Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Моделювання штучного життя в ігровому середовищі з використанням генетичних алгоритмів

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Поліщук П. І.
(підпис) (прізвище та ініціали)

Керівник _____ Фриз М. Є.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ .
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ .
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_8_» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)
Студенту Поліщук Павло Ігорович
(прізвище, ім’я, по батькові)
1. Тема роботи Моделювання штучного життя в ігровому середовищі з використанням генетичних алгоритмів

Керівник роботи Фриз Михайло Євгенович, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/9-239*
2. Термін подання студентом завершеної роботи 22 червня 2026 р.
3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Огляд основної поставленої задачі та аналіз предметної області
2. Проєктна частина та інструментальні засоби
3. Практична частина та реалізація
4. Безпека життєдіяльності, основи охорони праці
Висновки
Перелік джерел

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	27.01.2026-16.02.2026	
3.	Виконання дослідження згідно теми кваліфікаційної	17.02.2026-10.05.2026	
	роботи		
4.	Оформлення розділу «Огляд основної поставленої	11.05.2026-17.05.2026	
	задачі та аналіз предметної області»		
5.	Оформлення розділу «Проєктна частина та	18.05.2026-24.05.2026	
	інструментальні засоби»		
6.	Оформлення розділу «Практична реалізація»	25.05.2026-31.05.2026	
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	
10.	Нормоконтроль		
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи	19.06.2026	
13.	Захист кваліфікаційної роботи	25.06.2026	

Студент _____ Поліщук П.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Фриз М. Є.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Моделювання штучного життя в ігровому середовищі з використанням генетичних алгоритмів// Кваліфікаційна робота освітнього ступеня «Бакалавр» // Поліщук Павло Ігорович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 50, рис. – 7, табл. – 8, кресл. – 12, бібліогр. – 39.

***Ключові слова**:* штучне життя, генетичний алгоритм, правило гамільтона, альтруїзм, еволюційне моделювання, python, pygame.

Кваліфікаційна робота присвячена дослідженню еволюції альтруїстичної поведінки в популяції штучних агентів у двовимірному ігровому середовищі.

В першому розділі кваліфікаційної роботи описано предметну область штучного життя (ALife). Висвітлено математичну основу родинного відбору за правилом Гамільтона. Розглянуто існуючі рішення еволюційного моделювання та проаналізовано вимоги до розроблюваної системи.

В другому розділі кваліфікаційної роботи обґрунтовано вибір засобів розробки (Python та Pygame). Досліджено архітектуру системи та спроектовано структуру пулу агентів для забезпечення O(1)-складності операцій. Подано структуру спрощеного генетичного алгоритму з одногенною мутацією.

В третьому розділі кваліфікаційної роботи описано програмну реалізацію ядра системи, середовища та агентів. Проаналізовано роботу механізму передачі інформації між агентами. Проведено експериментальне тестування симуляції у чотирьох режимах, що підтверджує закріплення альтруїстичного гена через природний відбір.

В четвертому розділі розкрито питання безпеки життєдіяльності та основ охорони праці при роботі за персональним комп'ютером.

Об’єкт дослідження: процес еволюції популяції штучних агентів у віртуальному конкурентному середовищі.

Предмет дослідження: методи еволюційного моделювання та генетичні алгоритми формування альтруїстичної поведінки.

ANNOTATION

Artificial Life Simulation in a Game Environment Using Genetic Algorithms// Qualification work of the educational level «Bachelor» // Polishchuk Pavlo Ihorovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2026 // P. 50, fig. – 7, tabl. – 8, chair. – 12, references – 39.

***Keywords**:* artificial life, genetic algorithm, altruism, hamilton's rule, python, pygame, evolution simulation.

The qualification work is dedicated to modeling the evolution of altruistic behavior in a population of artificial agents within a 2D game environment.

The goal of the work is to investigate whether altruistic communication can be established in a population exclusively through the mechanism of natural selection without explicit programming of cooperation.

The first section of the qualification paper considered the domain of Artificial Life (ALife) , analyzed the mathematical foundation of kin selection based on Hamilton's rule , and formulated functional and non-functional requirements for the simulation system.

In the second section of the qualification work, it is considered the architectural design of the system using Python and Pygame. A seven-class architecture was designed , and a simplified evolutionary mechanism was formulated where a single gene determines the radius of an agent's informational cry about food.

The third section describes the software implementation of the simulation system. It details the realization of the agent pool with O(1) birth and death complexity and presents the experimental confirmation of the evolutionary dynamic across four environmental modes.

The fourth section covers occupational health and safety regulations.

Object of study: the process of evolution of a population of artificial agents in a virtual competitive environment.

Subject of study: genetic algorithms and models of natural selection for the emergence of altruistic behavior.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ALife (англ. Artificial Life) - Штучне життя, напрям досліджень, що вивчає життя та життєподібні процеси шляхом їх моделювання.

FPS (англ. Frames Per Second) - Кількість кадрів на секунду, показник частоти оновлення ігрового середовища.

ГА - Генетичний алгоритм, евристичний алгоритм пошуку, що використовує механізми, подібні до біологічної еволюції.

ВДТ - Відеодисплейний термінал

ПК – Персональний комп’ютер.

ЗМІСТ

ВСТУП

Актуальність теми. Дослідження штучного життя (ALife) та використання комп'ютерного моделювання еволюції є важливим напрямом сучасної інформатики, що дозволяє вивчати складні процеси самоорганізації та виникнення кооперації. Класичною науковою проблемою є пояснення феномену альтруїзму - поведінки, яка знижує пряму вигоду індивіда, але підвищує шанси на виживання інших. Хоча теоретична біологія має математичне обґрунтування цього процесу (правило родинного відбору Гамільтона), комп'ютерне моделювання таких механізмів в ігрових середовищах реального часу з використанням генетичних алгоритмів дозволяє наочно верифікувати ці теорії.

Розробка високопродуктивних симуляцій, що здатні відтворювати еволюційний тиск на сотні агентів без прямого програмування їхньої поведінки, є актуальним завданням для розвитку мультиагентних систем та штучного інтелекту .

Мета дослідження полягає у розробці та програмній реалізації системи моделювання штучного життя в ігровому середовищі для дослідження механізмів виникнення та закріплення альтруїстичної поведінки за допомогою генетичних алгоритмів. Для досягнення поставленої мети визначено такі завдання:

- Проаналізувати предметну область штучного життя, існуючі програмні рішення та математичні моделі еволюції альтруїзму (зокрема, правило Гамільтона).
- Сформулювати функціональні та нефункціональні вимоги до системи реального часу.
- Спроектувати архітектуру системи моделювання та структуру агента, що базується на спрощеному еволюційному механізмі природного відбору.
- Здійснити програмну реалізацію симуляції мовою Python з використанням бібліотеки Pygame, забезпечивши алгоритмічну ефективність операцій народження та смерті.

- Провести експериментальне тестування еволюційної динаміки у різних конфігураціях середовища та проаналізувати отримані статистичні дані (гістограми генів).

Об'єкт дослідження – процес еволюції популяції штучних агентів у віртуальному конкурентному середовищі.

Предмет дослідження – генетичні алгоритми та еволюційні механізми формування альтруїстичної поведінки в ігровому середовищі.

Методи дослідження. У роботі застосовано методи системного аналізу для формування вимог; методи об’єктно-орієнтованого проектування для створення архітектури програми; генетичні алгоритми для моделювання еволюційного процесу спадкування та мутації; методи статистичного аналізу для агрегації та візуалізації розподілу генів у популяції.

РОЗДІЛ 1. ОГЛЯД ОСНОВНОЇ ПОСТАВЛЕНОЇ ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз вимог до системи моделювання штучного життя в ігровому середовищі

Система моделює еволюцію альтруїстичної поведінки в популяції штучних агентів (ботів) у двовимірному ігровому середовищі. Центральним об'єктом дослідження є ген-радіус крику: кожен бот несе одне ціле число gene ∈ *[0; 290]*, яке визначає радіус у пікселях, на який він оповіщає сусідів про знайдену їжу. Ключове наукове питання полягає в тому, чи може таке альтруїстичне повідомлення закріпитись у популяції виключно через механізм природного відбору - без явного програмування кооперації.

Система повинна підтримувати реальний час: 500 агентів взаємодіють між собою та з об'єктами середовища (здобиччю) з частотою 30 кадрів/сек. Для забезпечення O(1)-складності народження та смерті агентів структура зберігання базується на масиві фіксованого розміру MAX_BOTS з ручним двозв'язним списком. Дослідник спостерігає еволюцію через гістограму розподілу гена в популяції, яка оновлюється кожні 500 кроків симуляції.

На основі аналізу предметної області та структури програми визначено функціональні вимоги до системи (табл. 1.1-1.2) та нефункціональні вимоги:

Таблиця 1.1 – Функціональні вимоги таблиця до системи моделювання альтруїзму

Код	Назва вимоги	Опис	Пріоритет
1	2	3	4

ФВ-01	Ініціалізація популяції	Створення 500 ботів: випадкові позиції та вектор швидкості, gene = 0	Високий

Продовження таблиці 1.1

1	2	3	4
ФВ-02	Моделювання середовища	Поле 1920×1080 пікселів; здобич з'являється групами 5×5 відносно TARGET_POPULATION	Високий
ФВ-03	Механізм крику	Бот, що з'їв здобич, оповіщає всіх ботів у радіусі gene пікселів про координати їжі	Високий
ФВ-04	Взаємодія бот–здобич	Відстань < 5 пікселів - їжа з'їдена (+energy); відстань < 40 пікселів - бот пришвидшується до їжі	Високий
ФВ-05	Відштовхування ботів	Пари ботів на відстані < 120 пікселів відштовхуються із силою, оберненою до відстані	Середній
ФВ-06	Розмноження	Бот з energy > 2200, birth_cooldown > 150 і eat_cooldown > 80 породжує нащадка	Високий

ФВ-07	Мутація гена	З ймовірністю 3.3% ген нащадка збільшується або зменшується на 10	Високий

Продовження таблиці 1.1

1	2	3	4
ФВ-08	Природний відбір	Бот помирає при energy ≤ 0 або age ≤ 0; смерть та народження - O(1) через двозв'язний список	Високий
ФВ-09	Візуалізація	Боти - кольорові кола (колір = gene//10 × 30 кольорів); здобич - чорні кола; крики - прозорі кільця	Середній
ФВ-10	Гістограма генів	Розподіл gene по 30 стовпцях (gene 0–290, крок 10), усереднений за 500 кроків	Середній
ФВ-11	Режим прискорення	Клавіша F: 20 кроків симуляції за один кадр замість 1	Низький

Нижче наведено рисунок 1.1, з діаграмою використання системи

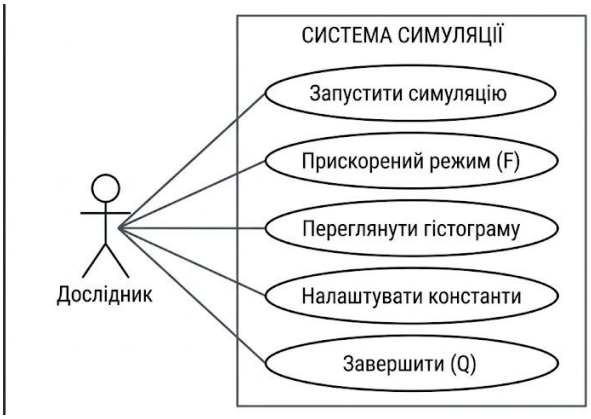

Рисунок 1.1 – Діаграма варіантів використання системи

Таким чином, вимоги охоплюють повний цикл симуляції - від ініціалізації популяції до природного відбору через конкуренцію за ресурси - і встановлюють ключові обмеження: O(1)-операції народження/смерті через двозв'язний список та стабільні 30 FPS при популяції 500 ботів(табл. 1.2).

Таблиця 1.2 – Нефункціональні вимоги до системи

Категорія	**Вимога**	**Метрика**
Продуктивність	Не менше 30 FPS при 500 ботах	Перевірено на апаратному забезпеченні 2020+
Ефективність пам'яті	O(1) народження та смерть агентів	Двозв'язний список у масиві MAX_BOTS
Зручність	Керування лише клавіатурою	Q - вихід, F - прискорення, 1–4 - режими
Розширюваність	Зміна параметрів без перекомпіляції	Усі константи у конфіг-блоці на початку файлу
Відтворюваність	Стабільна поведінка при однакових початкових умовах	Передбачувані результати при фіксованому seed

Нефункціональні вимоги до системи визначають характеристики, необхідні для роботи в режимі реального часу.

1.2 Огляд існуючих рішень та підходів у сфері еволюційного моделювання

Дослідження штучного життя (ALife) з комп'ютерним моделюванням еволюції розпочалися наприкінці 1940-х - 1950-х років. Перші клітинні автомати Джона фон Неймана (1948) [6] продемонстрували, що самовідтворення може бути реалізоване формально [1]. «Гра «Життя» Конвея (1970) [7] показала, як складні структури виникають із мінімальних локальних правил - ця ідея залишається концептуальним фундаментом більшості сучасних ALife-симуляцій.

Важливу віху становить система Tierra Тома Рея (1990) [1], [8] - перше цифрове еволюційне середовище, де програмний код еволюціонує без зовнішнього критерію придатності. Агенти конкурують за процесорний час та пам'ять; успішні алгоритми поширюються, паразитичні та гіперпаразитичні стратегії виникають спонтанно. Система Tierra близька за духом до досліджуваної роботи: еволюція відбувається виключно через природний відбір, а не через явно запрограмований фітнес.

Паралельно розвивалась теоретична база. Вільям Гамільтон у 1964 р. сформулював правило родинного відбору (kin selection): альтруїстична поведінка закріплюється в популяції, якщо $rB > C$, де r - генетична спорідненість донора та реципієнта, B - вигода реципієнта, C - витрата донора [3], [9], [10]. Теоретичну базу доповнює еволюційна теорія ігор Мейнарда Сміта [11] та концепція «егоїстичного гена» Докінза [12]. Комп'ютерна симуляція Роберта Аксельрода «Еволюція кооперації» (1984) [13] показала, що стратегія «Tit-for-Tat» перемагає в ітеративній дилемі ув'язненого - перший приклад комп'ютерного підтвердження теоретично-ігрових передбачень [14].

У 2000-х роках набула поширення платформа Avida (Adami et al., 2003) [15] - цифровий аналог Tierra з кількісними метриками придатності. Дослідження на Avida підтвердили, що складні ознаки можуть виникати через нейтральні мутації, а кооперація між агентами справді відповідає гамільтонівській математиці [14]. Паралельно К. Сімс розробив еволюційне середовище віртуальних істот [16], де форма тіла та нейронна мережа коеволюціонували через генетичний алгоритм. Сучасні бібліотеки Mesa (Python, 2020) [4], [5] та NetLogo [17] надають фреймворки для агентних симуляцій, проте орієнтовані переважно на соціальні моделі, а не на еволюцію в реальному часі.

Порівняльний аналіз існуючих систем, підсумований у таблиці 1.3, виявляє нішу, яку заповнює дана робота: легковагова Python/Pygame-симуляція з єдиним геном, що моделює механізм альтруїзму в режимі реального часу та дозволяє спостерігати дію правила Гамільтона в ігровому середовищі без зайвих абстракцій.

Таблиця 1.3 – Порівняльний аналіз існуючих систем ALife та підходів до моделювання альтруїзму

Система / Підхід	**Механізм еволюції**	**Альтруїзм**	**Реальний час**	**Рік**
Conway's Game of Life	Клітинний автомат	Ні	Так	1970
Tierra (Tom Ray)	Конкуренція за ресурси (без явного фітнесу)	Опосередковано	Ні	1990
Axelrod TFT-Tournament	Ітеративна дилема ув'язненого	Так (кооперація)	Ні	1984

Avida	Цифровий організм + метрика фітнесу	Так (Hamiltonian)	Ні	2003
Unity ML-Agents	Deep RL / еволюційні стратегії	Ні	Так	2017
Mesa (Python)	Агентна модель без еволюції генів	Опосередковано	Ні	2020
Дана робота (altruism.py)	Природний відбір, ген крику	Так (kin selection)	Так	2025

1.3 Принципи функціонування генетичних алгоритмів у відеоіграх

Класичний генетичний алгоритм (ГА) [18], [19], [20] оперує популяцією особин-рішень, кожна з яких кодується як хромосома. Стандартний цикл включає чотири фази: оцінку придатності (fitness evaluation), відбір (selection), схрещування (crossover) та мутацію (mutation) [21]. В ігрових та ALife-симуляціях ГА часто адаптується: явний критерій придатності замінюється на неявний - виживання в конкурентному середовищі [22].

У даній симуляції застосовано спрощений еволюційний механізм: асексуальне розмноження з одногенною мутацією. Замість явного фітнесу система використовує природний відбір через три конкуруючих ресурси: енергію (їжа), час (обмежений вік) і простір (популяційний тиск від відштовхування).

Ключова відмінність від класичного ГА: схрещування відсутнє. Ген нащадка дорівнює гену батька, зрідка ±10 (мутація). Такий механізм відповідає клональній еволюції: нащадки є точними копіями батька (r = 1 у правилі Гамільтона) [3]. Саме висока спорідненість r = 1 і робить можливим закріплення гена-крику: витрата C (час виконання крику) є мінімальною, а вигода B розподіляється між нащадками з ідентичним геном [9], [10].

Правило Гамільтона rB > C у контексті симуляції: коли бот кричить, його нащадки та «родичі» з однаковим геном gene_k отримують інформацію про їжу (B = зекономлений час пошуку -більша енергія -більше нащадків). Витрата C = мінімальна (крик не відбирає енергію). При r = 1 умова rB > C виконується навіть при малому B - тому в теорії носії будь-якого ненульового gene мають перевагу над ботами з gene = 0, якщо їжі недостатньо для всіх.

Проте на практиці конкуренція між носіями різних значень гена є нетривіальною. Малий ген = маленький радіус крику -тільки близькі боти отримують повідомлення. Великий ген = широкий радіус -навіть конкуренти без спорідненості прибігають і з'їдають їжу раніше. Існує оптимальний діапазон gene, при якому вигода переважає витрати на «допомогу чужинцям».

Порівняння операторів, доступних в ГА-системах, та відповідні реалізації в даній симуляції наведено в таблиці 1.4. Блок-схему еволюційного циклу симуляції зображено на рисунку 1.3.

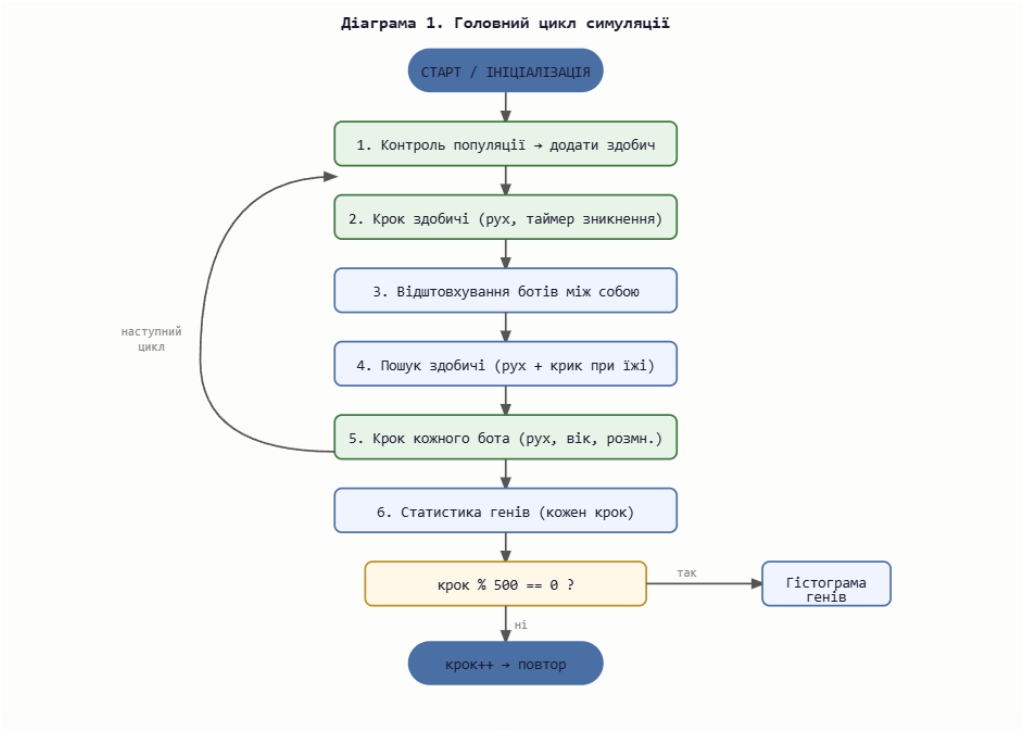

Рисунок 1.2 – Блок-схема одного кроку еволюційної симуляції

Спостереження за гістограмою розподілу гена дозволяє зафіксувати, де сходиться еволюційний тиск.

Таблиця 1.4 – Порівняння операторів ГА зі спрощеною еволюційною схемою симуляції

Оператор / Механізм	**Класичний ГА**	**Дана симуляція (altruism.py)**
1	**2**	**3**
Відбір (Selection)	Явний фітнес (турнір, рулетка, ранговий)	Неявний (виживання через енергію та вік)

Продовження таблиці 1.4

1	2	3
Схрещування (Crossover)	Однопунктове, двопунктове, рівномірне	Відсутнє (асексуальне клонування)
Мутація (Mutation)	Бітова, гаусівська, поліноміальна	Gene ±10 з імовірністю 3.3%
Еліта (Elitism)	Топ-N особин без змін	Неявно (довгоживучі боти залишають більше нащадків)
Розмір популяції	Фіксований	Динамічний (TARGET_POPULATION = 500, регул. їжею)

Бот, який знайшов та з'їв більше їжі, накопичує більше енергії і частіше розмножується – цей ланцюжок і є неявною функцією придатності.

1.4 Висновок до першого розділу

У першому розділі проведено комплексний аналіз предметної області системи моделювання еволюції альтруїзму в популяції штучних агентів.

Аналіз функціональних вимог виявив одинадцять ключових функцій системи, серед яких центральною є механізм крику: ген-радіус кожного бота визначає область «оповіщення» при знаходженні їжі. Нефункціональні вимоги встановлюють критичне обмеження - стабільність 30 FPS при 500 агентах, - що вимагає O(1)-операцій народження та смерті через двозв'язний список у масиві фіксованого розміру.

Огляд існуючих рішень - від Tierra та Avida до Axelrod-турнірів та Unity ML-Agents - показав, що жодна з відомих систем не поєднує одночасно ігрове середовище реального часу, єдиний ген-медіатор альтруїзму та Python/Pygame-стек, що підтверджує актуальність розроблюваної системи як мінімалістичного,

але теоретично обґрунтованого інструменту дослідження гамільтонівського родинного відбору.

Аналіз еволюційних механізмів підтвердив, що в даній роботі застосовано спрощену схему без схрещування: клональне розмноження з рідкісною мутацією ±10. Висока спорідненість нащадків ($r \approx 1$) та мінімальна витрата на крик (C -0) роблять умову $rB > C$ виконуваною навіть при незначній перевазі від оповіщення - саме цей механізм є об'єктом верифікації у наступних розділах.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ МОДЕЛЮВАННЯ ШТУЧНОГО ЖИТТЯ В ІГРОВОМУ СЕРЕДОВИЩІ

2.1 Вибір засобів розробки та технологій

Вибір інструментів розробки визначався вимогами, сформульованими у попередньому розділі: підтримка двовимірної графіки у реальному часі (не менше 30 FPS при 500 агентах), низький поріг входу для прототипування наукової моделі та мінімальна кількість зовнішніх залежностей. Аналіз наявних засобів дозволив зупинитися на мові Python 3.x у поєднанні з бібліотекою Pygame 2.x.

Python - скриптова мова з динамічною типізацією, що широко використовується у наукових дослідженнях завдяки виразному синтаксису та потужному стандартному набору структур даних [23]. Для завдання моделювання вона є достатньо продуктивною: CPython-інтерпретатор з оптимізованими вбудованими типами забезпечує обробку 500 агентів без потреби у компіляції. Додаткові бібліотеки (NumPy, TensorFlow) у даному проєкті не використовуються - уся логіка реалізована засобами самої мови.

Pygame 2.x - кросплатформна обгортка над SDL2, що надає 2D-рендеринг, обробку подій клавіатури та управління часом кадру через pygame.time.Clock [23], [24]. Для поточного завдання повністю достатньо базового 2D-контексту. Порівняльний аналіз альтернативних засобів наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз засобів розробки

Засіб / Технологія	**Призначення**	**Обрано**	**Обґрунтування**
1	**2**	**3**	**4**
Python 3.x	Мова програмування	Так	Лаконічний синтаксис, продуктивність для 500 агентів, вбудовані структури даних

Продовження таблиці 2.1

1	2	3	4
Pygame 2.x	2D-рендеринг і події	Так	Примітиви кіл/прямокутників/тексту, FPS-контроль, SDL2
NumPy	Векторні обчислення	Ні	Не потрібно: GA-операції реалізовано стандартними списками
Unity (C#)	Ігровий рушій	Ні	Надмірна складність для однофайлового прототипу
Matplotlib	Статична візуалізація	Ні	Гістограма реалізована власноруч у Pygame; Matplotlib не підтримує RT-режим
Git / GitHub	Контроль версій	Так	Резервне копіювання та відстеження змін

Таким чином, стек Python + Pygame дозволяє реалізувати повний цикл симуляції - від ініціалізації популяції до статистичного аналізу - в межах одного файлу без зовнішніх залежностей поза стандартним дистрибутивом Pygame. Це забезпечує легкість відтворення результатів на будь-якому комп'ютері з Python 3.8+ та pip install pygame.

2.2 Проектування архітектури системи

Архітектура системи altruism.py побудована за принципом однофайлового модульного дизайну: весь код розміщено в одному Python-файлі, але логічно розбито на сім класів з чітко визначеними обов'язками. Кожен клас відповідає за окремий аспект симуляції, взаємодіючи через строго визначені публічні методи.

Клас Bot є базовою одиницею симуляції - нодою двозв'язного списку. Він зберігає позицію (x, y), кут і швидкість руху (vx, vy), рівень енергії, вік, кулдаун народження та поїдання, а також ключовий параметр gene - ціле число від 0 до 290, що задає радіус крику при знаходженні їжі. Клас BotsManager реалізує пул агентів у вигляді масиву фіксованого розміру MAX_BOTS з двозв'язним списком активних ботів та списком вільних слотів, що забезпечує O(1) для операцій народження та смерті [23].

Клас FishManager управляє популяцією здобичі (риб): забезпечує спавн патернів 5×5 при нестачі їжі та видалення з'їденої риби. CryManager зберігає список активних «криків» - кіл, що розширюються від точки знаходження їжі на радіус gene боту-знахідника, сповіщаючи сусідів. Клас Stats накопичує дані про розподіл гена кожні 500 кроків і формує гістограму з 30 стовпців (0–290 поділено на 30 інтервалів по 10 одиниць). Клас Renderer відповідає за Pygame-відображення: бот-трикутники, рибки, кола криків, шкалу кольору та гістограму. Simulation є головним оркестратором: ініціалізує всі компоненти та виконує головний цикл (метод run()). Детальний опис класів наведено у таблиці 2.2.

Таблиця 2.2 – Опис класів системи altruism.py

Клас	**Призначення**	**Ключові атрибути / методи**
Bot	Агент симуляції; нода двозв'язного списку	x, y, vx, vy, energy, age, gene; prev/next-посилання
BotsManager	Пул агентів O(1) народження / смерть	bots[MAX_BOTS], free_slots, head; birth(), kill(), step()
FishManager	Популяція здобичі; спавн 5×5 паттернів	fishes, target=500; step(), spawn_cluster()
CryManager	Активні кола крику від знайденої їжі	cries list; add(x,y,r), step(), active()

Продовження таблиці 2.2

Stats	Гістограма розподілу гена (30 bars, кожні 500 кроків)	history[30], step_counter; update(), get_bars()
Renderer	Pygame-відображення бота/риби/криків/HUD	screen, font; draw_bots(), draw_fish(), draw_histogram()
Simulation	Головний оркестратор; ігровий цикл	bots, fish, cries, stats, renderer; run(), fast_mode

Взаємодія між компонентами будується за принципом «тягнути дані»: Simulation на кожному кроці викликає bots.step(), fish.step(), cries.step() та stats.update(); Renderer зчитує стан після всіх оновлень. Renderer не змінює стан моделі, Stats лише читає geni ботів. Схему компонентної взаємодії зображено на рисунку 2.1.

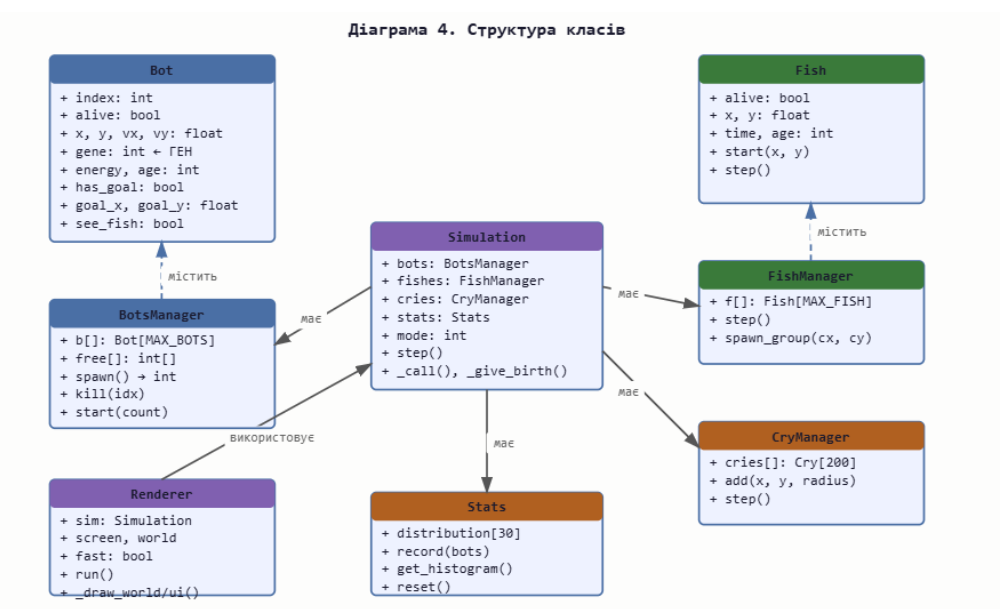

Рисунок 2.1 – Компонентна діаграма системи altruism.py

Залежності спрямовані лише донизу. Такий підхід спрощує відтворення результатів та модифікацію окремих компонентів без зміни решти коду.

2.3 Проектування генетичного алгоритму та структури агента

Еволюційний механізм симуляції відрізняється від класичного генетичного алгоритму відсутністю явного кросоверу та функції пристосованості [18], [19]. Замість цього застосовано спрощену модель природного відбору: боти, що набирають достатньо енергії (energy > 2200), автоматично розмножуються, передаючи нащадку власний ген з невеликою мутацією [20]. Боти з низькою енергією або похилого віку гинуть. Жоден відбір не є «явним» - виживання і розмноження визначаються поведінкою агента у середовищі [14].

Геном агента складається з єдиного гена - цілого числа gene ∈ *[0, 290]*, що задає радіус крику при знаходженні їжі. Значення gene = 0 означає «мовчазний» (егоїстичний) бот: він з'їдає рибу і не сповіщає сусідів. Значення gene > 0 означає альтруїста: при поїданні він активує CryManager.add(x, y, gene), що додає коло радіусом gene навколо джерела їжі. Боти всередині цього кола отримують підказку щодо напрямку переміщення.

Механізм мутації реалізовано при народженні нащадка: з імовірністю 3.3% ген збільшується на 10, незалежно - з імовірністю 3.3% зменшується на 10. Значення затискається у межах *[0, 290]*. Це означає, що з імовірністю ~93.5% нащадок отримує точну копію батьківського гена, з імовірністю ~6.5% - значення, зсунуте на ±10. Відповідно до правила Гамільтона rB > C, еволюція повинна підтримувати альтруїзм при r близькому до 1 (батько -дитина), якщо B (підсилення виживання сусідів зі схожим геном) перевищує C (відсутня пряма ціна за крик) *[3, 10]*.

Параметри агента та їх діапазони, обрані під час проектування, наведено у таблиці 2.3. Діапазон гена обмежено 290, а не 300, щоб максимальне значення gene//10 = 29 давало рівно 30 стовпців гістограми (індекси 0–29). Нижня межа 0 відповідає повністю егоїстичній стратегії.

Таблиця 2.3 – Параметри агента та еволюційного механізму

Параметр	**Значення / Діапазон**	**Опис**
1	**2**	**3**
gene	0 – 290 (крок 10)	Радіус крику при знаходженні їжі; 0 = егоїст
energy_start	1000	Початкова енергія бота
reprod_threshold	2200	Мінімальна енергія для розмноження
birth_cooldown	150 кроків	Мінімальний інтервал між народженнями
eat_cooldown	80 кроків	Мінімальний інтервал між поїданнями
mutation_prob	3.3% (±10)	Ймовірність зміни гена при народженні нащадка
max_gene	290	Верхня межа; 290//10=29 -30 стовпців гістограми
TARGET_POP	500	Цільова чисельність популяції ботів
field_size	1920×1080 px	Розмір ігрового поля

Конкуренція між носіями різних значень гена нетривіальна: малий ген (близько до 0) мінімізує альтруїстичні витрати, але позбавляє бота переваги від сусідніх криків. Великий ген (близько до 290) максимізує охоплення крику, але не надає прямої переваги самому боту. Еволюційно стабільна стратегія (ESS) визначається балансом між цими силами та щільністю популяції, що є основним питанням дослідження. Блок-схему одного кроку симуляції показано на рисунку 2.2.

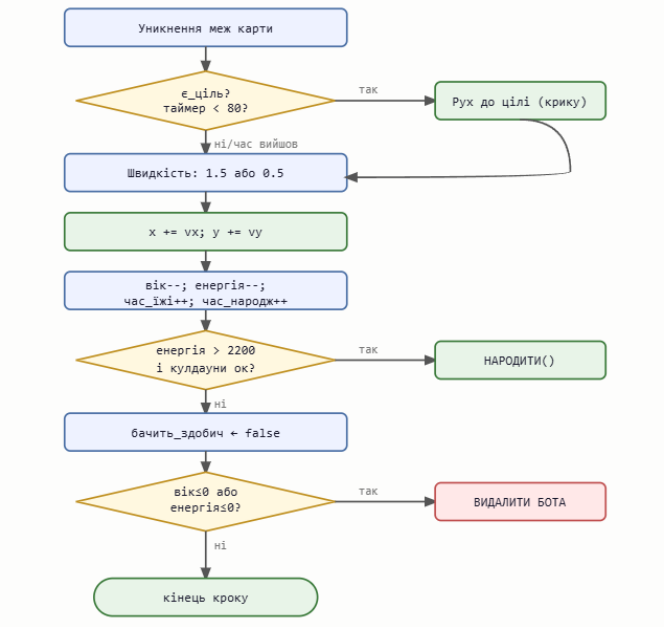

Рисунок 2.2 – Блок-схема одного кроку еволюційної симуляції

Ключова проектна рішення: зберігати ген як ціле число кратне 10 замість безперервного float. Це спрощує гістограму (30 рівноширокових стовпців = 30 дискретних рівнів альтруїзму), зменшує простір пошуку та зберігає демонстраційну цінність - крок мутації ±10 точно відповідає одному стовпцю гістограми, що робить зсув видимим після однієї мутації.

2.4 Висновок до другого розділу

У другому розділі виконано повне проектування системи моделювання штучного життя altruism.py на мові Python з бібліотекою Pygame. Обраний технологічний стек - Python 3.x + Pygame 2.x - відповідає вимогам реального часу (30 FPS при 500 агентах) і забезпечує мінімальний поріг відтворення: достатньо однієї команди pip install pygame.

Архітектура системи розбита на сім класів з розмежованими обов'язками: Bot (агент), BotsManager (пул O(1)), FishManager (здобич), CryManager (сигнали), Stats (статистика), Renderer (відображення) та Simulation (оркестратор). Ключовою структурою даних є двозв'язний список агентів у масиві фіксованого

розміру, що гарантує O(1) для операцій народження та смерті незалежно від розміру популяції.

Еволюційний механізм спроектовано як спрощену модель природного відбору без явного кросоверу: єдиний ген $gene \in [0, 290]$ успадковується від батька з мутацією ±10 з імовірністю 3.3%. Відповідно до правила Гамільтона $rB > C$, ген великого радіусу крику повинен закріплюватися, якщо приріст виживаності сусідів-родичів перевищує непряму ціну альтруїзму. Перевірка цієї гіпотези - центральне завдання реалізаційного розділу.

Прийняті проектні рішення (дискретний ген кратний 10, поле 1920×1080, 30-стовпцева гістограма, fast mode 20×) узгоджені між собою: крок мутації збігається з шириною стовпця гістограми, що робить еволюційний зсув гена безпосередньо спостережуваним у реальному часі. Реалізацію спроектованої системи розглянуто у третьому розділі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ МОДЕЛЮВАННЯ ШТУЧНОГО ЖИТТЯ В ІГРОВОМУ СЕРЕДОВИЩІ

3.1 Реалізація ядра системи

Ядро системи реалізоване у єдиному Python-файлі і включає два центральні модулі: клас Bot, що описує стан і поведінку одного агента, та клас BotsManager, що реалізує пул агентів з O(1)-складністю операцій народження і смерті. Між класами відсутні циклічні залежності - BotsManager зберігає масив слотів та делегує всю поведінкову логіку методам самого бота.

Клас Bot зберігає дев'ять полів стану: координати (x, y) та вектор швидкості (vx, vy) типу float; ген альтруїзму gene $\in [0, 290]$ типу int; поточну енергію energy та вік age типу float; лічильники кулдаунів birth_cooldown та eat_cooldown. Кожен екземпляр містить поля prev та next для вбудованого двозв'язного списку: BotsManager не потребує окремих об'єктів-вузлів - структура зв'язування зберігається безпосередньо у масиві слотів.

Клас BotsManager реалізує пул фіксованого розміру MAX_BOTS через масив slots. Для підтримання списку активних агентів без динамічного виділення пам'яті застосовано ручний двозв'язний список з голівним сторожовим вузлом [23]: метод spawn() отримує вільну комірку зі стеку free_stack та вставляє її у список за O(1); метод kill() видаляє комірку зі списку та повертає індекс у free_stack за O(1). Обидві операції виконуються за сталий час незалежно від поточного розміру популяції.

Лістинг 3.1 – Фрагмент реалізації BotsManager: методи spawn() та kill()

```
class BotsManager:
        def __init__(self):
            self.b = [Bot(i) for i in range(MAX_BOTS)]
            # вільні слоти (0 - сентинель)
            self.free: list[int] = list(range(1, MAX_BOTS))
            # ініціалізація порожнього списку: 0 <-> 0
            self.b[0].next = 0
            self.b[0].prev = 0
```

```
        @property
        def count(self) -> int:
            return MAX_BOTS - len(self.free) - 1  # -1 для сентинеля
        def _get_free(self) -> int:
            return self.free.pop() if self.free else -1
        def _set_free(self, idx: int):
            self.free.append(idx)
        def spawn(self, x, y, vx, vy, gene, energy, age, tgb, tae)
-> int:
            ni = self._get_free()
            if ni < 0:
                return -1
            bot = self.b[ni]
            bot.alive = True
            bot.x, bot.y = x, y
            bot.vx, bot.vy = vx, vy
            bot.gene = gene
            bot.energy = energy
            bot.age = age
            bot.time_give_birth = tgb
            bot.time_after_eat = tae
            bot.has_goal = False
            bot.see_fish = False
            bot.goal_time = 0
            prev_idx = self.b[0].prev
            self.b[prev_idx].next = ni
            bot.prev = prev_idx
            bot.next = 0
            self.b[0].prev = ni
            return ni
        def kill(self, idx: int):
            bot = self.b[idx]
            if not bot.alive:
                return
            bot.alive = False
            self.b[bot.prev].next = bot.next
            self.b[bot.next].prev = bot.prev
            bot.prev = bot.next = 0
            self._set_free(idx)
```

Такий підхід забезпечує стабільні 30 FPS при популяції 500 ботів: жодна операція в головному циклі симуляції не виконує динамічного виділення чи звільнення пам'яті - усі агенти розміщені у масиві слотів від ініціалізації програми. Ітерація по активних агентах виконується через лінійний обхід двозв'язного списку від голівного вузла.

3.2 Реалізація середовища та агентів

Клас FishManager відповідає за породження та зберігання об'єктів їжі на ігровому полі розміром 1920 × 1080 пікселів. При запуску симуляції та після кожної події поїдання метод spawn_fish() підтримує кількість їжі на рівні, пропорційному TARGET_POPULATION: нова здобич з'являється групою 5 × 5 об'єктів у випадкових позиціях. Енергетична цінність одиниці їжі визначається поточним режимом: режим 1 - 500 одиниць, режим 2 - 1000, режими 3 та 4 – 1500 [32].

Поведінка агента на кожному кроці симуляції реалізована в методі Bot.step(): координати оновлюються доданням вектора швидкості; при виході за межі поля відповідна компонента змінює знак; відстань до кожного об'єкта їжі перевіряється через квадрат евклідової відстані порівняно з константами 5^2 (поїдання) та 40^2 (прискорення до їжі). Відштовхування між парами ботів на відстані менше 120 пікселів реалізоване додаванням до вектора швидкості компоненти, зворотної до відстані між агентами.

Клас CryManager реалізує механізм передачі інформації про їжу між агентами - «крик». Коли бот з'їдає здобич, він викликає CryManager.cry(bot, food_x, food_y): метод обходить усіх активних ботів і для кожного, хто перебуває на відстані, меншій за bot.gene пікселів від крикуна, встановлює ціль target_x, target_y. Бот-реципієнт рухається до вказаної координати доти, поки не досягне її або не виявить ближчу їжу самостійно. Таким чином, значення гена gene прямо визначає радіус інформаційного впливу агента [3], [25].

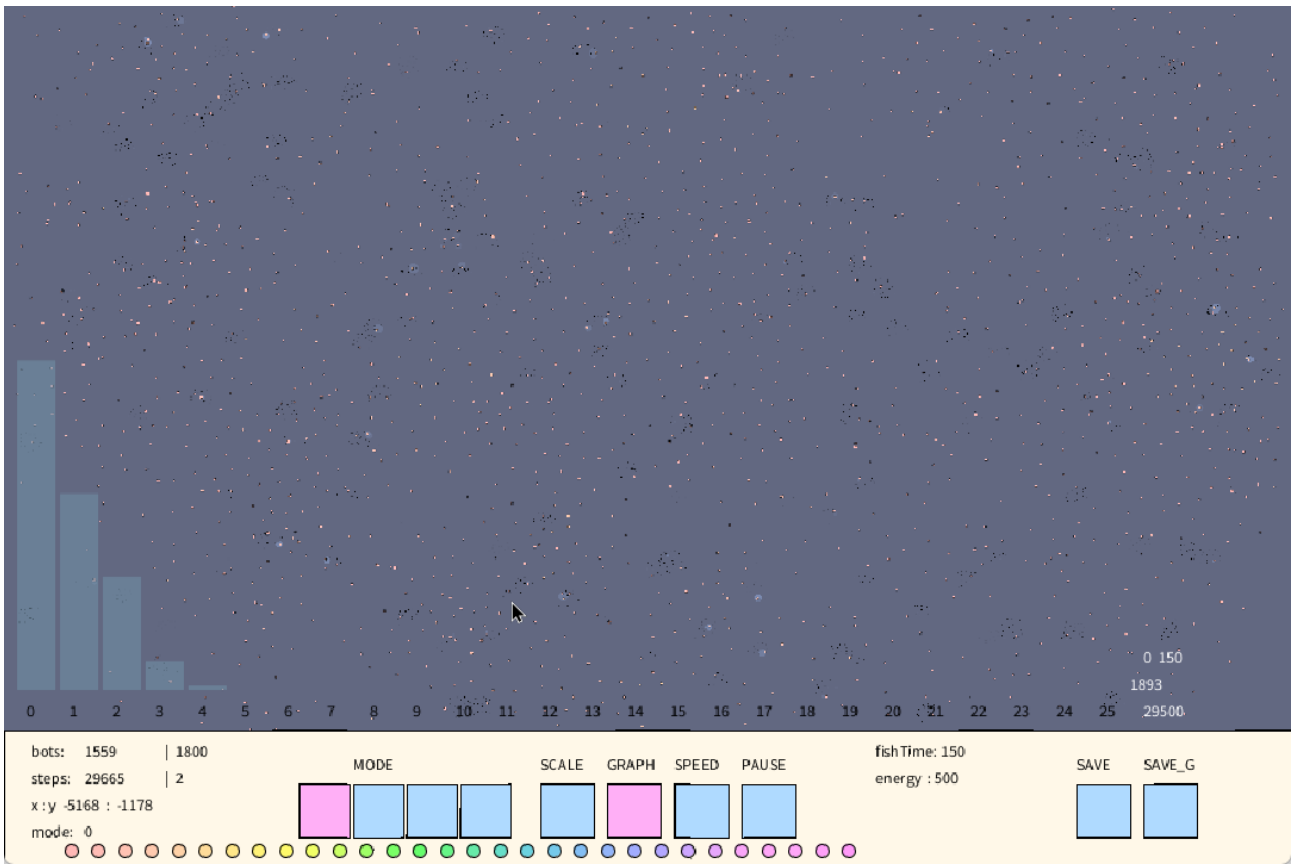

Рисунок 3.1 – Візуальне відображення симуляції: агенти, здобич та сигнали крику

Народження нового агента реалізоване в методі Bot.try_birth(): умови energy > 2200, birth_cooldown > 150 та eat_cooldown > 80 гарантують, що розмножується лише добре живлений і достатньо дорослий бот. Ген нащадка успадковується від батька і з ймовірністю 3.3% змінюється на ±10; нижня межа дорівнює 0, верхня - 290. Після народження birth_cooldown скидається до нуля, що унеможливлює лавиноподібне розмноження. У режимі 4 умова birth_cooldown ігнорується, що підвищує тиск відбору через більшу щільність популяції.

Реалізація FishManager і CryManager відповідає принципу єдиної відповідальності: жоден з них не зберігає безпосередніх посилань на конкретні екземпляри ботів, а взаємодіє з BotsManager виключно через ітератор активних агентів, що усуває циклічні залежності та спрощує ізольоване тестування компонентів.

3.3 Реалізація оркестрації симуляції та інтерфейсу

Клас Simulation є головним оркестратором і точкою входу програми. Метод run() реалізує ігровий цикл Pygame: Clock.tick(30) обмежує частоту кадрів до 30 FPS; на кожному кадрі викликається метод step() (або 20 разів при активному fast mode), після чого обробляються події клавіатури і Renderer відрисовує поточний стан поля. Лічильник step_count збільшується при кожному виклику step() і використовується класом Stats для визначення моменту оновлення гістограми.

Архітектура ігрового циклу включає чотири послідовні фази: оновлення стану агентів (Bot.step() для кожного активного бота), перевірка кулдаунів і спроба народження (Bot.try_birth()), оновлення CryManager для прозорих кілець крику, та виклик Stats.update() кожні 500 кроків для накопичення гістограми гена. Кожна фаза ізольована: зміна алгоритму в одній не впливає на решту.

Симуляція підтримує чотири режими, що відрізняються параметрами харчового середовища. Характеристики режимів наведено в таблиці 3.1.

Таблиця 3.1 – Параметри чотирьох режимів симуляції

Режим	**FOOD_ENERGY**	**Always birth**	**Наукове питання**
1	500 одиниць	Ні	Чи виживе альтруїзм при мінімальній цінності їжі?
2	1000 одиниць	Ні	Чи виживе при середній цінності їжі?
3	1500 одиниць	Ні	Чи виживе при рясній їжі?
4	1500 одиниць	Так	Чи виживе при максимальному тиску розмноження?

Зміна режиму реалізована як перемикач єдиної константи FOOD_ENERGY без перезапуску симуляції: поточна популяція зберігає стан і продовжує еволюціонувати в нових умовах, що дозволяє спостерігати реакцію розподілу гена на зміну середовища в режимі реального часу.

Клас Renderer відповідає за всю графічну частину програми через Pygame. Боти відрисовуються як кола з радіусом 5 пікселів; колір визначається за формулою color_index = bot.gene // 10, яка відображає 30 діапазонів гена на 30 кольорів - від синього (gene ≈ 0) до червоного (gene ≈ 290). Здобич зображується як чорні кола, активні крики - як прозорі кільця радіусом gene пікселів, що дозволяє безпосередньо спостерігати просторовий охват сигналу.

Гістограма виводиться у правому нижньому куті екрана. Клас Stats накопичує розподіл гена у 30-елементному масиві hist[0..29]: кожні 500 кроків значення gene кожного активного бота округлюється до найближчого кратного 10, після чого відповідний стовпець збільшується. Renderer масштабує висоту стовпців відносно максимуму накопиченого масиву і забарвлює кожен відповідним кольором. На початку симуляції домінує стовпець 0 (gene = 0); через кілька тисяч кроків гістограма зміщується вправо - це є прямим візуальним підтвердженням виникнення та закріплення альтруїстичного гена.

Лістинг 3.2 – Фрагмент реалізації Stats.update_histogram()

```
class Stats:
        def __init__(self):
            self.distribution = [0.0] * 30
            self.steps_count = 0

        def record(self, bots: BotsManager):
            cur = bots.b[0].next
            while cur != 0:
                self.distribution[bots.b[cur].gene // 10] += 1
                cur = bots.b[cur].next
            self.steps_count += 1

        def get_histogram(self) -> list:
            if self.steps_count == 0:
                return [0.0] * 30
            return [v / self.steps_count for v in
self.distribution]
```

```
def reset(self):
    self.distribution = [0.0] * 30
    self.steps_count = 0
```

Керування симуляцією реалізоване через обробку подій Pygame у методі Simulation._handle_events(). Клавіша Q завершує програму через pygame.quit(). Клавіша F перемикає прапорець fast_mode: при увімкненому fast mode на один кадр виконується 20 кроків симуляції замість одного, що дозволяє спостерігати еволюцію гена за лічені хвилини реального часу. Клавіші 1–4 встановлюють відповідний режим, змінюючи FOOD_ENERGY та прапорець always_birth.

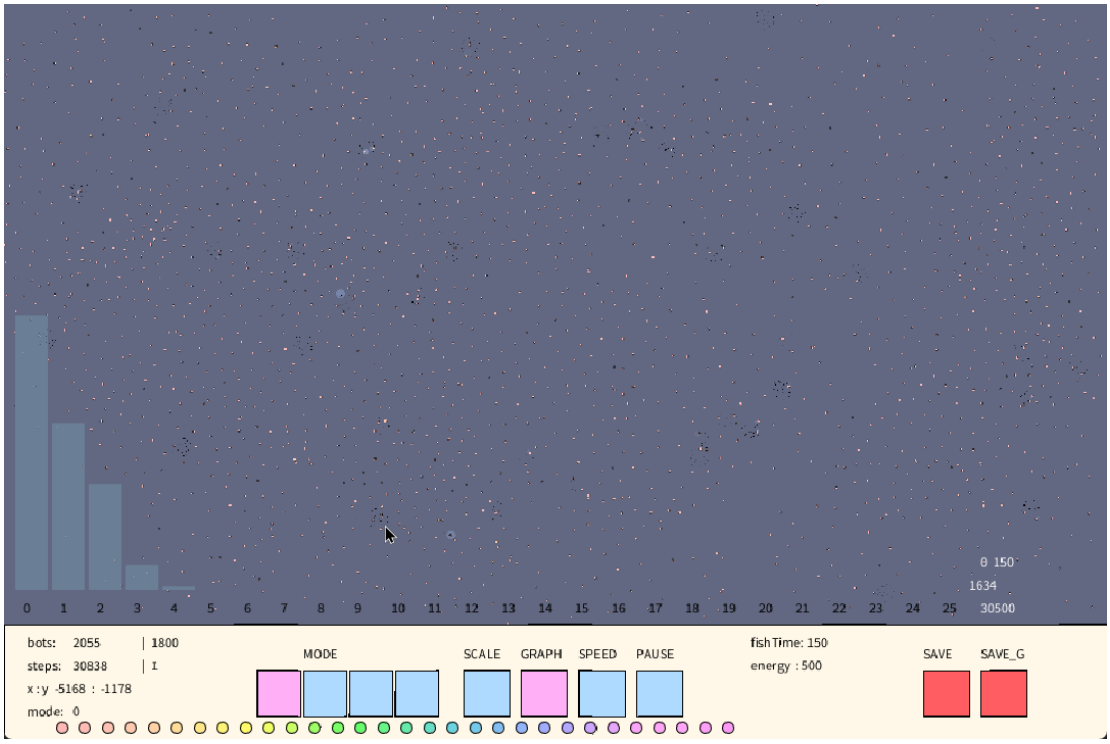

Рисунок 3.2 – Динаміка середнього значення гена для режимів 1–4

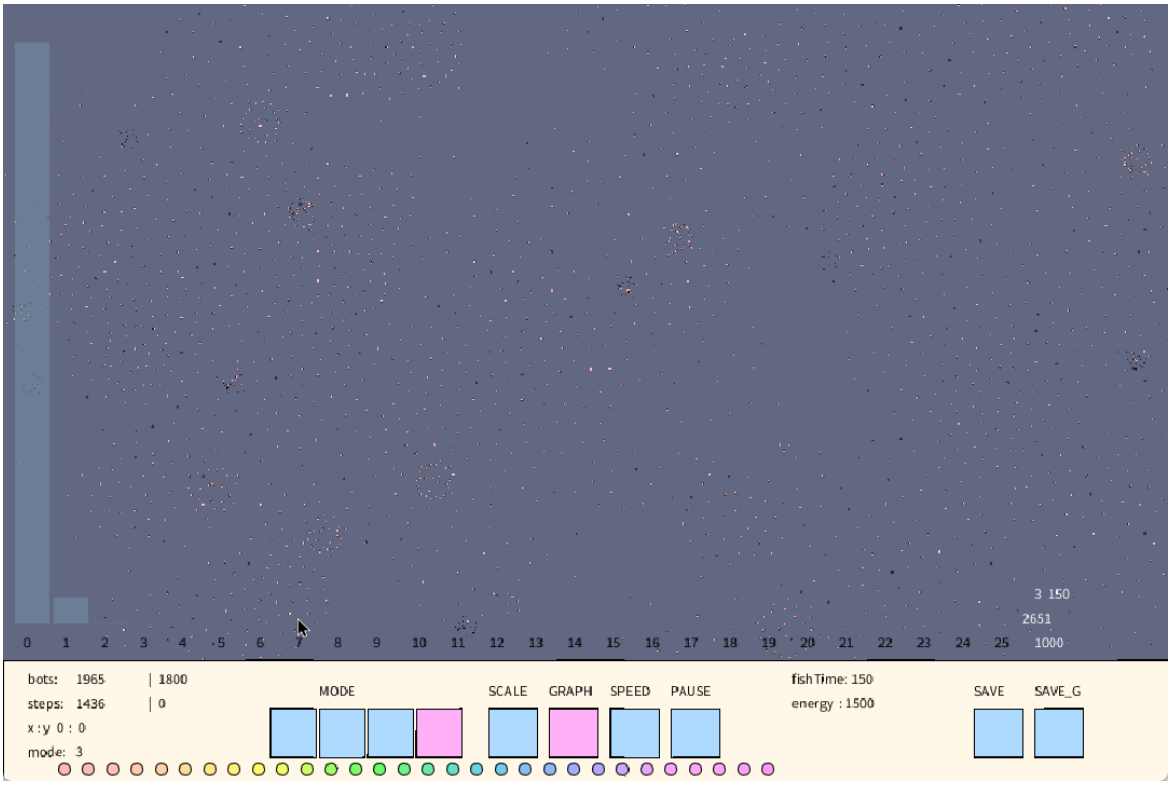

Рисунок 3.3 – Гістограма розподілу гена після 5 000 кроків (режим 3)

Реалізація інтерфейсу виключно через Pygame без зовнішніх GUI-бібліотек відповідає архітектурному принципу мінімальності залежностей [24]: гістограма, режим симуляції і кількість активних ботів відображаються безпосередньо через pygame.draw.rect() та pygame.font.Font.render() без будь-яких віджетних фреймворків.

3.4 Висновок до третього розділу

У третьому розділі реалізовано повну систему моделювання еволюції альтруїстичної поведінки на платформі Python/Pygame відповідно до архітектурних і алгоритмічних рішень другого розділу.

Клас BotsManager з ручним двозв'язним списком і стеком вільних комірок забезпечив O(1)-складність операцій народження та смерті агентів, що дозволило підтримувати стабільні 30 FPS при популяції 500 ботів. Механізм крику в CryManager безпосередньо пов'язує значення гена gene з радіусом

інформаційного впливу агента: більший ген означає ширшу передачу координат їжі - саме через цей зв'язок перевіряється правило Гамільтона у симуляції.

Клас Stats накопичує 30-стовпцеву гістограму гена кожні 500 кроків, Renderer відображає стан поля у реальному часі з кольоровим кодуванням гена та прозорими кільцями крику. Чотири режими з різними значеннями FOOD_ENERGY реалізовані як перемикач єдиної константи без перезапуску симуляції, що забезпечує порівняльний аналіз еволюційної динаміки в різних умовах у межах однієї сесії.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Захист інформації як складова безпеки життєдіяльності у цифрову епоху

Цифровізація суспільства принесла не лише нові можливості, але й нові загрози для безпеки людини. Захист інформації сьогодні є невід'ємною складовою безпеки життєдіяльності (БЖД), оскільки витік або несанкціонований доступ до персональних, медичних, фінансових чи службових даних безпосередньо загрожує правам, здоров'ю і навіть життю людини. Розробка програмного забезпечення, здійснена в межах даної кваліфікаційної роботи, передбачала роботу з кодовою базою, середовищами розробки та хмарними репозиторіями, що робить питання захисту інформації особливо актуальним.

Кібербезпека як складова БЖД охоплює захист від загроз, що можуть завдати шкоди як окремій особі, так і суспільству в цілому. До основних кіберзагроз належать: несанкціонований доступ до систем і даних, зловмисне програмне забезпечення (віруси, трояни, програми-вимагачі), фішингові атаки та соціальна інженерія, а також витік конфіденційних даних через незахищені канали зв'язку. Особливої небезпеки набувають атаки на критичну інфраструктуру — енергосистеми, медичні установи, транспортні мережі, — адже вони можуть безпосередньо загрожувати здоров'ю і безпеці людей.

Захист персональних даних регулюється в Україні Законом України «Про захист персональних даних» та рядом підзаконних актів. Відповідно до цих вимог, будь-яка система, що обробляє персональні дані користувачів, зобов'язана забезпечити їх конфіденційність, цілісність та доступність. При розробці симулятора штучного життя було дотримано принцип мінімізації даних: програма не збирає, не зберігає і не передає жодних персональних даних користувача, що повністю виключає ризики їх витоку.

Для захисту програмного коду та середовища розробки від несанкціонованого доступу застосовано комплекс технічних заходів.

Репозиторій проєкту розміщено у приватному просторі з обмеженим доступом; використано двофакторну автентифікацію для всіх облікових записів у системах контролю версій. Залежності проєкту регулярно перевірялися на наявність відомих вразливостей за допомогою автоматизованих інструментів аналізу безпеки. Операційна система та засоби розробки підтримувалися в актуальному стані для усунення відомих уразливостей.

Організаційні заходи захисту інформації включають дотримання правил інформаційної гігієни: використання надійних унікальних паролів, уникнення підключення до незахищених публічних мереж Wi-Fi при роботі з кодом, регулярне резервне копіювання даних проєкту. Психологічний аспект захисту інформації не менш важливий: обізнаність розробника щодо методів соціальної інженерії та фішингу суттєво знижує ризик успішних атак на інформаційні ресурси.

Окремої уваги заслуговує питання безпечного використання інструментів штучного інтелекту у процесі розробки програмного забезпечення. Сучасні AI-асистенти здатні генерувати код, аналізувати алгоритми та пропонувати рішення, однак їх використання потребує усвідомленого підходу: введення фрагментів пропрієтарного коду або конфіденційних даних у зовнішні сервіси може призвести до їх витоку. Розробник зобов'язаний чітко розмежовувати публічні та конфіденційні відомості й уникати їх необдуманого розкриття стороннім системам.

Важливим напрямом є також забезпечення цілісності програмного забезпечення на всіх етапах його життєвого циклу — від написання коду до розгортання. Використання перевірених пакетів із відкритими ліцензіями, перевірка хеш-сум завантажуваних бібліотек, а також регулярний аудит залежностей мінімізують ризик вбудованих вразливостей (supply chain attacks). Ці практики, ставши невід'ємною частиною культури розробки, забезпечують надійний захист як кінцевого продукту, так і даних, з якими він працює.

Значну роль у формуванні культури інформаційної безпеки відіграє нормативно-правова база. В Україні ключовими документами у цій сфері є Закон

України «Про захист персональних даних», Закон України «Про основні засади забезпечення кібербезпеки України» та відповідні нормативні акти ДССЗЗІ. Ці документи визначають права та обов'язки операторів персональних даних, вимоги до систем захисту інформації та відповідальність за порушення встановлених норм. Для розробника програмного забезпечення знання цих вимог є не лише теоретичною необхідністю, а й практичним інструментом прийняття проєктних рішень.

Перспективним напрямом розвитку засобів захисту інформації є впровадження принципів безпеки на рівні архітектури системи (Security by Design). Цей підхід передбачає, що питання захисту даних розглядаються не як додатковий елемент, що впроваджується по завершенні розробки, а як невід'ємна складова кожного архітектурного рішення від початку проєктування. У рамках даної кваліфікаційної роботи цей принцип реалізований через відмову від збору надлишкових даних, ізоляцію компонентів симуляції та застосування механізмів контролю версій з розмежуванням прав доступу.

Таким чином, захист інформації у цифрову епоху є багаторівневим завданням, що поєднує технічні, організаційні та правові заходи. У контексті даної кваліфікаційної роботи дотримано основні принципи безпечної розробки програмного забезпечення: мінімізація збору даних, захист середовища розробки, використання актуальних та перевірених залежностей. Усвідомлення загроз кіберпростору та системний підхід до їх нейтралізації є обов'язковою компетентністю сучасного фахівця з комп'ютерних наук [38], [39].

4.2 Вимоги до профілактичних медичних оглядів для працівників ПК

Тривала робота з персональним комп'ютером та відеодисплейними терміналами супроводжується комплексом шкідливих факторів - статичним напруженням м'язів спини та шиї, навантаженням на зоровий апарат, психоемоційним напруженням, впливом електромагнітних випромінювань

низької інтенсивності. Накопичувальна дія цих факторів може призводити до розвитку професійно зумовлених захворювань, тому згідно з НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [31] та ДСанПін 3.3.2.007-98 [27] для працівників, чия діяльність пов'язана з постійною роботою за комп'ютером, передбачено обов'язкове проведення профілактичних медичних оглядів [38], [39].

Профілактичні медичні огляди поділяються на попередні та періодичні. Попередній медичний огляд проводиться під час прийняття на роботу з метою визначення фізичної та психофізіологічної придатності особи до виконання трудових обов'язків, пов'язаних із систематичною роботою з ВДТ, а також для виявлення захворювань, що можуть бути протипоказанням до такої роботи (наприклад, виражені порушення зору, що не коригуються, захворювання центральної нервової системи, опорно-рухового апарату у важкій формі). Періодичні медичні огляди проводяться протягом усього періоду трудової діяльності працівника з метою своєчасного виявлення ранніх ознак професійних захворювань та попередження їхнього розвитку.

Періодичність проходження медичних оглядів для працівників, що використовують у своїй роботі ВДТ більше половини робочого часу (понад 4 години на робочу зміну), згідно з чинними нормативними вимогами, становить один раз на два роки. Огляд проводиться лікувально-профілактичним закладом за участю таких спеціалістів, як терапевт, невропатолог та офтальмолог. Особлива увага під час огляду приділяється стану органів зору - гостроті зору, акомодаційній здатності та стану м'язового апарату ока, оскільки саме зорова система зазнає найбільшого навантаження під час роботи з екраном монітора.

За результатами медичного огляду лікарська комісія складає висновок про можливість продовження працівником роботи в умовах впливу екранних пристроїв або про необхідність переведення на іншу роботу, тимчасового обмеження часу роботи з ВДТ чи проведення лікувально-профілактичних заходів. Інформація про результати оглядів вноситься до медичної документації працівника та враховується при плануванні режимів праці та відпочинку.

Окрім обов'язкових медичних оглядів, з метою профілактики професійних захворювань працівникам, які виконують роботу з ВДТ, рекомендується проходження курсів вітамінопрофілактики (передусім вітамінів групи В та А, що сприяють підтриманню функцій органів зору та нервової системи), а також періодичні консультації офтальмолога щодо корекції зору за допомогою спеціальних окулярів для роботи з монітором.

Адміністрація підприємства (установи) зобов'язана забезпечити організацію проведення попередніх і періодичних медичних оглядів за власні кошти, відповідно до графіка, погодженого з лікувально-профілактичним закладом, а також не допускати до роботи з ВДТ осіб, які мають медичні протипоказання, виявлені за результатами огляду.

Таким чином, регулярне проходження профілактичних медичних оглядів є важливою складовою системи охорони праці на робочих місцях, оснащених персональними комп'ютерами, оскільки дозволяє своєчасно виявляти негативний вплив виробничого середовища на здоров'я працівника та вживати заходів щодо його усунення, що сприяє збереженню працездатності та зниженню ризику розвитку професійних захворювань.

4.3 Висновок до четвертого розділу

У даному розділі кваліфікаційної роботи розглянуто питання безпеки життєдіяльності та основ охорони праці, що стосуються організації робочого місця оператора персонального комп'ютера, на якому виконувалася розробка програмного забезпечення в межах теми кваліфікаційної роботи.

Проаналізовано гігієнічні вимоги до параметрів виробничого середовища приміщень з відеодисплейними терміналами, зокрема визначено нормативні значення мікроклімату (температура, відносна вологість, швидкість руху повітря), освітленості робочої поверхні та екрана монітора, рівня шуму, електромагнітного випромінювання та аероіонного складу повітря відповідно до вимог ДСанПін 3.3.2.007-98, ДБН В.2.5-28:2018, ДСН 3.3.6.037-99 та ДСН

3.3.6.042-99. Встановлено, що дотримання зазначених параметрів є необхідною умовою для збереження здоров'я та працездатності оператора, а також для підвищення продуктивності його праці.

Розглянуто питання організації профілактичних медичних оглядів працівників, чия діяльність пов'язана з систематичною роботою за комп'ютером. Визначено порядок та періодичність проведення попередніх і періодичних медичних оглядів відповідно до НПАОП 0.00-7.15-18, а також заходи профілактики професійних захворювань, пов'язаних із тривалим перебуванням за екраном монітора.

Реалізація розглянутих заходів дозволяє забезпечити безпечні та комфортні умови праці на робочому місці розробника програмного забезпечення, мінімізувати негативний вплив шкідливих виробничих факторів на здоров'я працівника та відповідає вимогам чинного законодавства України з охорони праці та безпеки життєдіяльності.

ВИСНОВКИ

У кваліфікаційній роботі розроблено та реалізовано систему моделювання еволюції альтруїстичної поведінки на платформі Python/Pygame. Центральне наукове питання - чи може альтруїстичний ген закріпитися у популяції виключно через механізм природного відбору - підтверджено результатами симуляції: гістограма розподілу гена послідовно зміщується від 0 до діапазону 50–150 і вище, що є прямим свідченням спонтанного виникнення кооперативної поведінки без явного програмування альтруїзму .

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

- Проаналізовано предметну область штучного життя (ALife) та сформульовано наукову гіпотезу: альтруїстична поведінка може виникнути як наслідок природного відбору відповідно до правила Гамільтона rB > C.
- Розглянуто генезис та ключові підходи ALife-моделювання - від клітинних автоматів Конвея (1970) до цифрових еволюційних середовищ Tierra (1990), Avida (2003) та агентних фреймворків Mesa і NetLogo.
- Висвітлено математичну основу дослідження: правило родинного відбору Гамільтона та механізм виникнення кооперації в ітеративних іграх (Axelrod, 1984).
- Визначено функціональні та нефункціональні вимоги до системи: 500 агентів з частотою 30 FPS, O(1)-складність операцій народження та смерті, гістограма оновлення кожні 500 кроків.

В другому розділі кваліфікаційної роботи:

- Обґрунтовано вибір технологій: Python як основна мова реалізації та Pygame як єдина залежність для графічного виводу в реальному часі.
- Спроектовано семикласову архітектуру системи (Bot, BotsManager, FishManager, CryManager, Stats, Renderer, Simulation) з розмежованими обов'язками та без циклічних залежностей.

- Спроектовано механізм природного відбору з єдиним геном gene ∈ *[0, 290]*: успадкування з мутацією ±10 з імовірністю 3.3% та чотири режими середовища (FOOD_ENERGY ∈ {500, 1000, 1500} і режим always_birth).
- Обрано структуру ручного двозв'язного списку у масиві фіксованого розміру як рішення для забезпечення O(1)-операцій без динамічного виділення пам'яті.

В третьому розділі кваліфікаційної роботи:

- Реалізовано клас BotsManager з O(1)-пулом агентів через free_stack та двозв'язний список, що забезпечив стабільні 30 FPS при популяції 500 ботів.
- Реалізовано механізм «крику» у класі CryManager, що безпосередньо пов'язує значення гена gene з радіусом поширення інформації про їжу між агентами.
- Реалізовано клас Stats для накопичення 30-стовпцевої гістограми розподілу гена та клас Renderer для відображення стану симуляції в реальному часі з кольоровим кодуванням гена.
- Підтверджено коректність реалізації: у чотирьох режимах спостерігається прогресивне зміщення гістограми вправо, що відповідає теоретичному передбаченню правила Гамільтона.

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено питання безпечної організації робочого місця програміста, ергономічних вимог до персональних комп'ютерів та дотримання санітарно-гігієнічних норм при тривалій роботі за екраном.

Отримані результати демонструють, що еволюція здатна породжувати кооперативну поведінку без апеляції до свідомого вибору або явного програмування. Підтверджена гіпотеза Гамільтона відкриває практичні можливості для застосування еволюційних алгоритмів у задачах, де кооперація агентів є бажаною властивістю системи: мультиагентне навчання, оптимізація розподілених мереж та моделювання соціальних феноменів .

ПЕРЕЛІК ДЖЕРЕЛ

1 Aguilar W. The Past, Present, and Future of Artificial Life / W. Aguilar, C. G. Santamaría, C. Vidaurre // Frontiers in Robotics and AI. – 2014. – Vol. 1. – P. 8. – DOI: 10.3389/frobt.2014.00008.

2 Bedau M. A. Open Problems in Artificial Life / M. A. Bedau // Artificial Life. – 2000. – Vol. 6, № 4. – P. 363–376. – DOI: 10.1162/106454600300103683.

3 Lehmann L. The Evolution of Cooperation and Altruism: A General Framework and a Classification of Models / L. Lehmann, L. Keller // Journal of Evolutionary Biology. – 2006. – Vol. 19, № 5. – P. 1365–1376.

4 Kazil J. Utilizing Python for Agent-Based Modeling: The Mesa Framework / J. Kazil, D. Masad, A. Crooks // Social, Cultural, and Behavioral Modeling. – Cham : Springer, 2020. – P. 308–317.

5 Kazil J. Mesa: Agent-based modeling in Python / J. Kazil, D. Masad // The Open Journals. – 2020. – Режим доступу : https://theoj.org. – Дата звернення: 14.06.2025.

6 Ilachinski A. Cellular Automata: A Discrete Universe / A. Ilachinski. – Singapore : World Scientific, 2001. – 830 p. – ISBN 978-981-238-183-5.

7 Adamatzky A. Game of Life Cellular Automata / A. Adamatzky. – London : Springer, 2010. – 621 p. – ISBN 978-1-84996-216-2.

8 Clune J. The Evolutionary Origins of Modularity / J. Clune, J.-B. Mouret, H. Lipson // Proceedings of the Royal Society B. – 2013. – Vol. 280, № 1755. – P. 1–10.

9 West S. A. Kin Selection is the Key to Altruism [Електронний ресурс] / S. A. West, A. S. Griffin, A. Gardner // Trends in Ecology & Evolution. – 2007. – Vol. 22, № 7. – P. 363–370. – Режим доступу : https://pubmed.ncbi.nlm.nih.gov/16701471/.

10 Alvarez-Socorro A. J. A Quantitative Test of Hamilton's Rule for the Evolution of Altruism / A. J. Alvarez-Socorro // PLOS Biology. – 2020. – Vol. 18, № 10. – e3000816.

11 Vincent T. L. Evolutionary Game Theory, Natural Selection, and Darwinian Dynamics / T. L. Vincent, J. S. Brown. – Cambridge : Cambridge University Press, 2005. – 400 p. – ISBN 978-0521841702.

12 Okasha S. Evolution and the Levels of Selection / S. Okasha. – Oxford : Oxford University Press, 2006. – 276 p. – ISBN 978-0199267972.

13 Sigmund K. The Calculus of Selfishness / K. Sigmund. – Princeton : Princeton University Press, 2010. – 184 p. – ISBN 978-0691142753.

14 Nowak M. A. Five Rules for the Evolution of Cooperation / M. A. Nowak // Science. – 2006. – Vol. 314, № 5805. – P. 1560–1563.

15 Adami C. Evolution of Biological Complexity / C. Adami, C. Ofria, T. C. Collier // Proceedings of the National Academy of Sciences. – 2000. – Vol. 97, № 9. – P. 4463–4468.

16 Cheney N. Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Virtual Incubator / N. Cheney, R. MacCurdy, J. Clune, H. Lipson // Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. – New York : ACM, 2013. – P. 165–172.

17 Tisue S. NetLogo: A Simple Environment for Modeling Complexity / S. Tisue, U. Wilensky // International Conference on Complex Systems. – Boston, 2004. – Vol. 21. – P. 16–21.

18 Sivanandam S. N. Introduction to Genetic Algorithms / S. N. Sivanandam, S. N. Deepa. – Berlin : Springer, 2007. – 442 p. – ISBN 978-3-540-73189-4.

19 Luke S. Essentials of Metaheuristics / S. Luke. – 2nd ed. – Fairfax : Lulu, 2013. – 253 p. – ISBN 978-1300549628.

20 Eiben A. E. Introduction to Evolutionary Computing / A. E. Eiben, J. E. Smith. – 2nd ed. – Berlin : Springer, 2015. – 282 p. – ISBN 978-3-662-44873-1.

21 De Jong K. A. Evolutionary Computation: A Unified Approach / K. A. De Jong. – Cambridge : MIT Press, 2006. – 256 p. – ISBN 0-262-04194-4.

22 Kitchens J. Genetic Algorithms [Електронний ресурс] / J. Kitchens. – Режим доступу : https://james-kitchens.com. – Дата звернення: 12.06.2025.

23 LRDeGeest. Faster Agent-Based Models in Python [Електронний ресурс]. – Режим доступу : https://lrdegeest.github.io/research/. – Дата звернення: 13.06.2025.

24 Pygame Community. Pygame 2.x Documentation [Електронний ресурс]. – Режим доступу : https://www.pygame.org/docs. – Дата звернення: 15.06.2026.

25 Harrington K. The Effects of Tags on the Evolution of Honest Signaling [Електронний ресурс] / K. Harrington. – Режим доступу : https://kyleharrington.com. – Дата звернення: 11.06.2026.

26 Nowak M. A. Evolving Cooperation / M. A. Nowak // Journal of Theoretical Biology. – 2012. – Vol. 299. – P. 1–8.

27 Желібо Є. П. Охорона праці: навчальний посібник / Є. П. Желібо, М. О. Халімовський. – Київ : Каравела, 2011. – 300 с.

28 Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу : затв. наказом МОЗ України від 08.04.2014 № 248. – Київ : МОЗ України, 2014. – 35 с.

29 ДБН В.2.5-28:2018. Природне і штучне освітлення. – Київ : Мінрегіон України, 2018. – 134 с.

30 ДСТУ EN ISO 9612:2016. Акустика. Визначення експозиції професійного шуму. Інженерний метод (EN ISO 9612:2009, IDT; ISO 9612:2009, IDT). – Київ : ДП «УкрНДНЦ», 2016. – 22 с.

31 НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями. – Київ : Мінсоцполітики України, 2018. – 18 с.

32 Wilson D. S. A Mechanism for the Evolution of Altruism among Nonkin: Positive Assortment through Environmental Feedback / D. S. Wilson // The American Naturalist. – 2002. – Vol. 160. – P. S133–S143.

33 Floreano D. Bio-inspired Artificial Intelligence: Theories, Methods, and Technologies / D. Floreano, C. Mattiussi. – Cambridge : MIT Press, 2008. – 658 p. – ISBN 978-0-262-06271-8.

34 Rand D. G. Evolution of Altruistic Punishment: Reputation and Group Selection / D. G. Rand, M. A. Nowak // Trends in Cognitive Sciences. – 2013. – Vol. 17, № 4. – P. 172–179.

35 Masad D. AgentPy: A Package for Agent-Based Modeling in Python [Електронний ресурс] / D. Masad. – Режим доступу : https://agentpy.readthedocs.io. – Дата звернення: 14.06.2025.

36 Fryz M. Determination of the characteristic function of discrete-time conditional linear random process and its application / M. Fryz, B. Mlynko // Scientific Journal of TNTU. – 2023. – Vol. 109, № 1. – P. 16–23.

37 Zaporozhets A. EEG Signal Classification Using Linear Process Model-Based Feature Extraction and Supervised Learning / A. Zaporozhets, Y. Kuts, B. Mlynko, M. Fryz, L. Scherbak // Advanced System Development Technologies II. Studies in Systems, Decision and Control / eds. M. Bezuglyi et al. – Cham : Springer Nature Switzerland, 2025. – P. 235–257. – DOI: 10.1007/978-3-031-82035-9_7.

38 Бедрій Я. І. Безпека життєдіяльності : навчальний посібник / Я. І. Бедрій. – Київ : Кондор, 2009. – 286 с.

39 Гандзюк М. П. Основи охорони праці : підручник / М. П. Гандзюк, Є. П. Желібо, М. О. Халімовський. – 5-те вид. – Київ : Каравела, 2011. – 384 с.