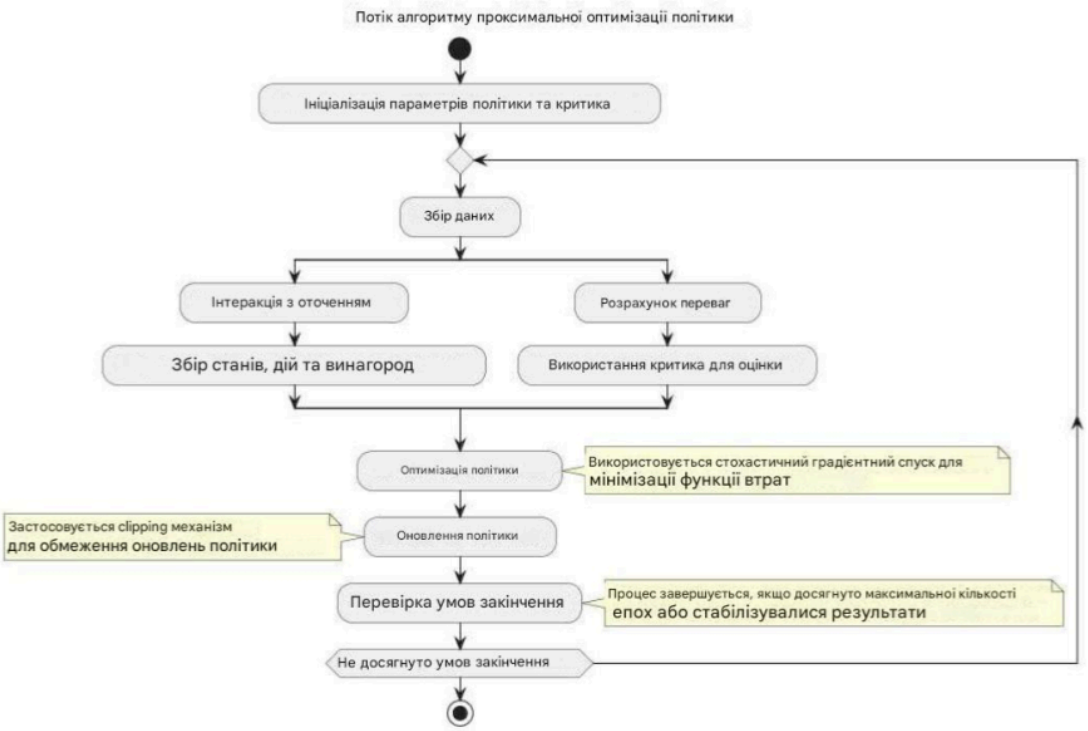
Цільова функція PPO (Clipped Objective):

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

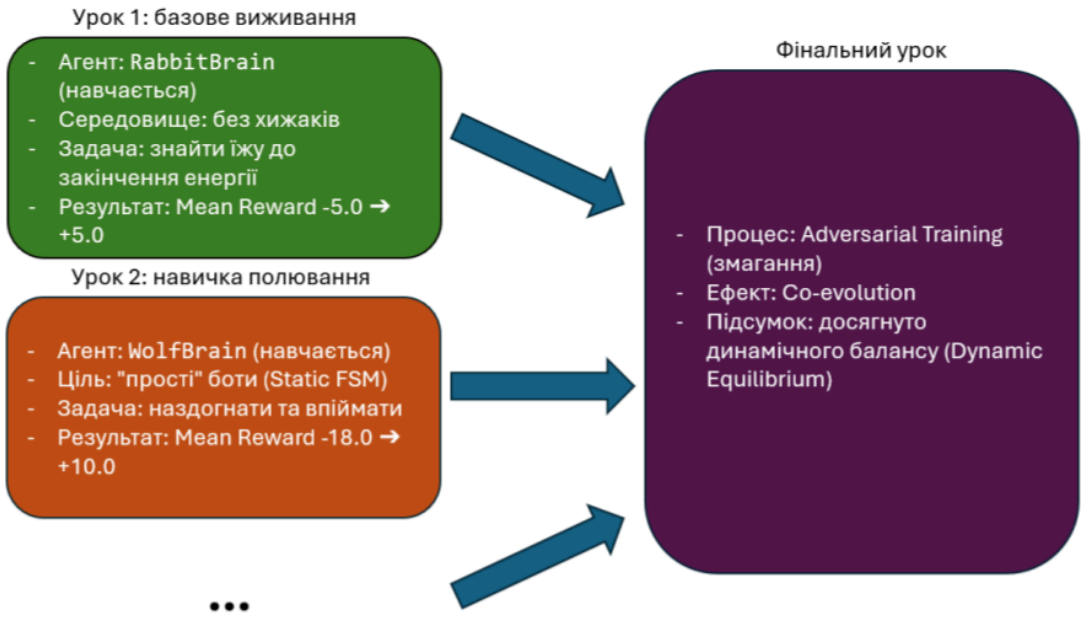
де θ – параметри нейромережі (політика); $r\square(\theta)$ – відношення ймовірностей (нова/стара політика); $\hat{E}\square$ – усереднення за часом (очікуване значення); $\hat{A}\square$ – функція переваги (Advantage); ϵ – параметр обрізання (clipping); clip(...) – функція обрізання (clipping).

6

АЛГОРИТМ PROXIMAL POLICY OPTIMIZATION (PPO)



РОЗРОБЛЕНИЙ МЕТОД ПОЕТАПНОГО НАВЧАННЯ (CURRICULUM LEARNING)



ЕТАПИ ДИНАМІЧНОЇ РАНДОМІЗАЦІЇ СЕРЕДОВИЩА

Проблема	Рішення
Статичне розташування ресурсів	Скрипт генерації валідних зон їжі (ObjectSpawner.cs)
"Перенавчання" (Overfitting), агент не вчиться шукати, а просто запам'ятовує координати	Стохастична генерація позицій у межах зони
Неадаптивна поведінка	Формування навички активного пошуку

9

ПРАКТИЧНА РЕАЛІЗАЦІЯ: ВІЗУАЛІЗАЦІЯ ТА РОБОТА СЕНСОРІВ

Візуалізація променів сенсора (Ray Perception Sensor), агент сканує середовище для виявлення здобичі та перешкод

Реалізація природного ландшафту та взаємодія агентів у фінальній версії симуляції

10

ПОРІВНЯННЯ СТАБІЛЬНОСТІ НАВЧАННЯ

Метрика	Baseline (PPO)	Curriculum Method	Покращення
Середня винагорода	-0.85	+2.50	Якісний стрибок
Виживання (кінець епізоду)	0%	~56%	+56 п.п.
Час до збіжності (кроків)	> 5,000,000	1,200,000	швидше в 4 рази

11

ВИСНОВКИ

1. Проаналізовано існуючі підходи до моделювання екосистем, встановлено, що статичні методи не забезпечують необхідної адаптивності, тому обґрунтовано вибір методу навчання з підкріпленням.
2. Спроектовано та реалізовано гібридну архітектуру симуляції на базі Unity ML-Agents та Python. Вирішено проблему перенавчання шляхом впровадження алгоритму динамічної рандомізації ресурсів.
3. Розроблено методику поетапного навчання. Розподіл процесу на ізольовані уроки дозволив уникнути проблеми «спіралі смерті», яка виникає при прямому навчанні агентів.
4. Експериментально підтверджено ефективність запропонованого методу. У фінальному тесті досягнуто стійкого динамічного балансу популяції: показники середньої винагороди стабілізувалися на позитивному рівні.
5. Отримано автономні моделі поведінки, здатні до адаптації та вироблення стратегій виживання в умовах конкурентного середовища.

12

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Бур'янов Д. С., Дібрівний О. А. Аналіз сучасних підходів до моделювання адаптивної поведінки агентів у віртуальних екосистемах// Зв’язок. №5 (177), 2025. С.95-100.

Тези доповідей:

1. Бур'янов Д. С., Герцюк М. М. Розробка методу симуляції екосистеми в дикій природі з використанням нейронних мереж і C# в Unity. II Міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп’ютерній інженерії». Збірник тез. – К.: ДУІКТ, 19 - 21 грудня 2024, стр. 247
2. Бур'янов Д. С., Дібрівний О. А. Створення адаптивної моделі симуляції екосистеми у Unity з використанням нейронних мереж. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. – К.: ДУІКТ, 24 квітня 2025, стр. 321

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```
using UnityEngine;
using UnityEngine.Events;
using Unity.MLAgents;
using UnityEngine.AI;

public class AnimalStats : MonoBehaviour
{
    [Header("Core Stats")]
    public float maxHealth = 100f;
    public float currentHealth;

    [Header("Needs")]
    public float maxHunger = 100f;
    public float currentHunger = 50f;
    public float hungerRate = 0.1f;

    public float maxThirst = 100f;
    public float currentThirst = 50f;
    public float thirstRate = 0.2f;

    [Header("Thresholds")]
    public float hungerThreshold = 60f;
    public float thirstThreshold = 60f;

    public bool isDead = false;

    public UnityEvent OnDeath;
    private Agent agent;
    private Animator animator;

    void Start()
    {
        currentHealth = maxHealth;
        agent = GetComponent<Agent>();
        animator = GetComponent<Animator>();
    }

    void Update()
    {
        if (isDead) return;

        currentHunger += hungerRate * Time.deltaTime;
        currentThirst += thirstRate * Time.deltaTime;

            currentHunger = Mathf.Clamp(currentHunger, 0,
maxHunger);
              currentThirst = Mathf.Clamp(currentThirst, 0,
maxThirst);

        if (currentHunger >= maxHunger)
        {
            TakeDamage(1f * Time.deltaTime);
        }
        if (currentThirst >= maxThirst)
        {
            TakeDamage(2f * Time.deltaTime);
        }
    }

    public void Eat(float amount)
    {
        currentHunger -= amount;
          Debug.Log(gameObject.name + " ate. Hunger: " +
currentHunger);
    }

    public void Drink(float amount)
    {
        currentThirst -= amount;
         Debug.Log(gameObject.name + " drank. Thirst: " +
currentThirst);
    }

    public void TakeDamage(float amount)
    {
        currentHealth -= amount;
        if (currentHealth <= 0 && !isDead)
        {
            Die();
        }
    }

    void Die()
    {
        isDead = true;
        OnDeath.Invoke();
        Debug.Log(gameObject.name + " died.");

        if (animator != null)
        {
            animator.SetTrigger("Died");
        }

        if (agent != null)
        {
            agent.SetReward(-2.0f);
            agent.EndEpisode();
        }
        else
        {
            if (GetComponent<WolfFSM>() != null)
            {
                GetComponent<WolfFSM>().enabled = false;
            }

            if (GetComponent<RabbitFSM>() != null)
            {
                    GetComponent<RabbitFSM>().enabled =
false;
            }

            if (GetComponent<NavMeshAgent>() != null)
            {
                GetComponent<NavMeshAgent>().isStopped
= true;
            }
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.AI;

public class ObjectSpawner : MonoBehaviour
{
   [Header("Spawn Area")]
   public Collider spawnArea;

   void Start()
   {
      MoveToRandomLocation();
   }

   public void MoveToRandomLocation()
   {
      if (spawnArea == null)
      {
         Debug.LogError("Spawn Area is not set!", this);
         return;
      }

      Bounds bounds = spawnArea.bounds;

      Vector3 randomPoint = new Vector3(
         Random.Range(bounds.min.x, bounds.max.x),
         transform.position.y,
         Random.Range(bounds.min.z, bounds.max.z)
      );

      NavMeshHit hit;
        if (NavMesh.SamplePosition(randomPoint, out hit,
5.0f, NavMesh.AllAreas))
      {
         transform.position = hit.position;
      }
      else
      {
            Debug.LogWarning("Could not find NavMesh
point near " + randomPoint, this);
      }
   }
}
```