

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ

НАВЧАННЯ АГЕНТА НА ПРИКЛАДІ ГРИ «ЗМІЙКА»

RESEARCH AND SOFTWARE IMPLEMENTATION OF AGENT

TRAINING METHODS USING THE SNAKE GAME AS AN EXAMPLE

Виконав(ла): студент(ка) 2 курсу, групи КНТ-124м

Спеціальності 121 Інженерія програмного

забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПАВЛЕНКО К.Є.

(ПРИЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ДЬЯЧУК Т.С.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПАВЛЕНКО Кирила Євгенійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження і програмна реалізація методів навчання
Агента на прикладі гри «Змійка».

Research and Software Implementation of Agent Training Methods Using the Snake
Game as an Example

керівник проєкту (роботи) к.т.н., доцент, СТЕПАНЕНКО Олександр
Олексійович

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Реферат. Перелік скорочень та умовних познач. Вступ. 1. Аналіз предметної області. 2. Вибір та обґрунтування структури системи, яка проєктується. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. 6. Експерименти та результати. Висновки. Перелік джерел посилань. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтролер	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5,6
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Кирило ПАВЛЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Олександр СТЕПАНЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
147 с., 17 табл., 36 рис., 3 дод., 45 джерел.

ЗМІЙКА, ГРА, ГЕНЕТИЧНИЙ АЛГОРИТМ, ШТУЧНИЙ ІНТЕЛЕКТ,
НЕЙРОННА МЕРЕЖА, PYTHON, MICROSOFT VISUAL STUDIO.

Об'єкт дослідження – процеси програмної інженерії, зокрема розроблення, тестування та забезпечення якості програмного забезпечення з інтегрованими алгоритмами штучного інтелекту.

Предмет дослідження – методи, алгоритми та технології проєктування, програмування, оптимізації та тестування програмного забезпечення, у якому застосовано штучний інтелект (генетичні алгоритми та нейронні мережі) для управління поведінкою агента у грі «Змійка».

Мета роботи – створення програмного продукту, що реалізує консольну гру «Змійка» із інтегрованими алгоритмами штучного інтелекту, який дозволяє досліджувати ефективність та поведінку цих алгоритмів у контексті програмної інженерії, зокрема для зменшення кількості помилкових дій агента, збільшення тривалості його виживання та оптимізації траєкторії руху під час гри.

Наукова новизна – розроблено модифікований генетичний алгоритм для вибору оптимальних дій агента та поєднано геймінгові механіки з методами машинного навчання для підвищення ефективності навчання агента.

Матеріали, методи та технічні засоби: мова програмування Python, бібліотеки Pygame, Matplotlib, NumPy та multiprocessing, середовище розробки Microsoft Visual Studio.

Результати. Створено програмний продукт, який реалізує класичний геймплей гри «Змійка» та забезпечує візуалізацію процесу навчання

штучного інтелекту, що дозволяє досліджувати ефективність алгоритмів у реальному часі.

Галузь використання – навчальні та дослідницькі цілі у програмній інженерії, зокрема у сфері інтеграції алгоритмів штучного інтелекту в ігрові та навчальні симуляції.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 147 pages, 17 tables, 36 figures, 3 appendixes, 45 sources.

SNAKE, GAME, GENETIC ALGORITHM, ARTIFICIAL INTELLIGENCE, NEURAL NETWORK, PYTHON, MICROSOFT VISUAL STUDIO.

Research Object – the processes of software engineering, in particular the development, testing, and quality assurance of software systems incorporating artificial intelligence algorithms.

Subject of Research – the methods, algorithms, and technologies for designing, programming, optimizing, and testing software that employs artificial intelligence (genetic algorithms and neural networks) to control the behavior of an agent in the «Snake» game.

The aim of the work – to develop a software product that implements the console game “Snake” with integrated artificial intelligence algorithms, enabling the study of the effectiveness and behavior of these algorithms in the context of software engineering, particularly for reducing the number of erroneous agent actions, increasing its survival time, and optimizing its movement trajectory during gameplay.

Scientific novelty – a modified genetic algorithm for selecting optimal agent actions has been developed, and gaming mechanics have been combined with machine learning methods to improve the efficiency of agent training.

Materials, methods, and technical tools: Python programming language; Pygame, Matplotlib, NumPy, and multiprocessing libraries; Microsoft Visual Studio development environment.

The results. A software product has been created that implements the classic gameplay of the «Snake» game and provides visualization of the artificial

intelligence training process, enabling real-time evaluation of algorithm performance.

Area of application – educational and research purposes in software engineering, particularly in the integration of artificial intelligence algorithms into gaming and training simulations.

ЗМІСТ

	С.
Перелік умовних позначень, символів, одиниць і термінів.....	11
Вступ.....	12
1 Аналіз предметної області та огляд існуючих аналогів	13
1.1 Огляд існуючих методів вирішення завдання	15
1.1.1 Гібридний метод.....	15
1.1.2 Глибоке навчання з підкріпленням	16
1.1.3 Метод Монте-Карло	18
1.1.4 Евристичні алгоритми	20
1.1.5 Q-Learning.....	21
1.1.6 Порівняльна таблиця існуючих методів вирішення завдання	21
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання	23
1.2.1 Середовище для розробки та тестування алгоритмів «OpenAI Gym»	23
1.2.2 Метод навчання «Deep Q-Learning»	27
1.2.3 Метод оптимізації «Neuroevolution».....	29
1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів.....	31
1.3 Постановка завдання роботи.....	33
1.4 Висновки за розділом 1.....	33
2 Вибір та обґрунтування структури системи, яка проектується.....	35
2.1 Вибір інструментарію розробки програмного забезпечення	35
2.1.1 Огляд особливостей мови програмування	35
2.1.2 Огляд особливостей обраного середовища розробки	46
2.1.3 Огляд бібліотек Python, що будуть використовуватися	51
2.2 Структурна схема програми	54
2.3 Опис модулів розроблюваної системи.....	56
2.4 Визначення функціональних вимог до програми.....	56

	9
2.5 Висновки за розділом 2	57
3 Основні рішення щодо реалізації компонентів системи.....	59
3.1 Алгоритм функціонування розробленої програми.....	59
3.1.1 Основний алгоритм.....	61
3.1.2 Генетичний алгоритм	62
3.1.3 Алгоритм створення нового покоління	63
3.1.4 Алгоритм мутації	65
3.1.5 Алгоритм схрещування	70
3.1.6 Алгоритм симуляції гри	73
3.1.7 Алгоритм візуальної симуляції гри.....	74
3.1.8 Алгоритм прийняття рішення.....	75
3.2 Опис архітектури створеної моделі нейронної мережі.....	76
3.3 Опис модулів розроблюваної системи.....	77
3.3.1 Клас Individual.....	79
3.3.2 Клас SnakeGameSimulation	80
3.3.3 Клас SnakeGameVisualSimulation.....	81
3.3.4 Клас GeneticAlgorithm	83
3.3.5 Клас GameEnvironment.....	85
3.4 Процес навчання	86
3.4.1 Параметри, які впливають на навчання.....	86
3.4.2 Результати навчання	91
3.5 Основні рішення щодо розробки інтерфейсу.....	92
3.6 Висновки за розділом 3	95
4 Керівництво програміста.....	96
4.1 Призначення та умови застосування програми	96
4.1.1 Призначення програми.....	96
4.1.2 Функції програми.....	96
4.1.3 Умови застосування програми	97
4.2 Характеристика програми	97
4.2.1 Структура програми	97

4.3 Звертання до програми	98
4.4 Вхідні та вихідні дані.....	98
4.4.1 Вхідні дані	98
4.4.2 Вихідні дані	99
4.5 Повідомлення	99
5 Керівництво оператора	100
5.1 Призначення програми	100
5.2 Умови виконання програми	100
5.2.1 Апаратні вимоги програми	100
5.2.2 Вимоги до операторів.....	100
5.3 Виконання програми.....	101
5.3.1 Запуск програми.....	101
5.3.2 Виконання роботи з програмою	101
5.4 Повідомлення оператора	104
6 Тестування і результати.....	106
6.1 Опис методики проведення тестування.....	106
6.2 Результати використання розробленого програмного забезпечення ...	107
6.3 Результати тестування генетичного алгоритму	109
Висновки	111
Перелік джерел посилення	113
Додаток А Технічне завдання	117
Додаток Б Текст програми	124
Додаток В Слайди презентації.....	137

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

- ГА – генетичний алгоритм;
- ПЗ – програмне забезпечення;
- ПК – персональний комп'ютер;
- ТЗ – технічне завдання;
- AI – Artificial Intelligence;
- DQN – Deep Q-Learning;
- DRL – Deep Reinforcement Learning;
- ML – Machine Learning;
- MCT – Monte Carlo Tree Search.

ВСТУП

Сучасні комп'ютерні технології є важливою складовою життя людини, відкриваючи безліч можливостей для роботи, навчання, розваг і наукових досліджень. Зокрема, розробка алгоритмів, здатних вирішувати складні завдання, є одним із пріоритетів розвитку цифрових систем.

У сучасному світі ігрові середовища все частіше використовуються не лише для розваг, а й як платформа для тестування новітніх підходів і технологій. Завдяки своїй динамічності та складності вони є ефективним середовищем для дослідження поведінки алгоритмів штучного інтелекту. Одним із таких середовищ є популярна гра «Змійка», яка має прості правила, але водночас передбачає широкий спектр стратегій для досягнення оптимального результату.

Актуальність використання алгоритмів для навчання агентів обумовлена необхідністю автоматизації складних процесів, для яких написання моделі що навчається є більш швидким, ефективним і результативним. Реалізація навчання через дослідження гри «Змійка» допомагає глибше зрозуміти принципи роботи еволюційних та інших методів оптимізації.

Завданням дипломної роботи є дослідження можливостей застосування алгоритмів для створення ефективної стратегії в ігровому середовищі. У процесі виконання будуть враховані вимоги до адаптивності, точності та ефективності розробленого алгоритму, що дозволить отримати результати, які можуть бути корисними для подальшого розвитку штучного інтелекту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ

Сучасний розвиток інформаційних технологій створює нові можливості для автоматизації процесів, аналізу даних та розробки оптимізаційних алгоритмів. У багатьох сферах, включаючи ігрову індустрію, дедалі більшого поширення набувають підходи, які базуються на використанні штучного інтелекту та еволюційних алгоритмів. Їх застосування дозволяє створювати адаптивні системи, здатні самостійно навчатися та приймати ефективні рішення.

Предметною областю цього дослідження є ігрові середовища, зокрема гра «Змійка». Це популярна гра, яка поєднує прості правила з великим простором для стратегічних рішень. Особливість цієї гри полягає у потребі виваженого управління ресурсами та розробки ефективних рішень в умовах динамічно змінного середовища.

Основною проблемою, яка розглядається у цьому дослідженні, є відсутність універсальних підходів до автоматизації процесу навчання агентів у складних ігрових середовищах. Існуючі методи часто демонструють обмежену адаптивність або вимагають значних обчислювальних ресурсів, що ускладнює їхнє застосування на практиці.

Метою роботи є розробка та дослідження генетичного алгоритму, який дозволяє автоматизувати навчання стратегії гри «Змійка», забезпечуючи ефективність, адаптивність та оптимальність прийнятих рішень.

Для досягнення цієї мети спершу треба розглянути основні поняття, пов'язані з предметною областю. Зокрема, розглянемо, що таке генетичні алгоритми, ігрові середовища та оптимізація.

Генетичні алгоритми (Genetic Algorithms) — це метод оптимізації та пошуку, що імітує еволюційні процеси у природі. В основі цього підходу лежить використання операцій схрещування, мутації та добору для вдосконалення набору рішень [1].

Ігрові середовища (Game Environments) – це віртуальні простори, у яких агенти (гравці або програми) взаємодіють із певними правилами та умовами. Вони використовуються як платформи для навчання, тестування та демонстрації алгоритмів штучного інтелекту [2].

Оптимізація (Optimization) – це процес пошуку найкращого рішення серед можливих варіантів з урахуванням певних обмежень. У контексті генетичних алгоритмів оптимізація базується на ітеративному вдосконаленні рішень [3].

Штучний інтелект (Artificial Intelligence) – це галузь комп'ютерних наук, яка займається створенням систем, здатних виконувати завдання, що зазвичай вимагають людського інтелекту, таких як розпізнавання образів, прийняття рішень та навчання [4].

Еволюційні алгоритми (Evolutionary Algorithms) – це клас обчислювальних методів, заснованих на моделюванні природних еволюційних процесів, таких як відбір, мутація, схрещування та спадковість. Вони використовуються для вирішення складних задач оптимізації [5].

Агент (Agent) – це автономна програма або система, яка здатна сприймати навколишнє середовище за допомогою сенсорів і взаємодіяти з ним через виконавчі механізми. Агент може бути адаптивним і навчатися з досвіду [6].

Навчання з підкріпленням (Reinforcement Learning) – це підхід до машинного навчання, у якому агент навчається шляхом взаємодії з середовищем і отримання винагороди за певні дії. Мета агента – максимізувати сумарну винагороду [7].

Гра «Змійка» (Snake Game) – це класична комп'ютерна гра, у якій гравець керує змійкою, що рухається по ігровому полю, намагаючись зібрати їжу та уникати зіткнень із перешкодами або своїм тілом. Вона використовується як тестовий приклад для розробки алгоритмів оптимізації та навчання [2].

Далі розглянемо існуючі аналоги, які реалізують автоматизацію процесу навчання у подібних ігрових середовищах, та проаналізуємо їхні переваги й недоліки.

1.1 Огляд існуючих методів вирішення завдання

Існує кілька методів для вирішення завдань, пов'язаних з пошуком оптимальних стратегій у складних ігрових середовищах, таких як гра «Змійка». Ці методи базуються на різних підходах до машинного навчання, еволюційних алгоритмів та чисельних методів, кожен з яких має свої переваги та обмеження. Для вирішення задачі оптимізації стратегій в ігрових середовищах, таких як «Змійка», використовуються як класичні алгоритми пошуку, так і сучасні методи глибинного навчання. Найбільш популярними є: гібридні методи, навчання з підкріпленням, метод Монте-Карло, евристичні алгоритми та Q-Learning.

1.1.1 Гібридний метод

Гібридний метод поєднує два підходи: генетичні алгоритми (ГА) для еволюції стратегій і нейронні мережі для моделювання складних функцій. Генетичні алгоритми використовуються для пошуку оптимальних гіперпараметрів або архітектур мережі, тоді як нейронна мережа відповідає за навчання та прийняття рішень в складних середовищах.

Генетичні алгоритми зазвичай використовуються для задач оптимізації на основі пошуку, які важко та часозатратно розв'язати іншими загальними алгоритмами. Завдання оптимізації стосуються або максимізації, або мінімізації цільової функції. Метою генетичного алгоритму є знаходження оптимального або близького до оптимального рішення задачі оптимізації [8].

Нейронна мережа – це математична модель, здатна розв'язувати та моделювати складні шаблони даних і проблеми прогнозування. Алгоритми нейронних мереж розробляються шляхом повторення і використання процесів мозку як базової одиниці. Оскільки вона має здатність імітувати функціональність та операції людського мозку, вона може виконувати будь-які завдання, які може виконати людина [8].

Особливості:

- генетичні алгоритми використовуються для еволюції популяції нейронних мереж або для оптимізації їх архітектур;
- цей підхід дозволяє знаходити ефективні стратегії без необхідності точної аналітичної моделі середовища.

Переваги підходу:

- генетичні алгоритми можуть знаходити оптимальні параметри навіть у складних середовищах, де градієнтний спуск може застрягати;
- мережа навчається змінювати стратегію гри залежно від середовища;
- еволюційний підхід забезпечує широкий спектр можливих рішень.

Недоліки:

- потрібна велика кількість ресурсів для оцінки кожної особини в популяції;
- комбінування двох методів вимагає ретельного налаштування;
- швидкість повільніша порівняно з чисто нейронними мережами через тривалий етап еволюції.

1.1.2 Глибоке навчання з підкріпленням

Deep Reinforcement Learning (DRL) – це підхід до машинного навчання, в якому агент вчиться на основі взаємодії з середовищем,

отримуючи винагороду або штраф за свої дії. У DRL використовуються нейронні мережі для апроксимації функції цінності або політики агента.

У DRL агент взаємодіє з середовищем, щоб навчитися приймати оптимальні рішення. Кроки [9]:

- створення агента та налаштування проблеми;
- агент взаємодіє з навколишнім середовищем через дії, що призводить до зміни станів та отримання винагород;
- агент відстежує свій досвід та оновлює свій метод прийняття рішень;
- на основі отриманих даних алгоритми змінюють підхід агента;
- агент знаходить баланс між використанням відомих дій та випробуванням нових;
- агент навчається обирати дії, які забезпечать найбільшу можливу загальну винагороду;
- політика агента стає кращою і стабільною з часом;
- досвідчені агенти можуть застосовувати набуті знання в нових умовах;
- незнайоме середовище використовується для оцінки ефективності агента;
- використання навченого агента в практичних ситуаціях.

Особливості:

- DRL включає використання глибоких нейронних мереж для навчання оптимальних стратегій на основі відгуків середовища;
- поєднує Q-навчання та глибокі нейронні мережі для визначення оптимальної стратегії.

Переваги:

- ефективний у динамічних середовищах;
- здатний навчатися без нагляду (unsupervised learning);
- забезпечує стратегію в реальному часі.

Недоліки:

- складна реалізація;
- труднощі з вибором гіперпараметрів;
- потребує багато ресурсів і часу на навчання.

1.1.3 Метод Монте-Карло

Метод Монте-Карло – це чисельний метод, що використовує випадкові вибірки для оцінки функцій або ймовірностей у складних ситуаціях. В контексті ігрових середовищ, Монте-Карло може використовуватися для моделювання випадкових ітерацій для оцінки стратегій.

Особливості [10]:

- методи Монте-Карло можуть використовуватись для оцінки майбутніх винагород в завданнях, де ймовірності подій або результатів є невизначеними;
- використовується в задачах, де важко отримати аналітичне рішення.

Ось кілька причин, чому метод Монте-Карло з деревом пошуку Monte Carlo Tree Search (MCTS) часто використовується:

- MCTS відзначається у іграх з великими просторами пошуку, складною динамікою та стратегічним прийняттям рішень. Він успішно застосовувався в іграх, таких як го, шахи, сєгі, покер та багатьох інших, досягаючи вражаючих результатів, які часто перевищують людську експертизу. MCTS може ефективно досліджувати та оцінювати різні ходи чи дії, що призводить до сильної гри та прийняття рішень в таких іграх;
- MCTS підходить для ігор чи ситуацій з невідомою або неповною інформацією. Він покладається на статистичне вибіркове тестування і не потребує повного знання стану гри. Це робить MCTS застосовним до областей, де існує невизначеність або неповна інформація, таких як карткові ігри чи реальні сценарії з обмеженими або ненадійними даними;

- MCTS навчається на основі симуляцій, щоб оцінити вартість дій або станів. Через повторювані ітерації MCTS поступово уточнює свої знання та покращує прийняття рішень. Цей аспект навчання робить MCTS адаптивним та здатним до адаптації до змінюваних обставин або еволюціонуючих стратегій;
- MCTS ефективно збалансовує дослідження і використання під час процесу пошуку. Він інтелектуально досліджує невивчені області простору пошуку, одночасно використовуючи перспективні дії на основі наявних знань. Цей баланс між дослідженням та використанням дозволяє MCTS знаходити оптимальний баланс між відкриттям нових можливостей і використанням добре відомих ефективних дій;
- MCTS природно масштабований і може бути ефективно паралелізований. Він може використовувати розподілені обчислювальні ресурси або багатоядерні архітектури для пришвидшення пошуку та обробки більших просторів пошуку. Ця масштабованість робить MCTS придатним для задач, що вимагають значних обчислювальних ресурсів;
- MCTS є відносно незалежним від конкретного домену. Йому не потрібно спеціальних знань чи евристик для роботи. Хоча можна вносити доменно-специфічні вдосконалення для покращення результатів, базовий алгоритм MCTS може бути застосований до широкого кола задач без значних змін.

Переваги:

- ефективний у середовищах із багатьма варіантами;
- легко адаптується до змін у грі;
- Недоліки:
- може бути повільним для великих ігор;
- результати залежать від кількості симуляцій.

1.1.4 Евристичні алгоритми

Евристичні алгоритми використовують стратегії пошуку для знаходження рішень у просторі станів, зазвичай з метою мінімізації часу пошуку. Вони оцінюють поточний стан за допомогою евристичних функцій для прийняття рішень про наступний крок.

Евристичні методи пошуку застосовуються в широкому спектрі сценаріїв вирішення задач, зокрема [11]:

- виявлення найкоротшої відстані від початкової точки до кінцевої точки за допомогою координат або графа;
- розв'язання задачі оптимального розподілу ресурсів, планування або розміщення для досягнення максимальних результатів;
- використання штучного інтелекту в настільних іграх, таких як шахи або го, для надання порад і прийняття рішень на основі стратегії;
- планування місцеположення та руху роботів для керівництва експедиціями та виконання заданих завдань з високою ефективністю;
- завдання обробки мови, що включають алгоритми пошуку, такі як парсинг або семантичний аналіз.

Особливості:

- використовують задані правила та евристики для прийняття рішень;
- використовують евристичні функції для прискорення пошуку оптимальних шляхів чи стратегій.

Переваги:

- простота реалізації;
- не потребує навчання.

Недоліки:

- обмежена здатність до адаптації;
- не оптимальні результати для складних середовищ.

1.1.5 Q-Learning

Q-Learning – це алгоритм навчання з підкріпленням, який застосовується для пошуку оптимальної політики в умовах невизначеності. Агент вчиться вибирати найкращу дію в кожному стані, максимізуючи загальну винагороду.

Q-learning – це алгоритм без моделі, орієнтований на значення та позаполітичний, який використовується для знаходження оптимальної політики для агента в заданому середовищі. Алгоритм визначає найкращу послідовність дій, яку має виконати агент, залежно від його поточного стану [12].

Особливості:

- алгоритм без моделі, де агент отримує нагороди без знання наступного стану;
- оновлення Q-функції базується на принципі відкладу винагороди.

Переваги:

- простота реалізації для малих ігор;
- не потребує нейронної мережі.

Недоліки:

- важко масштабувати для великого простору станів;
- погана продуктивність у складних сценаріях.

1.1.6 Порівняльна таблиця існуючих методів вирішення завдання

У таблиці 1.1 було порівняні описані раніше методи вирішення завдання.

Таблиця 1.1 – Порівняння існуючих методів

Характеристики	ГА та нейронні мережі	DRL	Метод Монте-Карло	Евристичні алгоритми
1	2	3	4	5
Середній бал	4979.87	7650.54	2961.72	25433.31
Середнє відхилення	3886.56	4889.36	3096.10	9853.20
Середня довжина змійки	5.87	8.54	3.87	26.31
Найкращий результат	25122.00	27128.00	21122.00	59160.00

У ході дослідження було розглянуто кілька методів навчання агента: класичний метод Монте-Карло, евристичні алгоритми, метод глибинного навчання з підкріпленням (DRL) та гібридний підхід, що поєднує генетичні алгоритми з нейронними мережами.

Результати показали, що евристичні алгоритми забезпечують найвищі значення середнього балу, однак вони не належать до методів навчання — їхня ефективність зумовлена наявністю заздалегідь заданих правил поведінки.

Метод DRL продемонстрував кращі показники порівняно з базовими стохастичними підходами, але потребував значних обчислювальних ресурсів та точного налаштування параметрів навчання.

У свою чергу, гібридний метод на основі генетичних алгоритмів і нейронних мереж забезпечив високу ефективність при помірних обчислювальних витратах. Завдяки еволюційному відбору стратегій та можливості нейронної мережі узагальнювати інформацію про стан середовища, цей підхід показав стабільні результати навіть у складних і динамічних ігрових умовах.

Метод Q-Learning не був включений до експериментального порівняння через надмірно великий простір станів у грі «Змійка», що робить класичну табличну реалізацію неефективною. Замість нього було використано DRL, який є його сучасним аналогом.

У результаті було обрано гібридний метод, що поєднує генетичні алгоритми та нейронні мережі, оскільки він забезпечує баланс між ефективністю, стабільністю та обчислювальною економічністю, що робить його оптимальним вибором для задачі навчання агента у грі «Змійка».

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

Перш ніж розробляти нову програму, особливо в галузі штучного інтелекту та машинного навчання, важливо провести аналіз існуючих рішень. Це дозволяє зрозуміти поточний стан технологій, виявити їхні сильні та слабкі сторони, а також визначити напрями для вдосконалення. Порівняння аналогів допомагає уникнути повторення вже зробленого та спрямувати зусилля на створення інноваційного та ефективного продукту.

У контексті автоматизації гри «Змійка» існує кілька популярних підходів, які використовуються для розробки штучного інтелекту, що керує агентом. Серед них найбільш відомі: OpenAI Gym Environments, Deep Q-Learning for Snake, Neuroevolution for Snake. Нижче розглянуто ці методи, їх визначення, функції, особливості, переваги та недоліки.

1.2.1 Середовище для розробки та тестування алгоритмів «OpenAI Gym»

OpenAI Gym – це середовище для розробки та тестування алгоритмів навчання з підкріпленням. Воно містить стандартні інтерфейси для взаємодії з різноманітними симуляціями, включаючи гру «Змійка»

OpenAI Gym – це набір інструментів для розробки та порівняння алгоритмів навчання з підкріпленням. Він надає різноманітні середовища для тестування, включаючи класичні ігри, задачі управління та симуляції роботів. Основні функції OpenAI Gym включають стандартизований інтерфейс для взаємодії з різними середовищами, що спрощує процес розробки та тестування алгоритмів. Реалізація базується на мові Python, що забезпечує широку сумісність та легкість інтеграції з іншими бібліотеками машинного навчання [13].

Функції та особливості:

- надання стандартних середовищ для навчання агентів;
- вбудована підтримка різних симуляцій, що дозволяє легко змінювати середовище без значних змін коду;
- можливість інтеграції з популярними бібліотеками для навчання з підкріпленням, такими як TensorFlow або PyTorch.

Основним будівельним блоком OpenAI Gym є клас Env. Це клас на Python, який фактично реалізує симулятор, що запускає середовище, у якому ви хочете навчати свого агента. OpenAI Gym постачається з великою кількістю вбудованих середовищ, наприклад, таких, де потрібно підняти машину на пагорб, утримати маятник у рівновазі, отримати високий рахунок у іграх Atari тощо. Крім того, Gym дозволяє створювати власні середовища [14].

Функції класу Env, які допомагають агентові взаємодіяти із середовищем. Дві ключові функції [14]:

- reset – ця функція скидає середовище до початкового стану та повертає спостереження за середовищем, що відповідає цьому стану;
- step – ця функція приймає дію як вхідний параметр і застосовує її до середовища, внаслідок чого середовище переходить у новий стан.

Функція step повертає чотири результати:

- observation – спостереження за станом середовища;

- reward – винагорода, яку можна отримати після виконання дії, переданої у функцію step;
- done – чи завершено епізод. Якщо значення true, можливо, потрібно завершити симуляцію або скинути середовище для початку нового епізоду;
- info – додаткова інформація, залежно від середовища. Наприклад, кількість залишених життів або інша загальна інформація, яка може бути корисною для налагодження.

Наприклад, середовище під назвою MountainCar, у якому метою є керування автомобілем, щоб піднятися на гору. Автомобіль знаходиться на одновірній доріжці між двома «горами». Завдання полягає у підйомі на гору з правого боку; однак двигун автомобіля недостатньо потужний, щоб подолати підйом за один раз. Тому єдиний спосіб досягти успіху — розганятися, рухаючись вперед-назад, щоб набрати імпульс. На рисунку 1.1 можна побачити інтерфейс для цієї гри [14].

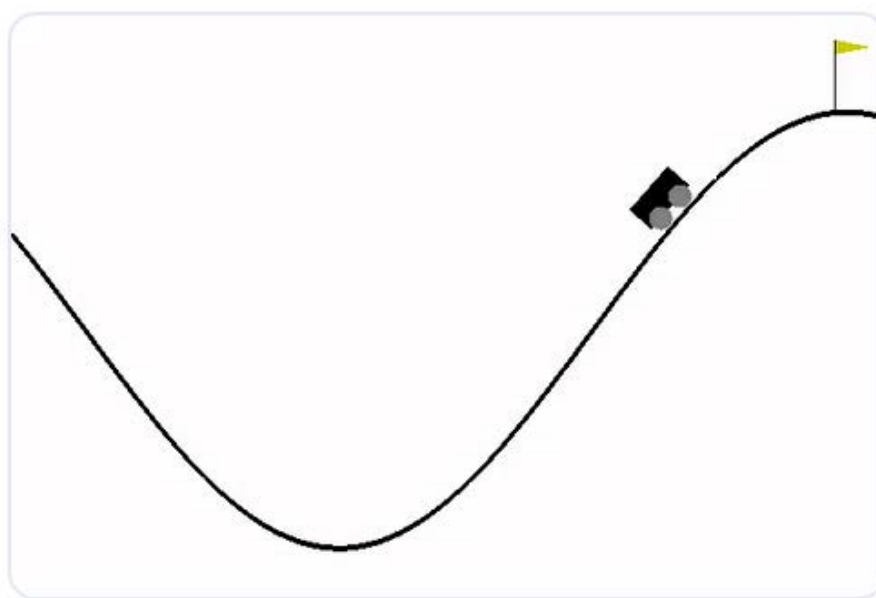


Рисунок 1.1 – Середовище під назвою MountainCar [14]

Також для гри «Змійка» є вже створені програмні рішення. Наприклад: «Gym -snake»

Gym -snake – це мультиагентна реалізація класичної гри «Змійка», створена як середовище OpenAI Gym [15].

На рисунку 1.2 можна побачити інтерфейс цього програмного забезпечення [15].

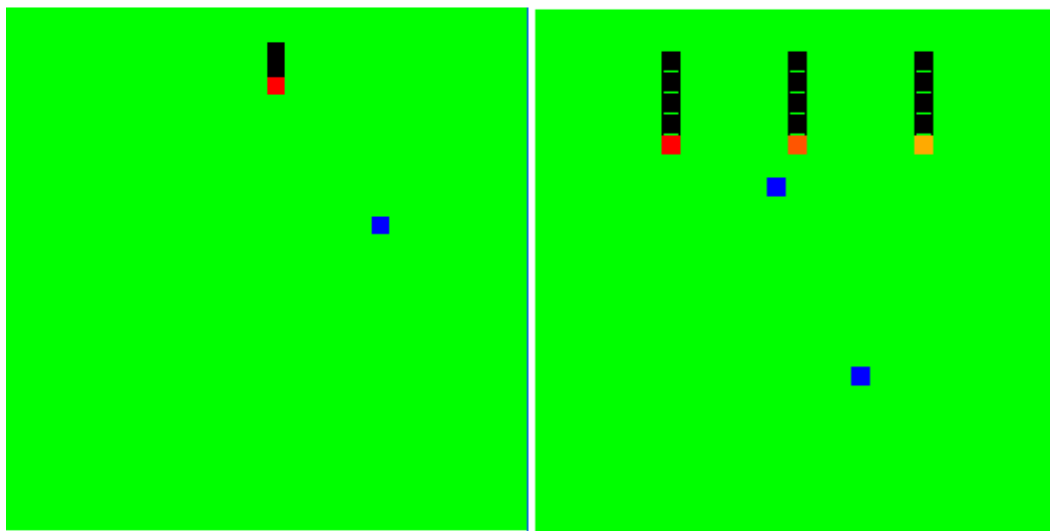


Рисунок 1.2 – Інтерфейс під назвою Gym -snake [15]

Дана програма пропонує два варіанта [15]:

- snake-v0 – це класична версія гри «Змійка»;
- snake-plural-v0 – це версія гри з кількома змійками та декількома елементами їжі на карті.

Переваги OpenAI Gym Environments:

- широкий спектр середовищ для тестування алгоритмів;
- простий та уніфікований інтерфейс;
- активна спільнота та наявність документації.

Недоліки:

- обмежена кількість складних середовищ для специфічних задач;
- потреба в додатковій реалізації для специфічних ігор, таких як «Змійка».

1.2.2 Метод навчання «Deep Q-Learning»

Deep Q-Learning – це метод навчання з підкріпленням, який використовує нейронні мережі для наближення Q-функції (функції цінності дій).

Ця функція оцінює, наскільки вигідним буде виконання певної дії в конкретному стані, з урахуванням майбутніх винагород. Основна ідея DQL полягає у використанні нейронної мережі як заміни таблиці Q-значень, що дозволяє ефективно працювати зі складними та великими просторами станів [16].

Функції та особливості:

- нейронна мережа в DQL використовується для передбачення Q-значень для кожної можливої дії в даному стані. Це дозволяє працювати з великими просторами станів, де неможливо використовувати табличний Q-Learning;
- під час навчання використовується буфер досвіду (Replay Buffer), у якому зберігаються попередні взаємодії агента з середовищем. Замість того, щоб навчатися на нових даних у кожній ітерації, агент випадковим чином вибирає досвід із буфера, що знижує кореляцію між даними та сприяє стабільному навчанню;
- В DQL використовуються дві нейронні мережі: основна мережа (Online Network), яка оновлюється під час навчання та цільова мережа (Target Network), параметри якої періодично копіюються з основної;
- функція винагороди визначає цілі агента;
- епсілон-жадібна стратегія використовується для балансу між дослідженням середовища (exploration) і використанням найкращих наявних стратегій (exploitation). На початку навчання агент випадковим чином виконує дії (дослідження), але поступово зменшує частоту випадкових дій у міру покращення стратегії.

У контексті гри «Змійка» цей метод застосовується для навчання агента грати в гру шляхом максимізації отриманих винагород. Реалізація включає створення середовища гри, визначення функції винагороди та побудову нейронної мережі, яка навчається приймати оптимальні рішення на основі стану гри. Приклад такої реалізації – це SnakeQ. На рисунку 1.3 можна побачити вигляд цієї гри за допомогою реалізації DQL [17]

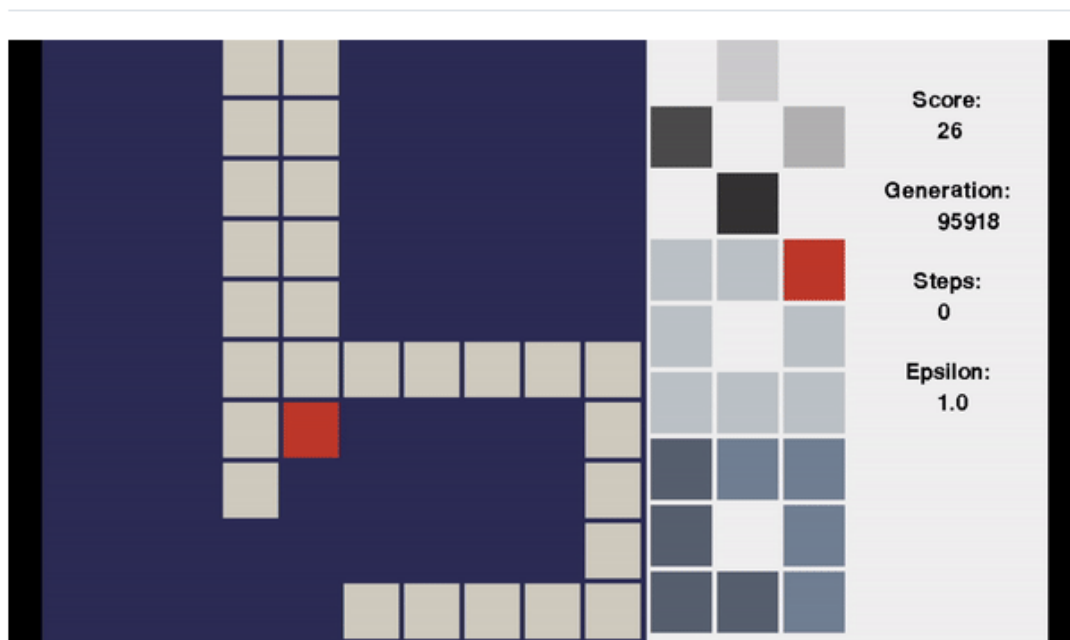


Рисунок 1.3 – Програмне рішення «SnakeQ» [17]

Переваги:

- можливість навчання агента без попередніх знань про гру;
- адаптивність до змін у середовищі.

Недоліки:

- високі обчислювальні витрати на навчання;
- можливість нестабільності під час навчання, що може вимагати налаштування гіперпараметрів.

1.2.3 Метод оптимізації «Neuroevolution»

Neuroevolution – це метод оптимізації архітектури та ваг нейронної мережі за допомогою еволюційних алгоритмів.

Neuroevolution – це метод оптимізації штучних нейронних мереж за допомогою еволюційних алгоритмів. Замість традиційного навчання з градієнтним спуском, нейронні мережі вдосконалюються через процеси, натхненні біологічною еволюцією: відбір, мутація та кросовер. Neuroevolution використовується для вирішення завдань, де обчислення градієнтів важке або неможливе, а також для відкриття інноваційних структур нейронних мереж [18].

Створюється початкова популяція агентів з випадковими нейронними мережами. Потім агенти оцінюються за функцією придатності (наприклад, довжина змійки). Найкращі агенти «розмножуються», утворюючи нові покоління.

Основні функції та особливості

- у Neuroevolution немає потреби обчислювати похідні, що дозволяє застосовувати цей метод у середовищах із дискретними діями або шумними функціями винагороди;
- крім ваг, Neuroevolution може оптимізувати топологію мереж, додаючи або видаляючи нейрони та з'єднання. Ця властивість відома як NEAT (NeuroEvolution of Augmenting Topologies);
- процес еволюції. Спочатку генерується початкова група нейронних мереж із випадковими параметрами. Потім кожна мережа тестується у середовищі, і її ефективність вимірюється за допомогою функції винагороди. Далі найкращі мережі вибираються для створення нового покоління. Після чого до мереж застосовуються випадкові зміни (мутації) параметрів або структури. І далі відбувається комбінація параметрів двох мереж для створення нового потомства;

– оскільки кожна мережа незалежно тестується в середовищі, Neuroevolution легко масштабується на паралельні обчислення.

У грі «Змійка» цей підхід використовується для еволюційного навчання агента шляхом мутацій та відбору найкращих стратегій гри. Реалізація включає створення популяції нейронних мереж, оцінку їхньої ефективності в грі та застосування еволюційних операторів для покращення продуктивності. Прикладом такої є реалізації є «Snake-neuroevolution», яку можна побачити на рисунку 1.4 [19].

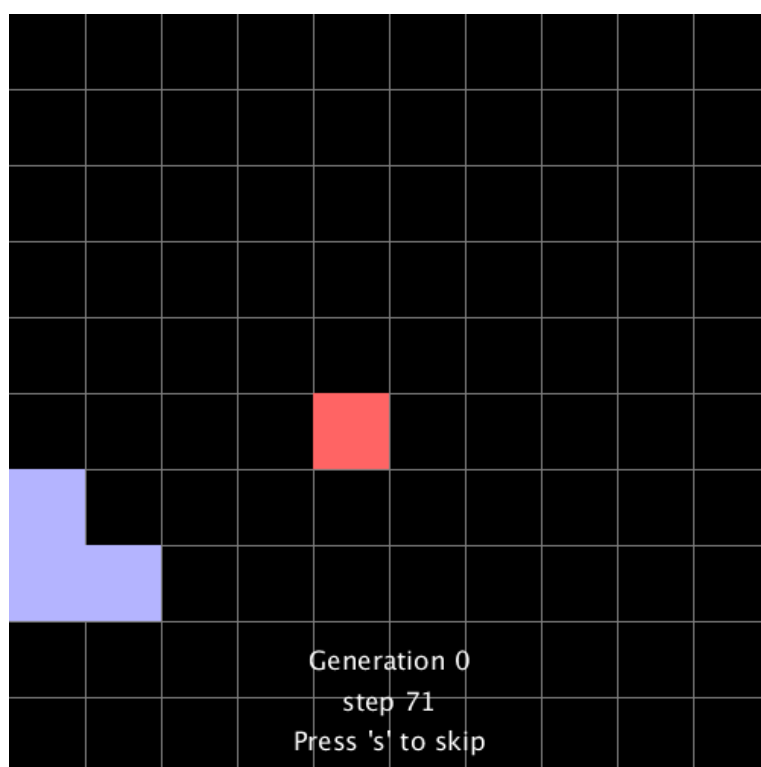


Рисунок 1.4 – Програмне рішення «Snake-neuroevolution» [19]

Переваги:

- не потребує градієнтних обчислень, що може бути корисним у складних середовищах;
- може знаходити нетривіальні стратегії гри.

Недоліки:

- може вимагати великої кількості обчислювальних ресурсів для еволюції;

- складність у налаштуванні параметрів еволюційного алгоритму.

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

В таблиці 1.2 представлено порівняння існуючих аналогів та програми, яка буде розроблена в ході виконання роботи.

Таблиця 1.2 – Порівняння існуючих аналогів

Можливості	SlytherSnake	Gym-Snake	SnakeQ	Snake-Neuroevolution
1	2	3	4	5
Методологія навчання	Гібридний метод	Мультиагентна система; класична гра з варіаціями	Deep Q-Learning	Neuroevolution (еволюція нейромереж)
Можливість симуляції середовища	Так (повна візуалізація найкращих агентів)	Так, з підтримкою мультиагентності	Лише для одного агента	Так
Візуалізація результатів	Симуляція найкращих агентів та створення графіків для аналізу ефективності навчання	Мінімальна візуалізація	Обмежена візуалізація	Мінімальна візуалізація

Продовження таблиці 1.2

1	2	3	4	5
Можливість роботи з кількома агентами	Ні	Так	Ні	Ні
Гнучкість налаштувань середовища	Висока	Обмежена, фіксована структура гри	Фіксована структура гри	Фіксована структура гри
Простота реалізації	Середня, потребує налаштування ГА та нейронних мереж	Висока, готове середовище	Середня, вимагає налаштування DQL	Висока, основна реалізація на основі ГА

Порівняльний аналіз в таблиці 1.2 показує, що розроблюваний застосунок «SlytherSnake» має конкурентні переваги завдяки гібридному підходу, який поєднує генетичні алгоритми та нейронні мережі. На відміну від «Gym-Snake», що пропонує лише базову симуляцію, «SlytherSnake» забезпечує гнучкість налаштувань та візуалізацію процесу навчання, а також генерує графіки для аналізу ефективності. У порівнянні зі «SnakeQ» та «Snake-Neuroevolution», «SlytherSnake» пропонує більш універсальне рішення, яке враховує як навчання агента, так і оптимізацію його рішень у складних середовищах.

Таким чином, запропонований програмний продукт відзначається високою адаптивністю, зручністю для аналізу та потенціалом для розширення, що робить його перспективним для використання як у навчальних, так і в науково-дослідницьких цілях.

Snake-Neuroevolution є найбільш відповідним прототипом. Ця програма вже містить базову реалізацію генетичних алгоритмів, що спрощує інтеграцію нейронних мереж.

1.3 Постановка завдання роботи

Основні функції програмного продукту програми «SlytherSnake»:

- симуляція ігрового середовища для гри «Змійка»;
- реалізація генетичного алгоритму для пошуку оптимального генному агента «Змійки»;
- вибір дії у середовищі агентом, залежно від генному та інформації про оточуюче середовище, за допомоги нейромережевого алгоритму;
- візуальна симуляція ігрового середовища для гри «Змійка» для найкращих агентів;
- створення графіків для аналізу ефективності навчання.

1.4 Висновки за розділом 1

Для розроблення гри «Змійка» було обрано підхід, що поєднує генетичні алгоритми та нейронні мережі, з метою забезпечення високої ефективності та адаптивності агента. Гібридний метод дозволяє використовувати генетичні алгоритми для оптимізації стратегій і нейронні мережі для обробки складних функцій вибору дій.

Під час аналізу існуючих програмних рішень були розглянуті такі аналоги, як «Gym-Snake», «SnakeQ» і «Snake-Neuroevolution». У результаті найбільш придатною для створення прототипу визнано програму «Snake-Neuroevolution».

На основі проведеного аналізу було вирішено розробити застосунок із розширеним функціоналом, що враховує найкращі аспекти аналогів та доповнює їх новими можливостями:

- симуляція ігрового середовища для гри «Змійка»;
- впровадження генетичного алгоритму для оптимізації стратегій агентів;
- використання нейронної мережі для прийняття рішень агентом на основі аналізу середовища;
- візуалізація ігрового процесу для найуспішніших агентів;
- інтеграція інструментів для створення графіків і аналізу результатів навчання.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ

2.1 Вибір інструментарію розробки програмного забезпечення

Правильний вибір інструментів для розробки програмного забезпечення є критично важливим етапом у процесі створення програми, оскільки це впливає на ефективність, стабільність і зручність роботи з проектом. Вибір мови програмування визначає можливості реалізації функціоналу, а також продуктивність та сумісність програми з іншими компонентами. Інтегроване середовище розробки (IDE) допомагає прискорити процес кодування, тестування та налагодження, забезпечуючи зручний інтерфейс і потужні інструменти для розробників. Вибір бібліотек є не менш важливим, оскільки вони дозволяють швидко реалізувати складні алгоритми і функції, зменшуючи час розробки і підвищуючи ефективність коду. У цьому розділі було розглянуто:

- мову програмування;
- IDE;
- бібліотеки Python, що використовуються для реалізації генетичних алгоритмів, нейронних мереж та інших необхідних функцій;
- структурну схему;
- основні модулі програми, що визначають логіку і взаємодію всіх компонентів системи.

2.1.1 Огляд особливостей мови програмування

Кожен рік опитування та індекси популярності мов програмування проводяться для того, щоб відслідковувати тенденції в галузі програмування, зрозуміти, які технології набирають популярності серед розробників, а також для допомоги у прийнятті рішень щодо вибору технологій для нових проектів. Такі дослідження дозволяють оцінити зміни у виборі мов

програмування серед професіоналів, дослідників та компаній. Зібрані дані можуть вказувати на найбільш ефективні та затребувані інструменти для вирішення різноманітних завдань, а також допомогти у визначенні стратегій розвитку освіти, набуття нових навичок та прогнозуванні попиту на певні технології в майбутньому.

Одним із найвідоміших і популярних опитувань у світі програмування є опитування GitHub, яке щорічно проводиться для збору даних про активність розробників, вибір технологій і тренди у галузі програмування. GitHub – це платформа для хостингу та розробки програмного забезпечення, яка активно використовується мільйонами розробників по всьому світу. Щорічне опитування GitHub Developer Survey надає цінну інформацію про популярні мови програмування, фреймворки, інструменти і тенденції в розробці програмного забезпечення. Ці дані дозволяють не тільки оцінити вибір технологій серед програмістів, а й спрогнозувати майбутні зміни в галузі. Опитування допомагає зрозуміти, які інструменти і мови є найбільш ефективними та затребуваними, а також визначити, як змінюється професійний розвиток і вподобання розробників.

У 2024 році Python обігнав JavaScript, ставши найпопулярнішою мовою на GitHub. Також спостерігається підвищений інтерес до AI-агентів і менших моделей, які потребують менше обчислювальних ресурсів, що відображає зсув у галузі, оскільки все більше людей зосереджуються на нових сценаріях використання штучного інтелекту [20].

Python є основною мовою для роботи з даними та розробки моделей штучного інтелекту. Інструменти, такі як TensorFlow, PyTorch і бібліотеки для аналізу даних (pandas, NumPy, scikit-learn), зробили Python незамінним для спеціалістів у цих галузях.

На рисунку 2.1 можна побачити зміни в популярності мов програмування [20].



Рисунок 2.1 – Популярність мов програмування за опитуванням від GitHub [20]

Python стає найпопулярнішою мовою на GitHub, випереджаючи JavaScript після 10 років його лідерства. Це перша значна зміна у топі двох мов з 2019 року, і вона відображає зростання популярності Python, яке супроводжувало бум генеративного штучного інтелекту за останні два роки [20].

Також на рисунку 2.2 можна побачити приріст у використанні Python в цьому році у порівнянні з 2023 роком [20].

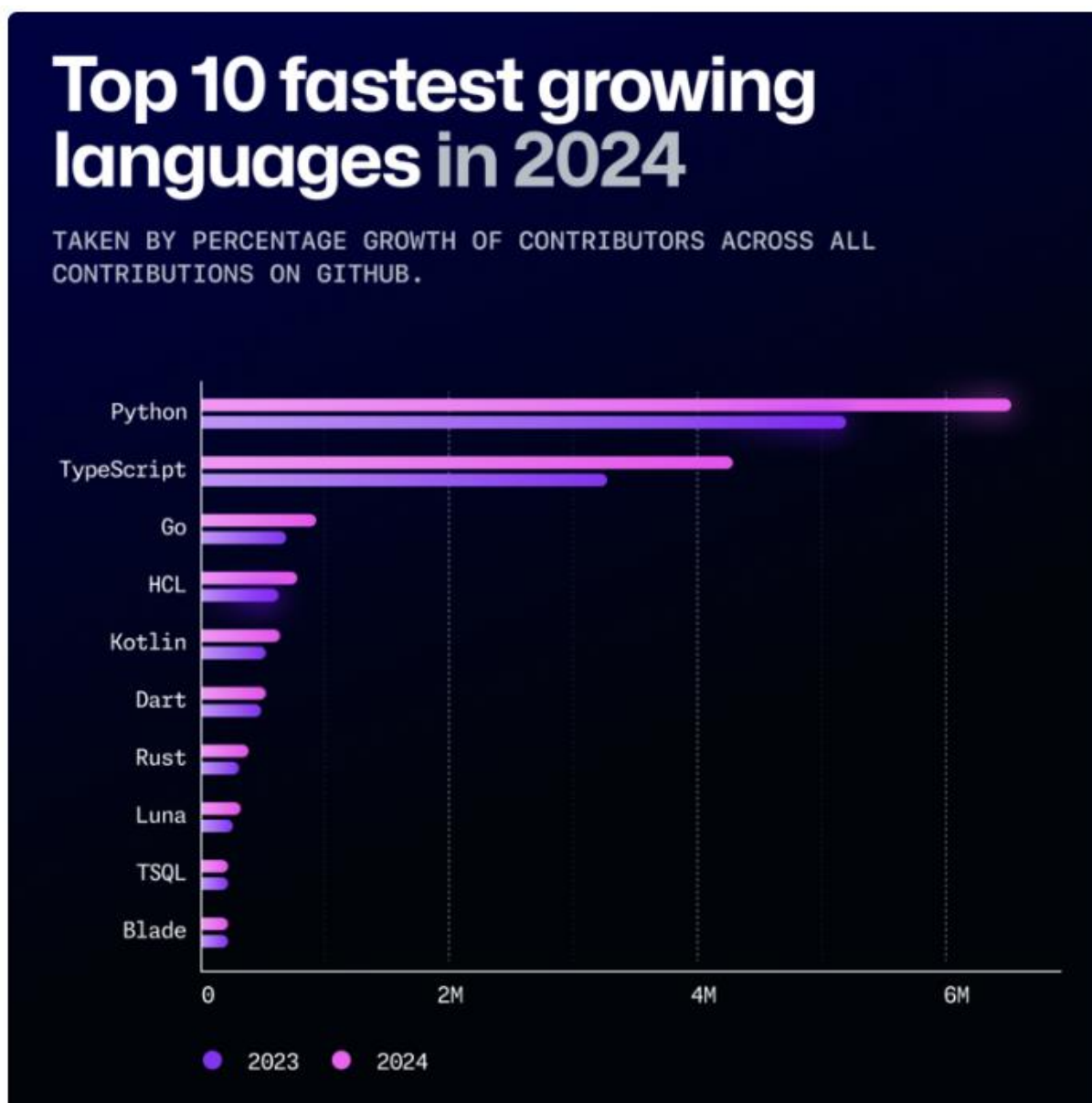


Рисунок 2.2 – Топ 10 швидко зростаючих мов програмування за опитуванням від GitHub [20]

Python є найбільш популярною мовою для науки про дані та досліджень, а його постійне зростання протягом останніх років може свідчити про те, що активність на GitHub виходить за межі традиційної розробки програмного забезпечення [20].

Індекс TIOBE є одним із найвідоміших показників популярності мов програмування у світі. Щомісячно оновлюваний, він базується на аналізі пошукових запитів із використанням таких платформ, як Google, Bing, Yahoo!, Wikipedia, Amazon та інших. TIOBE не лише відображає

популярність мов, але й демонструє тенденції їхнього розвитку, виявляючи, які мови набирають популярність, а які втрачають позиції. Індекс широко використовується фахівцями для вибору мов програмування в нових проектах, оцінки затребуваності технологій на ринку праці та прогнозування майбутніх тенденцій у галузі.

На рисунку 2.3 можна побачити графік популярності мов програмування з 2002 року по початок 2025 [21].

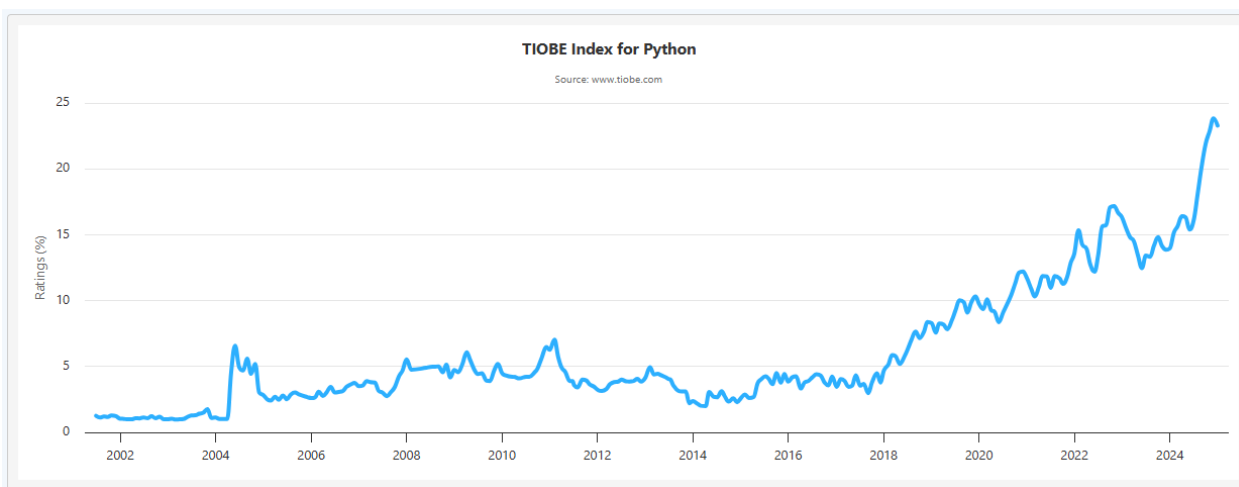


Рисунок 2.3 – Індекс TIOBE для мови програмування Python [21]

Згідно з даними індексу TIOBE на рисунку 2.3, Python досяг найвищої позиції у січні 2025 року, що свідчить про його зростаючу популярність серед розробників. Це зростання популярності можна пов'язати з його широким застосуванням у різних сферах, таких як наука про дані, машинне навчання та веб-розробка.

Опитування від DOU є одним із найвідоміших досліджень серед української IT-спільноти, яке проводиться щорічно. Його мета — відслідковувати тенденції у сфері інформаційних технологій, аналізувати популярність мов програмування, технологій, середовищ розробки, інструментів, а також з'ясувати демографічні та професійні характеристики IT-фахівців.

Результати таких опитувань дозволяють отримати цінну інформацію як для компаній, так і для самих фахівців, допомагаючи оцінити поточний стан галузі, визначити перспективи розвитку та краще орієнтуватися у світі технологій.

На рисунку 2.4 можна побачити опитування стосовно використання мови програмування на поточний час [22].

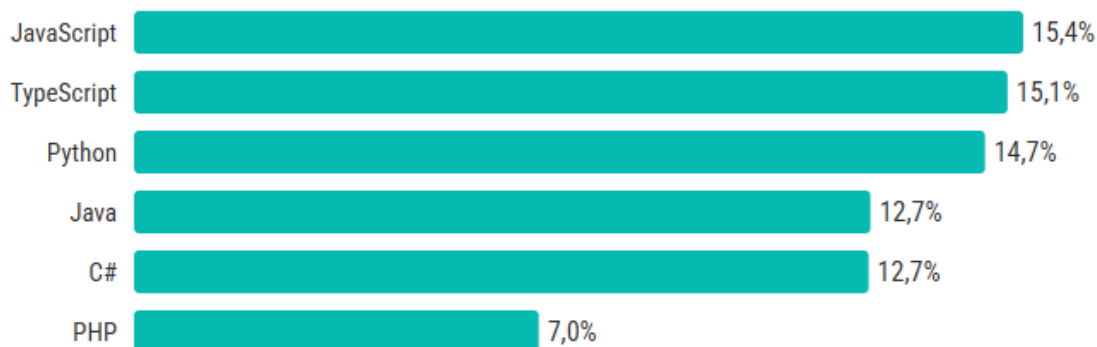


Рисунок 2.4 – Комерційне використання за опитуванням від DOU [22]

Як можна побачити на рисунку 2.4 15,1% опитаних використовують мову програмування Python у нових своїх проєктах. Також на рисунку 2.5 можна побачити приріст з 2019 по 2023 у використанні мови Python у нових своїх проєктах [22].

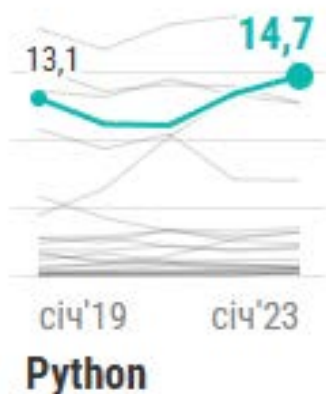


Рисунок 2.5 – Інформація про використання мови Python на поточний момент в період з січня 2019 по січень 2023 [22]

Також загалом можна побачити фінальні результати опитування про популярність мов програмування від компанії DOU на рисунку 2.6 [22].

Мова	Пишете для роботи зараз, %	Зміни	Пишете для роботи зараз	Використовуєте як додаткові	Використовуєте в своїх проектах	Індекс вподобання
JavaScript	15,3	-3,74 ▼	1259	2914	805	61,2%
TypeScript	15,0	+1,81 ▲	1237	1602	609	87,9%
Python	14,7	+1,29 ▲	1205	1239	559	80,7%
Java	12,7	-1,26 ▼	1043	708	423	79,1%
C#	12,7	-0,60 ▼	1040	435	391	89,3%
PHP	7,0	0,00 ■	573	386	236	71,1%

Рисунок 2.6 – Фінальні результати опитування від DOU [22]

Як можна побачити з рисунку 2.6 мова програмування Python залишається популярною серед українських розробників. Вона входить у трійку лідерів.

JavaScript – це високорівнева, динамічно типізована мова програмування, яка зазвичай використовується для створення інтерактивного функціоналу веб-сайтів та веб-застосунків. Вона є однією з основних технологій веб-розробки поряд із HTML та CSS [23].

C++ – це високорівнева мова програмування загального призначення, яка підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, процедурне та узагальнене програмування. Вона є розширенням мови C, додаючи нові функціональні можливості та особливості, такі як класи, шаблони, обробка виключень та стандартна бібліотека шаблонів [24].

Java – це об'єктно-орієнтована мова програмування високого рівня, яка була створена компанією Sun Microsystems (нині належить Oracle). Вона дозволяє розробляти портативні, безпечні та масштабовані додатки для різних платформ завдяки виконанню програм у віртуальній машині Java (JVM). Основний принцип Java – «Напиши один раз, запускай всюди» (Write Once, Run Anywhere), що забезпечує незалежність від операційної системи [25].

Під час розробки програмного забезпечення було вирішено обрати мову програмування, яка найкраще відповідає вимогам проекту, пов'язаного зі створенням штучного інтелекту для гри «Змійка». Особисто проаналізувавши можливості кількох популярних мов програмування, таких як Python, JavaScript, C++, та Java, в таблиці 2.1. Було враховано їх особливості, сильні сторони, недоліки та популярність серед розробників у сфері машинного навчання та штучного інтелекту.

Таблиця 2.1 – Порівняння мов програмування

Критерій	Python	JavaScript	C++	Java
1	2	3	4	5
Підтримка AI/ML	Багатий набір бібліотек	Обмежена підтримка, більше для візуалізації	Є бібліотеки, але менш популярні для AI	Можливості є, але менше спеціалізованих бібліотек
Швидкість виконання	Помірна	Помірна	Висока	Висока
Гнучкість	Підходить для будь-яких завдань	Краще для веб-розробки	Складна для швидкого прототипування	Менш зручна для швидкого створення прототипів
Інтеграція з бібліотеками	Легко інтегрується з бібліотеками та фреймворками	Легка інтеграція, але менше інструментів для AI	Інтеграція складна	Добра інтеграція, але менше поширена у AI
Популярність у AI/ML	Висока	Низька	Середня	Низька

Продовження таблиці 2.1

1	2	3	4	5
Застосування у ШІ	Лідирує у сфері AI та машинного навчання	Використовується для інтерфейсів і візуалізації	Використовується для складних симуляцій	Більше орієнтована на корпоративне використання

В результаті аналізу у таблиці 2.1 було вирішено обрати Python для розробки проекту через його домінування у сфері штучного інтелекту та машинного навчання, широкий спектр бібліотек і фреймворків, а також доступність великої кількості навчальних матеріалів. Простота синтаксису дозволяє швидко прототипувати та тестувати ідеї, що є важливим фактором у створенні програмного забезпечення для ШІ. Окрім цього, Python має активну спільноту, яка підтримує розробників у вирішенні складних завдань.

Історія створення Python розпочалася в кінці 1980-х років, коли Гвідо ван Россум, тодішній програміст у компанії Centrum Wiskunde & Informatica (CWI) у Нідерландах, розпочав роботу над новою мовою програмування. Ван Россум хотів створити мову, яка була б зручною, простою для читання та використовувала структури програмування, схожі на мовну концепцію ABC. Першу версію Python, Python 0.9.0, було випущено у 1991 році. Вона вже включала базові концепції, такі як обробка виключень, функції, мови обробки рядків, і підтримку модулів. На початковому етапі Python не був дуже популярним, але поступово набув визнання завдяки своїй простоті та багатій бібліотеці стандартних модулів [26].

Python отримав велике поширення у 2000-х роках завдяки багатьом оновленням, зокрема Python 2.0, яке вивело мову на новий рівень. Вже тоді почали розвиватися різні галузі використання Python, зокрема наукові

обчислення, автоматизація, веб-розробка та системне адміністрування. З 2010 року Python почав активно використовуватися у таких сферах, як машинне навчання, аналіз даних та штучний інтелект, що сприяло його популярності серед розробників [27].

Важливим етапом було випуск Python 3.0 у 2008 році, який став кроком до модернізації мови з покращеною підтримкою Unicode і новими можливостями. Ця версія мала деякі несумісності з Python 2, але з часом більшість розробників перемкнулися на нову версію [27].

За останні кілька років Python став однією з найбільш популярних мов програмування завдяки своїй зручності, потужним бібліотекам для обробки даних, машинного навчання та штучного інтелекту. Він залишається популярним і в 2020-х роках серед розробників завдяки широкому використанню в науці, освіті та промисловості.

Кілька основних особливостей мови Python [28]:

- Python відомий своєю чистою та зрозумілою граматикою. Його синтаксис максимально наближений до природної мови, що дозволяє програмістам зосередитися на вирішенні проблем, а не на складній синтаксичній структурі;
- Python поставляється з багатьма вбудованими модулями, що дозволяють виконувати різноманітні завдання, від роботи з файлами до обробки даних. Це знижує необхідність у сторонніх бібліотеках для багатьох стандартних операцій;
- динамічно типізована мова, що означає, що змінні не потребують явного вказання типів, і типи можуть змінюватися під час виконання програми. Це дає більшу гнучкість під час розробки;
- підтримує використання декораторів, які дозволяють змінювати функціональність функцій або класів без їх модифікації. Це зручно для реалізації функцій, таких як логування, перевірка доступу тощо;

- дозволяє використовувати об'єктно-орієнтовані парадигми програмування, надаючи підтримку для створення класів, наслідування, поліморфізму та інкапсуляції;
- має інтерактивний режим, що дозволяє швидко тестувати код у консолі та здійснювати розробку в реальному часі;
- підтримує паралельне виконання завдань через багатозадачність та багатопоточність, що особливо корисно для обробки великих обсягів даних;
- має відмінну сумісність з іншими мовами програмування, такими як C, C++ та Java, що дозволяє легко інтегрувати Python в багатокомпонентні системи.

Переваги Python [28]:

- має інтуїтивно зрозумілий синтаксис, що робить його ідеальним для початківців. Код, написаний на Python, часто виглядає так само, як опис алгоритму, що полегшує розуміння і підтримку програм;
- Читабельність та зрозумілість коду: Python робить акцент на читабельності коду, що спрощує його модифікацію та підтримку. Це забезпечує зручність при роботі в команді, оскільки програмісти можуть швидко зрозуміти, як працює програма, навіть якщо не писали її самі.
- має величезний набір бібліотек та фреймворків для різних завдань, таких як наука про дані (NumPy, pandas), машинне навчання (TensorFlow, scikit-learn), веб-розробка (Django, Flask), автоматизація (Selenium, PyAutoGUI), що дозволяє швидко розробляти складні програми;
- має одну з найбільших і найактивніших спільнот розробників. Це означає, що в разі проблем можна легко знайти рішення або задати питання на форумах чи платформах, таких як Stack Overflow;
- Python працює на різних операційних системах, таких як Windows, macOS, Linux. Це дозволяє використовувати Python для розробки програм, які працюють в різних середовищах без необхідності вносити великі зміни в код.

Недоліки Python [29]:

- Python є інтерпретованою мовою, тому програми можуть працювати повільніше порівняно з компільованими мовами, такими як C або C++. Це може бути критичним при роботі з великими обсягами даних або ресурсномісткими обчисленнями;
- використовує більше пам'яті в порівнянні з іншими мовами програмування, що може бути проблемою на пристроях з обмеженими ресурсами або в реальних часах, де важлива ефективність пам'яті;
- хоча Python можна використовувати для розробки мобільних додатків за допомогою фреймворків, таких як Kivy або BeeWare, підтримка мобільних платформ не така сильна, як у мовах, таких як Swift або Kotlin, які призначені спеціально для мобільної розробки;
- Python має обмеження, відоме як GIL, яке заважає ефективному використанню багатоядерних процесорів в певних випадках, оскільки в один момент може виконуватись лише один потік Python. Це обмежує паралельне виконання в деяких багатопоточних застосунках.

2.1.2 Огляд особливостей обраного середовища розробки

Вибір середовища розробки є критичним етапом у процесі створення програмного забезпечення, оскільки він безпосередньо впливає на ефективність та зручність роботи розробника.

Правильно обране IDE допомагає значно скоротити час на написання, тестування та дебагінг коду, а також покращує загальну продуктивність команди. Сучасні IDE мають безліч корисних функцій, таких як автодоповнення коду, інтеграція з системами контролю версій, інструменти для профілювання та оптимізації коду, що значно полегшують розробку програм.

Крім того, добре обране середовище дозволяє знижувати ймовірність помилок, полегшує відлагодження та підвищує стабільність програми, що особливо важливо при роботі з великими і складними проектами.

IDE складаються з інструментів для розробки, включаючи засоби на базі штучного інтелекту, які допомагають писати вихідний код, відлагоджувати та компілювати його. Деякі IDE призначені для конкретних мов програмування, в той час як інші вважаються найкращими для кількох мов, підтримуючи різноманітні середовища кодування [30].

Microsoft Visual Studio – це одне з найпопулярніших та найкраще інтегрованих середовищ розробки для веб-розробки.

Ви можете розробляти все, від консольних і графічних додатків, додатків Windows Forms/WPF, до вебсайтів, веб-додатків і веб-сервісів. Microsoft Visual Studio широко використовується в галузях комп'ютерного програмного забезпечення та IT-послуг. Воно однаково популярне серед програмістів та розробників програмного забезпечення в малих підприємствах та великих компаніях [30].

PyCharm – це інтегроване середовище розробки (IDE) для мови програмування Python, створене компанією JetBrains. Воно забезпечує розробників потужними інструментами для написання, налагодження та тестування коду. PyCharm підтримує широкий спектр функцій, які допомагають прискорити процес розробки та підвищити продуктивність [31].

Visual Studio Code (VS Code) – це безкоштовний, легкий і потужний редактор вихідного коду, розроблений компанією Microsoft. Він підтримує велику кількість мов програмування, має інтегровану систему налагодження та розширювану архітектуру через плагіни. Завдяки своїй універсальності, VS Code став одним із найпопулярніших редакторів коду серед розробників [32].

Для того, щоб зрозуміти, яке IDE є найкращим для мови програмування Python, яке відображено в таблиці 2.2.

Таблиця 2.2 – Порівняння IDE

Характеристика	Microsoft Visual Studio	PyCharm	Visual Studio Code
Особливості	Потужний редактор, підтримка багатьох мов	Спеціалізоване середовище для Python	Легкий текстовий редактор з розширеннями
Підтримка мови Python	Так	Так	Так
Переваги	Інтеграція з іншими технологіями, багатофункціональність, зручний інтерфейс	Чудова підтримка для Python, автоматичне доповнення коду, дебаггер	Легкий, зручний для багатьох мов програмування, великий вибір плагінів
Недоліки	Важкий для новачків, потребує ресурсів	Високі вимоги до системних ресурсів	Потрібно налаштовувати для повної підтримки Python

На основі порівняння з таблиці 2.2 різних IDE для Python, Microsoft Visual Studio є одним з найкращих варіантів для розробників, оскільки поєднує потужність інтеграції з іншими технологіями, багатофункціональність та зручний інтерфейс для Python. Цей інструмент надає необхідні функції для великих проектів та підтримує різноманітні мови, що робить його ідеальним вибором для професійних розробників, особливо тих, хто працює в багатомовних середовищах.

Microsoft Visual Studio (MVS) є однією з найпопулярніших і найкраще інтегрованих середовищ розробки для веб-розробки. Це потужне IDE

дозволяє розробляти різноманітні програми, від консольних додатків і GUI програм до Windows Forms/WPF додатків, вебсайтів, веб-додатків і веб-сервісів. MVS підтримує багаті інструменти для розробки, відлагодження, тестування та деплойменту додатків [33].

MVS популярний в індустрії комп'ютерного програмного забезпечення та IT-послуг, а також серед програмістів і розробників програмного забезпечення, як у малих, так і великих компаніях. Однією з переваг MVS є його здатність підтримувати багато мов програмування, включаючи Python, завдяки інтеграції з різними плагінами та інструментами.

Microsoft Visual Studio є відмінним вибором для розробників, які працюють з Python та іншими мовами програмування завдяки широким можливостям налаштування та інтеграції з різноманітними фреймворками та бібліотеками.

В 1997 році Microsoft Visual Studio було вперше анонсовано як частина Microsoft Developer Network (MSDN). Перші версії середовища включали інструменти для розробки програм для Windows, а також підтримку кількох мов програмування, таких як C++, Visual Basic та JavaScript. Visual Studio 97 стала основою для подальших версій, що включали численні вдосконалення й нові можливості для розробників [34].

В 2005 рік відбувся випуск Visual Studio 2005 приніс нові можливості для розробки, зокрема, підтримку 64-бітних додатків, нові можливості для веб-розробки та поліпшення інтерфейсу користувача [34].

В 2017 рік Microsoft випустила Visual Studio 2017, що принесло численні поліпшення у вигляді підтримки хмарних обчислень, інтеграції з Git, а також підтримки для нових мов програмування, включаючи Python і R для науки про дані [34].

Випуск Visual Studio 2019 включав нові можливості для покращення продуктивності, включаючи значно швидше завантаження середовища та підтримку новітніх стандартів для роботи з сучасними мовами програмування [29].

Деякі з ключових особливостей [35]:

- підтримує широкий спектр мов програмування, включаючи C++, C#, Python, JavaScript, F#, SQL, і багато інших. Це дозволяє розробникам працювати над різними типами проектів в одному середовищі;
- має вбудовану підтримку системи контролю версій Git, що дозволяє розробникам ефективно управляти своїм кодом, здійснювати коміти, створювати гілки, об'єднувати зміни і працювати з репозиторіями;
- надає інструменти для юніт-тестування (наприклад, NUnit, MSTest), що дозволяє автоматизувати перевірку якості коду, і підтримує інтеграцію з різними платформами для тестування;
- має потужний дебаггер, який дозволяє розробникам знаходити та виправляти помилки в коді. Він підтримує відладку коду як на рівні локального комп'ютера, так і в процесі розгортання на сервері або в хмарі;
- доступна велика кількість плагінів та розширень через Visual Studio Marketplace, що дозволяє налаштовувати інструмент під конкретні потреби розробників;
- має потужну систему автозавершення коду, яка допомагає розробникам писати код швидше, підказуючи можливі методи, змінні, типи і синтаксис у реальному часі.

Переваги Microsoft Visual Studio [35]:

- підтримує багато мов програмування;
- вбудовані інструменти профілювання і аналізу продуктивності допомагають розробникам знаходити і виправляти проблеми з продуктивністю, що робить додатки більш ефективними та менш ресурсозатратними;
- великий вибір плагінів і розширень через Visual Studio Marketplace дозволяє налаштувати IDE відповідно до специфічних потреб проекту.

Недоліки:

- є досить вимогливим до ресурсів комп'ютера, що може бути проблемою для старших або менш потужних машин. Це може вплинути на продуктивність під час розробки, особливо при роботі з великими проектами;
- установчий файл Visual Studio має великий розмір, що може зайняти значний обсяг дискового простору.

2.1.3 Огляд бібліотек Python, що будуть використовуватися

Бібліотеки Python є одним із ключових факторів, що визначають популярність цієї мови. Вони дозволяють розширити базові можливості Python, забезпечуючи готові інструменти для роботи з даними, машинного навчання, веб-розробки, автоматизації, графічного інтерфейсу тощо. Використання бібліотек дозволяє значно зменшити час на розробку та зосередитися на вирішенні прикладних задач замість написання базового функціоналу.

Бібліотека – це колекція модулів Python, які можуть бути імпортовані та використані в інших програмах. Вона забезпечує розробників інструментами для вирішення різних задач, зменшуючи час та зусилля, необхідні для написання власного коду [36].

У цьому проєкті планується використання кількох популярних бібліотек Python, таких як Pygame (для створення двовимірних ігор та симуляцій), Matplotlib (для візуалізації даних та побудови графіків), NumPy (для виконання числових обчислень та обробки багатовимірних масивів), а також Multiprocessing (для паралельного виконання процесів та використання багатоядерних процесорів для ефективної обробки даних). Бібліотека Multiprocessing дозволяє запускати незалежні процеси, що працюють паралельно, що значно покращує продуктивність у випадках, коли потрібно виконати великий обсяг обчислень або обробку даних. Кожна з цих бібліотек пропонує унікальні можливості, які спрощують та оптимізують процес розробки.

Ці бібліотеки дозволяють реалізувати графічний інтерфейс і візуалізувати результати роботи програми, що суттєво покращує користувацький досвід і забезпечує зручний інструмент для аналізу результатів.

Pygame – це набір модулів Python, призначений для розробки відеоігор. Він побудований на базі Simple DirectMedia Layer (SDL), що надає низькорівневий доступ до аудіо, клавіатури, миші та функцій відображення. Pygame дозволяє розробникам створювати 2D-ігри та мультимедійні додатки на Python, пропонуючи інструменти для роботи з графікою, звуком і пристроями введення [37].

Як популярна бібліотека з відкритим кодом, Pygame підходить як для початківців, так і для досвідчених розробників. Його проста синтаксична структура та гнучкість роблять його відмінним вибором для швидкої розробки ігор. Крім того, вона надає різні функції, такі як управління спрайтами, виявлення зіткнень і обробка зображень, що полегшує процес створення ігор.

Pygame має кілька особливостей, які роблять його зручним інструментом для розробки 2D-ігор і мультимедійних програм. Ось основні [38]:

- забезпечує потужний механізм для роботи з 2D-графікою. Він дозволяє малювати зображення, працювати зі спрайтами та анімаціями, а також підтримує ефекти прозорості та масштабування;
- включає підтримку роботи з аудіофайлами різних форматів, включаючи WAV, MP3 і Ogg, що дозволяє інтегрувати звукові ефекти та фонову музику у ваші ігри;
- надає прості інструменти для роботи з пристроями введення, такими як клавіатура, миша та джойстики. Це дозволяє розробникам швидко реалізувати механізми управління в грі;

- хоча Pygame спочатку не фокусується на мережевих можливостях, за допомогою додаткових модулів можна створювати багатокористувацькі ігри;

- завдяки своїй простій синтаксичній структурі, Pygame є чудовим вибором для початківців у програмуванні, оскільки розробники можуть легко навчитись основам ігор і швидко отримувати результат;

- Pygame працює на різних операційних системах, таких як Windows, macOS і Linux, що дозволяє розробляти ігри, які працюють на різних платформах без додаткових налаштувань.

Matplotlib – це потужна бібліотека Python для візуалізації даних, яка дозволяє створювати статичні, анімовані та інтерактивні графіки. Вона підтримує широкий спектр графічних елементів, включаючи лінійні графіки, гістограми, стовпчасті діаграми, кругові діаграми та багато інших. Matplotlib є основним інструментом для візуалізації даних у Python і широко використовується в наукових дослідженнях, аналізі даних та машинному навчанні [39].

Основні особливості Matplotlib [40]:

- створення лінійних графіків, гістограм, діаграм, кругових діаграм, 3D-графіків та багато інших;

- гнучке налаштування графіків, включаючи зміни кольорів, шрифтів, підписів, осей та інших елементів;

- можливість збереження графіків у різних форматах, таких як PNG, SVG, PDF;

- взаємодія з іншими бібліотеками, такими як Pandas, NumPy, для більш зручної обробки і аналізу даних.

NumPy – це бібліотека для мови програмування Python, яка надає потужні можливості для виконання числових обчислень. Вона забезпечує підтримку багатовимірних масивів та матриць, а також надає великий набір математичних функцій для операцій над цими структурами даних. Завдяки своїй ефективності та швидкості, NumPy широко використовується в

наукових обчисленнях, статистиці, машинному навчанні та аналізі даних [41].

NumPy є основною бібліотекою для числових обчислень у Python. Ось деякі з її основних особливостей [42]:

- дозволяє працювати з багатовимірними масивами (матрицями) і надає інструменти для маніпулювання ними;
- забезпечує значну оптимізацію обчислень завдяки використанню низькорівневих мов програмування, таких як C;
- надає велику кількість математичних функцій для роботи з масивами: від простих арифметичних операцій до складних лінійних алгебраїчних розрахунків;
- NumPy широко використовується разом із іншими бібліотеками, такими як Matplotlib для візуалізації та SciPy для наукових обчислень;
- завдяки ефективному коду, що написаний на C, NumPy має швидку продуктивність у порівнянні з іншими високорівневими мовами.

Multiprocessing – це бібліотека Python, яка надає інтерфейс для створення багатопроеесних додатків. Вона дозволяє паралельно виконувати код на декількох процесорах або ядрах. Бібліотека значно підвищує ефективність обчислень у випадках, коли необхідно виконати кілька важких обчислень одночасно, а також дозволяє поділити задачі на менші частини, що обробляються в різних процесорах [43].

2.2 Структурна схема програми

Структурна схема програми – це графічне представлення логіки роботи програми або її окремих частин. Вона використовується для візуалізації процесів, алгоритмів або зв'язків між компонентами системи. Структурна схема допомагає програмістам, розробникам та іншим учасникам проекту зрозуміти, як програма функціонує, як її компоненти взаємодіють і як дані рухаються між ними.

На рисунку 2.7 представлено структурну схему «SlytherSnake».

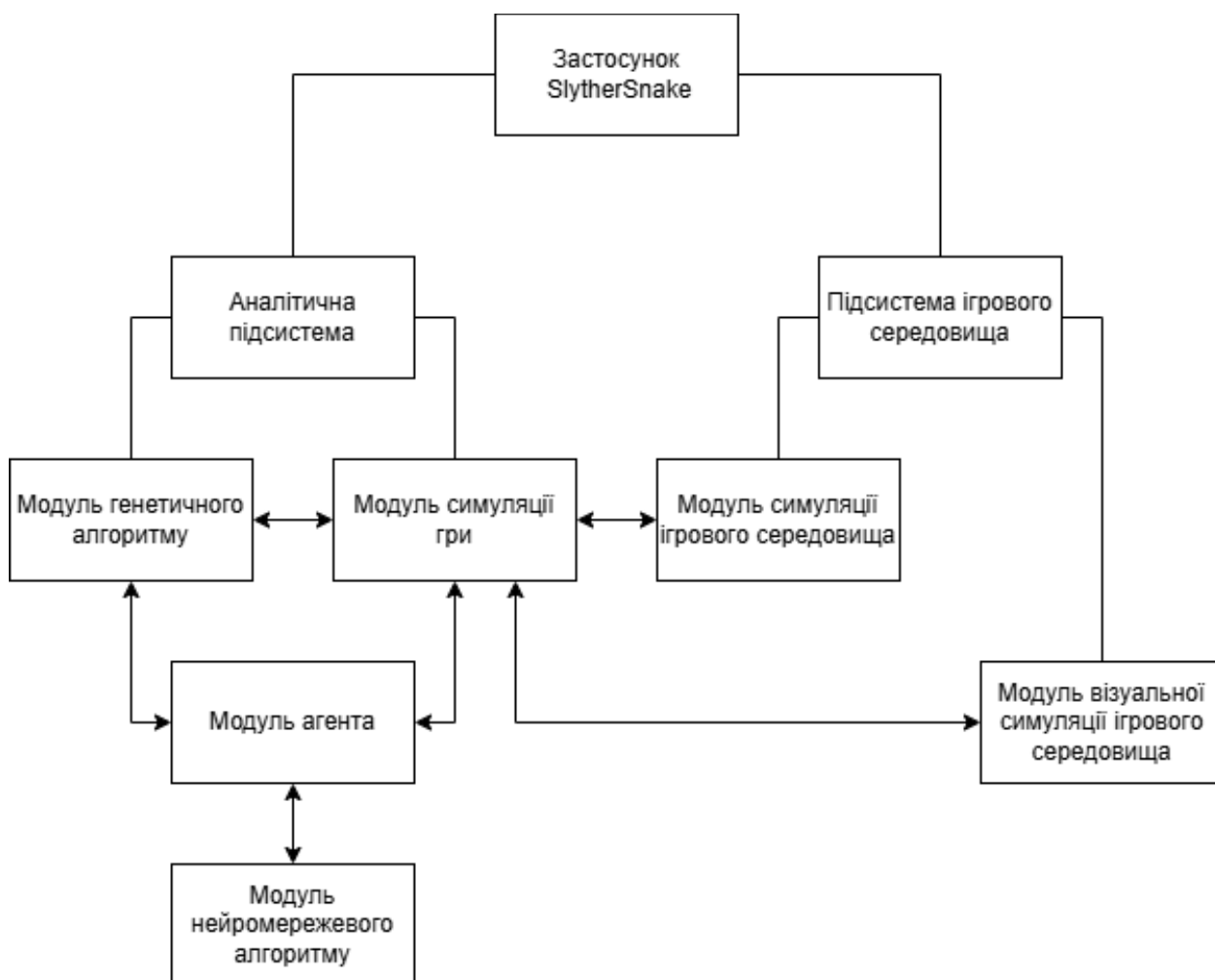


Рисунок 2.7 – Структурна схема

Як видно на рисунку 2.7 архітектура застосунку побудована на основі двох ключових підсистем: аналітичної та підсистеми ігрового середовища.

Підсистема ігрового середовища складається з модуля симуляції ігрового середовища та модуля візуалізації симуляції ігрового середовища.

Аналітична підсистема включає модуль генетичного алгоритму та модуль симуляції гри, які здійснюють тісну взаємодію між собою та з модулем агента. Модуль симуляції гри взаємодіє з обома модулями підсистеми ігрового середовища — симуляції ігрового середовища та візуалізації симуляції ігрового середовища.

Ключовою ланкою є модуль агента, який взаємодіє з модулем нейромережевого алгоритму.

2.3 Опис модулів розроблюваної системи

Таблиця 2.3 містить інформацію про кожен з модулів, їх функціональність.

Таблиця 2.3 – Опис модулів

Модуль	Опис
Модуль генетичного алгоритму	Цей модуль реалізує генетичний алгоритм для відбору найкращого генному змійки
Модуль симуляції гри	Цей модуль реалізує симуляцію процесу гри
Модуль агента	Цей модуль реалізує екземпляр агента, який грає за змійку
Модуль нейромережевого алгоритму	Цей модуль реалізує нейромережевий алгоритм для інтерпретації генному змійки та інформації з оточуючої середовища у вибір дії
Модуль симуляції ігрового середовища	Цей модуль реалізує ігрове середовище без візуальної симуляції для швидкої симуляції гри
Модуль візуальної симуляції ігрового середовища	Цей модуль реалізує ігрове середовище з візуальною симуляцією для наглядної демонстрації симуляції гри

2.4 Визначення функціональних вимог до програми

Схема прецедентів є важливим інструментом моделювання функціональних вимог програмного забезпечення, що дозволяє наочно представити взаємодію користувачів або зовнішніх систем із програмним

продуктом. Вона допомагає визначити основні сценарії використання, забезпечуючи структуроване розуміння функціональних можливостей та логіки роботи системи.

На рисунку 2.8 представлено схему прецедентів програми.

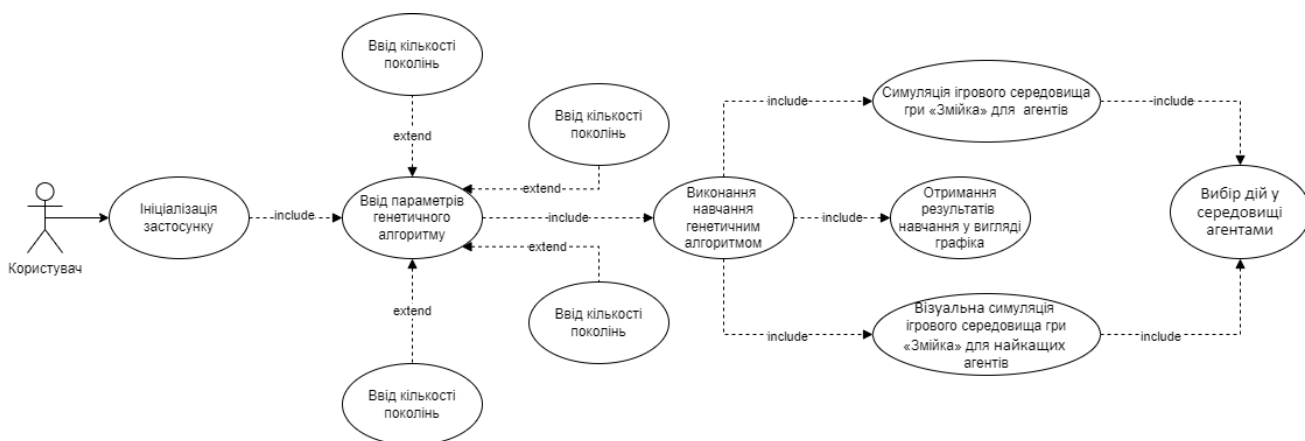


Рисунок 2.8 – Схема прецедентів

2.5 Висновки за розділом 2

У розділі 2 були детально розглянуті важливі аспекти, що стосуються вибору інструментів для розробки програмного забезпечення, а саме мови програмування, середовища розробки та бібліотек, які використовуються в даному проєкті також було створено структурну схему та описано модулі. Основні аспекти це:

- мова програмування Python була обрана через її простоту у використанні, велику кількість бібліотек для різноманітних задач, таких як обробка даних, розробка ігор, машинне навчання та візуалізація. Python є однією з найбільш популярних мов програмування, що робить її ідеальним вибором для швидкої розробки з підтримкою великої кількості інструментів і спільноти;

- як IDE, було обрано Microsoft Visual Studio через її гнучкість, багатофункціональність та підтримку різноманітних мов програмування, зокрема Python. Visual Studio забезпечує зручний інтерфейс для роботи з

великими проектами, інструменти для налагодження та багатий вибір плагінів для оптимізації процесу розробки;

Також слід зазначити інформацію про обрані бібліотеки:

- Pygame – для розробки двовимірних ігор та інтерактивних додатків, оскільки вона надає прості інструменти для роботи з графікою та звуком;

- Matplotlib – для візуалізації даних та створення графіків, що дозволяє ефективно аналізувати інформацію в реальному часі;

- NumPy – для роботи з числовими масивами та виконання складних математичних обчислень, що є основою для обробки та аналізу даних;

- Multiprocessing – для ефективного використання багатоядерних процесорів, що дозволяє оптимізувати обчислення та обробку даних.

Вибір цих інструментів базувався на вимогах проекту, потребах в обробці даних, створенні ігор та забезпеченні ефективної роботи на сучасних комп'ютерних системах. Кожен з обраних інструментів і бібліотек приносить конкретні переваги, що дозволяють оптимізувати розробку та забезпечити гнучкість, необхідну для досягнення поставлених цілей.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Алгоритм функціонування розробленої програми

Алгоритм функціонування розробленої програми є основою для забезпечення її правильного та ефективного виконання. Він описує послідовність кроків, які виконуються системою для досягнення заданого результату. Розробка алгоритму дозволяє структурувати роботу програми, визначити логіку її виконання та забезпечити узгодженість між усіма модулями. Алгоритм виступає базою для створення та аналізу програмного забезпечення.

На цьому етапі важливо визначити, як окремі частини програми взаємодіють між собою, які дані вони обробляють і як користувач взаємодіє з системою. Такий підхід забезпечує зрозумілу організацію процесів, дозволяє уникнути зайвих помилок та спрощує подальше розширення функціоналу.

Для ефективного представлення роботи програми та її взаємодії з користувачем використовується функціональна схема. Це інструмент, який дозволяє унаочнити послідовність виконання ключових операцій та логіку роботи всієї системи.

Функціональна схема – це графічне відображення основних процесів і дій, які виконує програма. Вона включає в себе основні етапи функціонування системи, їх взаємозв'язок і послідовність виконання. Така схема відображає як загальну структуру, так і деталі роботи окремих компонентів програми.

Функціональна схема дозволяє перейти від текстового опису алгоритму до його наочного представлення, яке допоможе як розробникам, так і користувачам зрозуміти логіку функціонування програми. Це сприяє чіткому визначенню ролі кожного модуля та забезпечує легкість аналізу, тестування та документування.

Основних процеси і дії, які виконує програма представлені у функціональній схемі на рисунку 3.1.

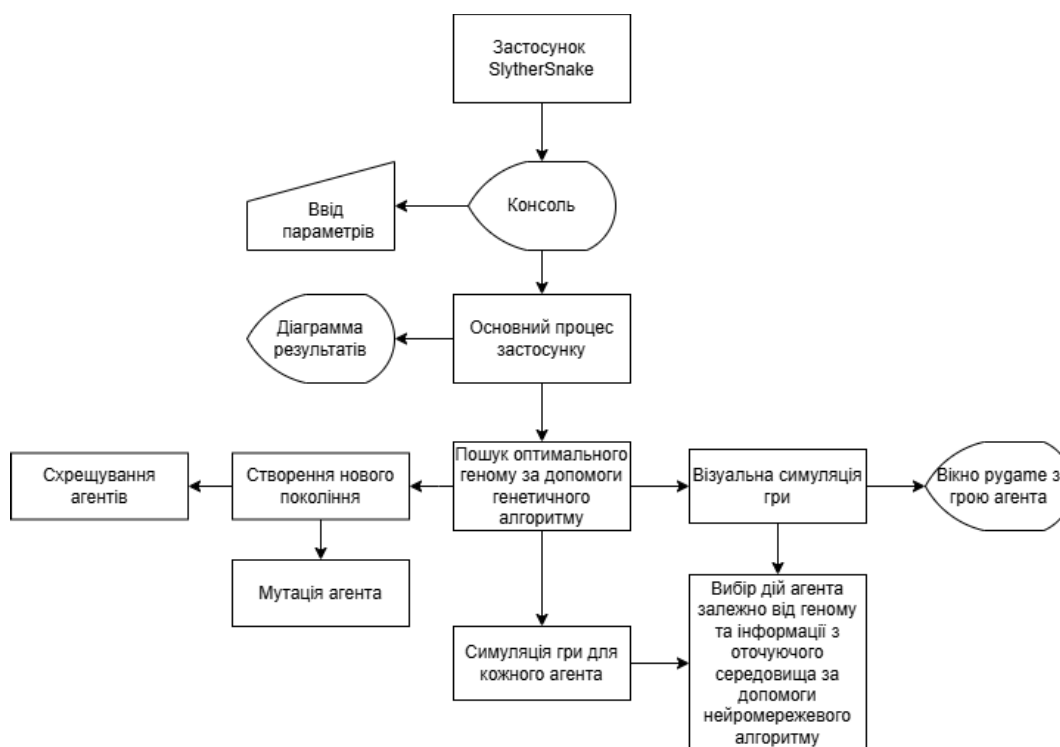


Рисунок 3.1 – Функціональна схема

У процесі створення програмного забезпечення «SlytherSnake» були сформовані такі алгоритми:

- основний алгоритм;
- генетичний алгоритм;
- алгоритм створення нового покоління.
- алгоритм мутації
- алгоритм схрещування
- алгоритм симуляції гри;
- алгоритм візуальної симуляції гри;
- алгоритм прийняття рішення.

На рисунку 3.1 можна побачити, що після запуску програмного забезпечення запускається консольний інтерфейс, де можливо задавати параметри для налаштування роботи алгоритму. На основі введених параметрів активується основний модуль, що виконує генетичний алгоритм.

Після завершення його роботи генерується діаграма, яка відображає динаміку навчання.

Генетичний алгоритм базується на взаємодії кількох підалгоритмів, зокрема: алгоритму створення нового покоління, алгоритму симуляції гри та алгоритму її візуалізації. Алгоритм створення нового покоління, своєю чергою, включає механізми мутації та схрещення.

Алгоритми симуляції та візуальної симуляції гри застосовують алгоритм прийняття рішень. Візуальна симуляція передбачає створення графічного інтерфейсу на основі бібліотеки Pygame, де демонструється ігровий процес найкращого агента з поточного покоління.

3.1.1 Основний алгоритм

Основний алгоритм розроблено для реалізації процесу зчитування параметрів, необхідних для навчання за допомогою генетичного алгоритму, його ініціалізації та виводу результатів для подальшого аналізу користувачем. На рисунку 3.2 зображено блок-схему алгоритму.

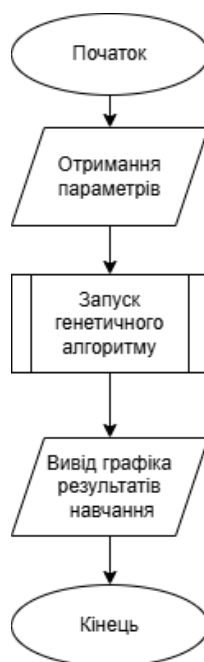


Рисунок 3.2 – Блок-схема для основного алгоритму

3.1.2 Генетичний алгоритм

Цей алгоритм створено для реалізації пошуку оптимального генному. На рисунку 3.3 зображено блок-схема алгоритму.

У кожному поколінні відбувається оцінка популяції, вибір батьків, схрещування, мутація і створення нової популяції. Перші кілька кращих особин (за результатами селекції) зберігаються без змін (елітизм). Процес повторюється для заданої кількості поколінь.

Функція пристосованості (fitness function), що визначає якість індивіда, може бути записана для алгоритму у вигляді:

$$f_i = \text{fitness}(G_i) \quad (3.1)$$

де G_i — геном i -ї особини;

f_i — її значення пристосованості.

Ймовірність вибору кожного батька обчислюється як:

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad (3.2)$$

де N — кількість особин у популяції.

На рисунку 3.3 зображено блок-схему алгоритму.



Рисунок 3.3 – Блок-схема для генетичного алгоритму

3.1.3 Алгоритм створення нового покоління

Цей алгоритм створено для реалізації створення нового покоління за допомоги елітизму, мутацій та схрещування.

Процес формування нового покоління можна подати як:

$$P_{t+1} = E_t \cup C_t \quad (3.3)$$

де E_t — підмножина найкращих елітних особин покоління t ;

C_t — множина нових особин, створених шляхом схрещування та мутації.

На рисунку 3.4 зображено блок-схему алгоритму.



Рисунок 3.4 – Блок-схему для алгоритму створення нового покоління

3.1.4 Алгоритм мутації

Мутація в контексті генетичних алгоритмів та штучного інтелекту – це один із ключових операторів еволюційного обчислення, який використовується для підтримання генетичного різноманіття в популяції. Мутація вносить випадкові зміни в генетичний код (хромосому) окремих індивідів, тим самим допомагаючи алгоритму уникнути передчасної збіжності до локального оптимуму.

Математично мутацію можна описати так:

$$g'_i = g_i + \mathcal{N}(0, \sigma) \quad (3.4)$$

де g'_i — початкове значення гена;

$\mathcal{N}(0, \sigma)$ — випадкове значення, взяте з нормального розподілу з нульовим середнім і стандартним відхиленням σ .

Цей процес імітує природні мутації у біології та сприяє дослідженню нового пошукового простору, що може привести до кращих рішень. Мутація визначається як механізм внесення випадкових змін до окремих компонентів рішень (генів), що дозволяє алгоритму досліджувати нові області пошукового простору [44].

Мутація в генетичних алгоритмах – це важливий етап, який допомагає зберігати різноманітність у популяції й уникати потрапляння алгоритму в локальні оптимуми.

В таблиці 3.1 наведено порівняння способів мутації.

Таблиця 3.1 – Порівняння способів мутації

Тип мутації	Опис	Переваги	Недоліки
1	2	3	4
Гаусова мутація	Зміна значень генів за допомогою випадкового нормального відхилення з адаптивною зміною величини мутації в залежності від покоління.	Динамічно адаптується до процесу еволюції, зменшуючи хаос на більш пізніх етапах. Контролює ступінь мутації, що допомагає зберігати стабільність і уникати занадто великих змін.	Може бути недостатньо агресивною на початкових етапах, коли великі зміни можуть бути корисні.
Просте додавання або віднімання	Випадкове додавання або віднімання фіксованої величини від значення гена.	Простий метод. Легко реалізується.	Може призвести до втраченого різноманіття, якщо мутація відбувається занадто часто. Не дозволяє динамічно адаптувати мутацію до розвитку популяції.

Продовження таблиці 3.1

1	2	3	4
Бітова мутація	Зміна одного бітового значення (для бінарних генномів) на протилежне.	Проста в реалізації для бінарних генів. Висока ймовірність незначних змін.	Не підходить для числових або реальних значень генів. Може привести до локальних оптимумів, не забезпечуючи достатньої різноманітності.
Перемішування генів	Обмін частинами генному між різними генами (використовується в деяких випадках для перестановок в послідовностях).	Може підвищити різноманіття популяції.	Може призвести до розбалансованості генному. Потребує складних алгоритмів для правильної реалізації.

В результаті аналізу у таблиці 3.1 було вирішено працювати з адаптивною Гаусовою мутацією, яка динамічно зменшується з кожним поколінням.

Це дозволяє зберегти стабільність у пізніших поколіннях, коли значні мутації можуть лише порушити досягнуту оптимізацію.

Це підвищує ефективність еволюційного процесу і дозволяє зберігати різноманіття на ранніх етапах.

Адаптивну Гаусову мутацію, де σ зменшується із кожним поколінням можна виразити так:

$$\sigma = \max(0.1 * (1 - \frac{t}{T}), 0.01) \quad (3.5)$$

де t — поточне покоління;

T — загальна кількість поколінь.

Після мутації значення гена обмежується у межах $[-1,1]$:

$$g'_i = \text{clip}(g'_i, -1.1) \quad (3.6)$$

Інші методи, як проста мутація або бітова мутація, не надають такої адаптивності і можуть призвести до швидкого втраченого різноманіття або локальних мінімумів.

Мутація застосовується до кожного гена особини з певною ймовірністю (за замовчуванням 1%).

Мутація змінює ген значенням, що підсумовується з випадковим нормальним відхиленням, зменшеним в залежності від поточного покоління.

Цей алгоритм створено для реалізації процесу мутації генома. На рисунку 3.5 зображено блок-схему алгоритму.

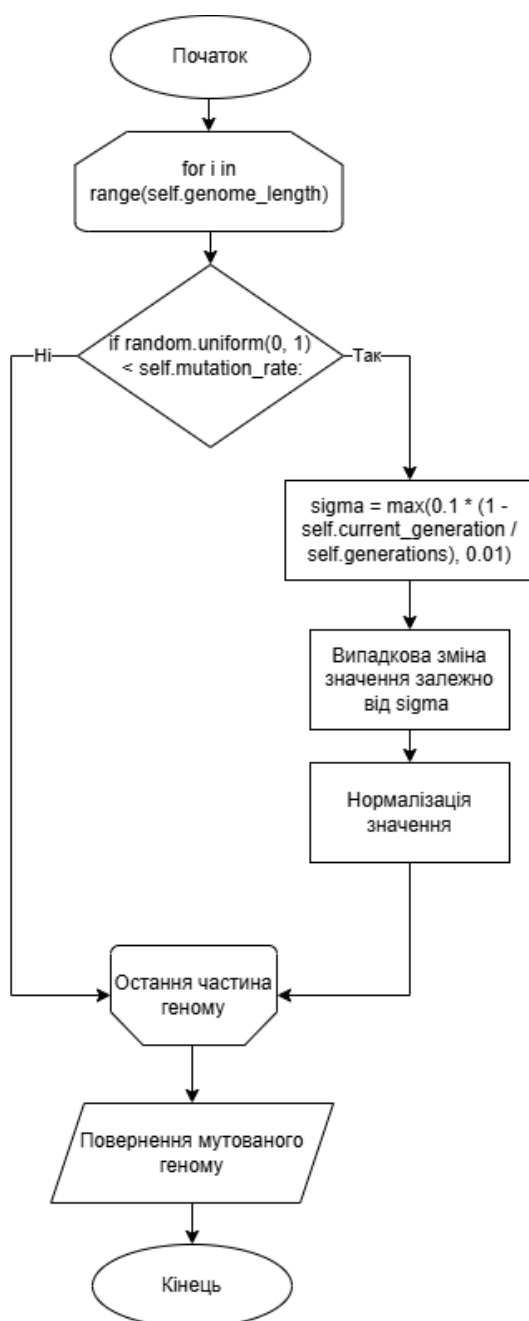


Рисунок 3.5 – Блок-схему для алгоритму мутації

Переваги підходу:

- мутація враховує прогрес алгоритму, що підвищує його ефективність. На ранніх етапах – дослідження, на пізніх – уточнення;
- використання нормального розподілу дозволяє регулювати величину змін, що корисно для задач із безперервними параметрами;
- гарантія, що гени не вийдуть за задані межі, що запобігає помилкам або невідповідним результатам.

3.1.5 Алгоритм схрещування

Схрещування – це одна з ключових операцій генетичних алгоритмів (ГА), яка імітує процес статевого розмноження у природі. Воно полягає в комбінуванні генетичної інформації двох батьків для створення нових особин (нащадків) із характеристиками обох. Ця операція сприяє обміну генетичним матеріалом між батьками, забезпечуючи пошук нових можливих рішень у просторі задачі [45].

В таблиці 3.2 наведено порівняння способів схрещування.

Таблиця 3.2 – Порівняння способів мутації

Тип кросовера	Опис	Переваги	Недоліки
1	2	3	4
Лінійний кросовер (ваш метод)	Геном дитини формується як лінійна комбінація генів батьків з певними вагами (60% і 40%).	Простий і зрозумілий, легкий в реалізації. Забезпечує поступовий перехід між батьками.	Може бути недостатньо гнучким для деяких завдань. Проста лінійна комбінація не завжди дає оптимальні результати.
Одноточковий кросовер	Вибір випадкової точки в геномах батьків і обмін частинами після цієї точки.	Простий і популярний метод. Підходить для задач з бінарними або цілими генами.	Може втратити важливу інформацію, якщо точка вибрана невдало. Не зберігає багато генетичної різноманітності.

Продовження таблиці 3.2

1	2	3	4
Двоточковий кросовер	Вибір двох точок в геномах батьків, після яких відбувається обмін.	Дозволяє зберігати більше різноманіття в потомстві.	Може бути складнішим в реалізації. Іноді може привести до непередбачуваних результатів, якщо точки вибрані невдало.
Рівномірний кросовер	Випадковий вибір генів з батьків для кожного гена дитини.	Забезпечує високу генетичну різноманітність.	Може призвести до дуже великих змін у геномі, що може порушити стабільність.

В результаті аналізу у таблиці 3.2 було вирішено працювати з лінійним кросовером з конкретними вагами (60/40), що дозволяє створювати помірковані комбінації між батьками. Це дає можливість контролювати ступінь змішування генів та дозволяє більш точно підлаштовувати генетичний матеріал під завдання.

Використаний метод — лінійний кросовер, визначається рівнянням:

$$g_i^{child} = \alpha * g_i^{(1)} + (1 - \alpha) * g_i^{(2)} \quad (3.7)$$

де $g_i^{(1)}$ і $g_i^{(2)}$ — відповідні гени першого та другого батьків;

$\alpha = 0.6$ — вага першого батька.

Результат обмежується в межах $[-1,1]$:

$$g_i^{child} = clip(g_i^{child}, -1.1) \quad (3.8)$$

Одноточковий або двоточковий кросовер можуть призвести до втраченого різноманіття, якщо вибір точок не буде оптимальним, і не дають такого плавного контролю, як ваш метод, який може бути налаштований для змішування більш ефективно.

Метод crossover здійснює схрещування між двома батьками. Геном нащадка формуються як комбінація генів батьків з певними вагами (у даному випадку 60% від першого батька та 40% від другого).

Цей алгоритм створено для реалізації процесу схрещення агентів. На рисунку 3.6 зображено блок-схему алгоритму.



Рисунок 3.6 – Блок-схема для алгоритму схрещування

3.1.6 Алгоритм симуляції гри

Цей алгоритм створено для реалізації симуляції процесу гри для агента. На рисунку 3.7 зображено блок-схема алгоритму.



Рисунок 3.7 – Блок-схема для алгоритму симуляції гри

3.1.7 Алгоритм візуальної симуляції гри

Цей алгоритм створено для реалізації візуальної демонстрації гри агента. На рисунку 3.8 зображено блок-схему алгоритму.



Рисунок 3.8 – Блок-схема для алгоритму візуальної симуляції гри

3.1.8 Алгоритм прийняття рішення

Цей алгоритм поєднує еволюційний підхід з елементами нейронних мереж для оптимізації поведінки агента в ігровому середовищі. Модель має генетичну складову (геном), яка визначає структуру та ваги для агентної поведінки, проте використовує непряму нелінійність, щоб забезпечити більшу гнучкість в ухваленні рішень.

Цей алгоритм можна назвати частково нейромережевим, оскільки він використовує концепцію нелінійної трансформації вхідних ознак (через функцію активації $\tanh$), що є класичним елементом нейронних мереж. Хоча структура цього алгоритму спрощена, і в ньому немає складної багатошарової архітектури, використання геному як ваг для прийняття рішень є аналогом ваг в нейронних мережах.

На рисунку 3.9 зображено блок-схему алгоритму.



Рисунок 3.9 – Блок-схема для алгоритму прийняття рішення

3.2 Опис архітектури створеної моделі нейронної мережі

Для реалізації інтелектуальної складової програмного забезпечення «SlytherSnake» важливим етапом є вибір відповідної нейронної мережі. Нейронна мережа виступає ключовим компонентом для розв’язання задач, пов’язаних із навчанням гри, адаптацією до змінних умов та підвищенням ефективності прийняття рішень. Правильний вибір архітектури нейронної мережі дозволяє оптимізувати процес обробки даних, забезпечити точність прогнозів та знизити затрати обчислювальних ресурсів. Далі буде розглянуто основні фактори, які вплинули на вибір нейромережі, а також визначено її архітектуру, що найкраще підходить для даного проєкту.

Нейромережа складається з чотирьох основних компонентів: шару вхідних характеристик, геному, шару дій та механізму вибору оптимальної дії. Її архітектуру можна побачити на рисунку 3.10.

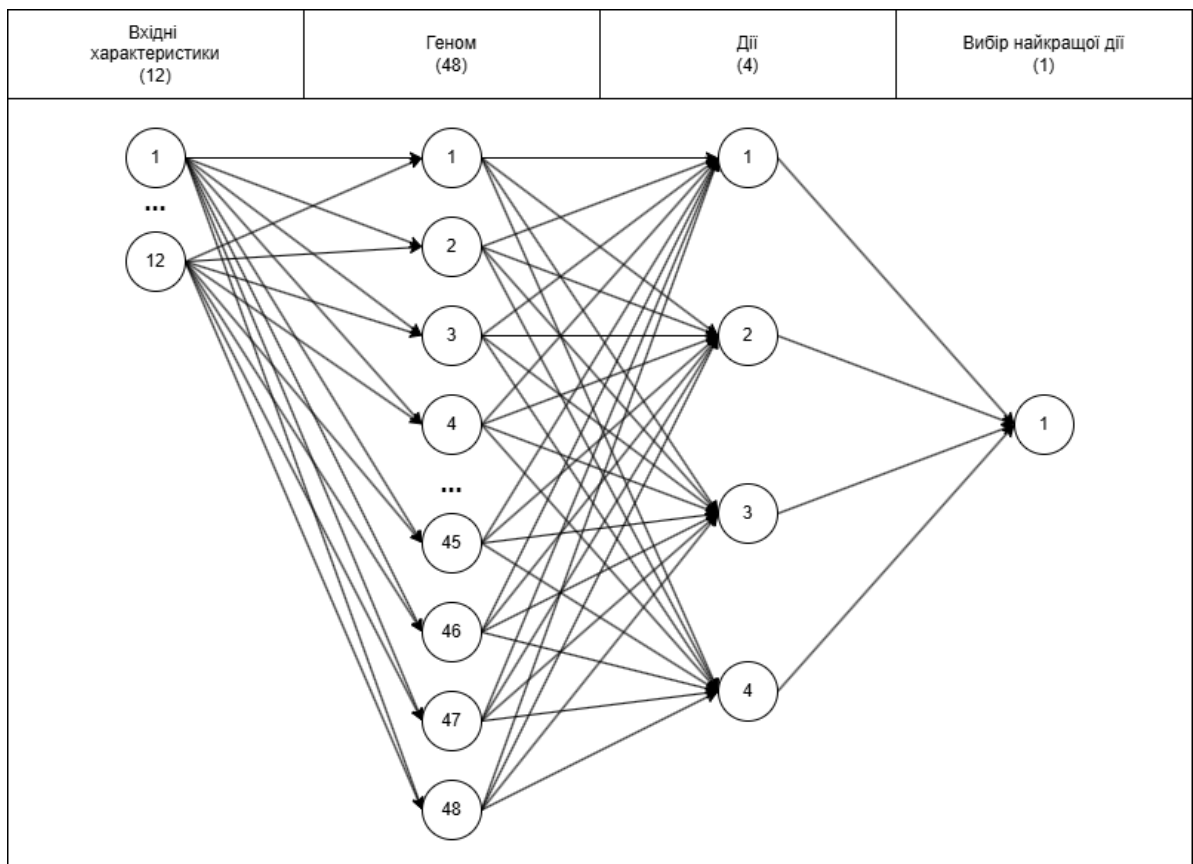


Рисунок 3.10 – Створена нейромережа

Вхідні характеристики – це інформація про стан ігрового середовища, вони проходять через нелінійну активацію ($\tanh$) для врахування нелінійних залежностей між характеристиками. Після цього ваги геному визначають вплив кожної характеристики на вибір дії. В результаті на виході нейромережа обирає дію, яка має найвищу оцінку, і змійка виконує цей рух.

Вхідні характеристики (12 нейронів) – цей шар відповідає за отримання і нормалізацію інформації про стан ігрового середовища, містить значення, такі як: відстань до їжі по x та y координатах, відстань до стін у чотирьох напрямках, ознаки розташування їжі відносно змійки (вертикальне та горизонтальне), відстань до перешкод у чотирьох напрямках.

Геном (48 ваг) представлений у вигляді набору ваг, які зв'язують вхідні нейрони з нейронами шару оцінки дій. Кожна група ваг відповідає за обчислення оцінки певної дії.

Дії (4 нейрони) – цей шар відповідає за оцінювання можливих дій змійки: рух вгору, рух вниз, рух ліворуч, рух праворуч. Для кожної дії обчислюється оцінка як скалярний добуток між нормалізованими вхідними характеристиками та відповідними вагами геному.

Вибір найкращої дії (1 нейрон). Серед усіх обчислених оцінок обирається дія з найвищим значенням. Ця дія вважається найбільш доцільною в поточному стані гри, і змійка виконує відповідний рух.

3.3 Опис модулів розроблюваної системи

Для розроблених модулів в ході роботи були утворенні такі класи, які показані рисунку 3.11:

- Individual;
- SnakeGameSimulation;
- SnakeGameVisualSimulation;
- GeneticAlgorithm;

- GameEnvironment;
- GameProcess.

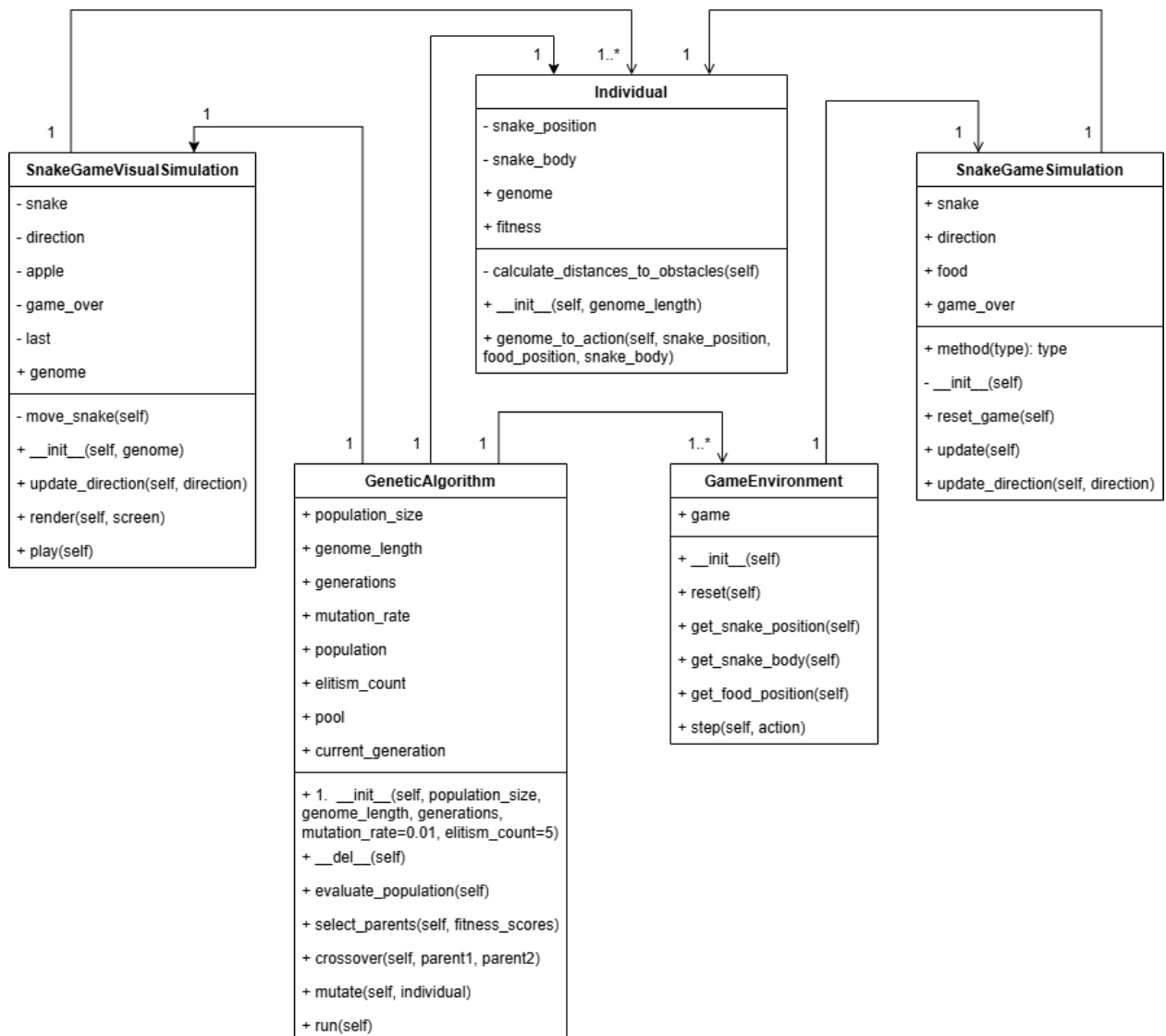


Рисунок 3.11 – Діаграма класів

Як видно з рисунку 3.11, класи системи структурно взаємопов’язані за допомогою асоціацій різних типів.

Клас GeneticAlgorithm формує асоціації типу «один до одного» з класами Individual та SnakeGameVisualSimulation, а також асоціацію типу «один до багатьох» із класом GameEnvironment. Клас GameEnvironment пов’язаний асоціацією «один до одного» з класом SnakeGameSimulation, який, у свою чергу, має асоціацію «один до одного» з класом Individual. Клас

SnakeGameVisualSimulation встановлює асоціацію типу «один до багатьох» із класом Individual

3.3.1 Клас Individual

Цей клас представляє індивідуума в генетичному алгоритмі для гри «Змійка». Кожен індивідуум має геном, який визначає його поведінку в грі, і методи для перетворення ігрового стану на дії. Дані та методи класу наведені у таблиці 3.3.

Таблиця 3.3 – Опис полів та методів класу Individual

Поля та методи класу	Опис
1	2
<i>public:</i>	
genome	Змінна, що містить геном індивідуума, представлений як масив випадкових чисел у діапазоні від -1 до 1
fitness	Змінна, що містить оцінку пристосованості індивідуума, яка встановлюється під час оцінювання в алгоритмі
__init__(self, genome_length)	Функція, що ініціалізує індивідуума з випадковим геномом заданої довжини та початковим значенням пристосованості
genome_to_action(self, snake_position, food_position, snake_body)	Функція, що перетворює геном індивідуума та стан гри (позицію змійки, їжі та її тіла) на дію (напрямок руху змійки)

Продовження таблиці 3.3

1	2
<i>private:</i>	
snake_position	Змінна, що містить поточну позицію голови змійки
snake_body	Змінна, що містить поточні координати всього тіла змійки
calculate_distances_to_obstacles(self)	Функція, що обчислює відстані до перешкод (стіни або тіла змійки) у всіх чотирьох напрямках: вгору, вниз, вліво, вправо

3.3.2 Клас SnakeGameSimulation

Цей клас моделює гру «Змійка», забезпечуючи базову логіку для управління змійкою, генерації їжі та обробки станів гри, таких як зіткнення чи з'їдання їжі.

Дані та методи класу наведені у таблиці 3.4.

Таблиця 3.4 – Опис полів та методів класу SnakeGameSimulation

Поля та методи класу	Опис
1	2
<i>public:</i>	
snake	Змінна, що містить список координат, що представляють змійку
direction	Змінна, що містить напрямок руху змійки у вигляді дельта-кортежу
food	Змінна, що містить координати їжі на полі

Продовження таблиці 3.4

1	2
game_over	Змінна, що містить булева змінна, що вказує на завершення гри
__init__(self)	Функція, що ініціалізує гру, викликаючи метод reset_game для встановлення початкових умов
reset_game(self)	Функція, що скидає гру до початкового стану, ініціалізуючи змійку, напрямок руху та розміщуючи їжу
update(self)	Функція, що) оновлює стан гри: переміщує змійку, перевіряє зіткнення зі стінами чи тілом, а також обробляє з'їдання їжі
update_direction(self, direction)	Функція, що оновлює напрямок руху змійки
private:	
place_food(self)	Функція, що розміщує їжу у випадковій позиції, яка не перетинається з тілом змійки

3.3.3 Клас SnakeGameVisualSimulation

Цей клас відповідає за візуалізовану симуляцію гри «Змійка», де дії змійки визначаються геномом найкращого індивідуума. Реалізовано візуалізацію, логіку руху, взаємодію зі стінами, яблуком і самою змійкою. Дані та методи класу наведені у таблиці 3.5.

Таблиця 3.5 – Опис полів та методів класу SnakeGameVisualSimulation

Поля та методи класу	Опис
<i>public:</i>	
genome	Змінна, що містить геном, який визначає дії змійки на основі стану гри
__init__(self, genome)	Функція, що ініціалізує гру
update_direction(self, direction)	Функція, що оновлює напрямок руху змійки відповідно до переданої дії
render(self, screen)	Функція, що відповідає за відображення поточного стану гри
play(self)	Функція, що запускає симуляцію
<i>private:</i>	
snake	Змінна, що містить список координат блоків, які утворюють тіло змійки. перший елемент — голова
direction	Змінна, що містить поточний напрямок руху змійки, представлений як вектор зміщення
apple	Змінна, що містить координати яблука на ігровому полі
game_over	Змінна, що містить флаг завершення гри
last	Змінна, що містить координати останнього блоку тіла змійки
move_snake(self)	Функція, що рухає змійку в поточному напрямку, додаючи нову голову змійки й видаляючи хвіст. Виявляє зіткнення зі стінами чи тілом змійки й завершує гру у разі колізії. Перевіряє, чи з'їла змійка яблуко, додає новий сегмент до змійки та генерує нове яблуко в унікальній позиції

3.3.4 Клас GeneticAlgorithm

Цей клас реалізує генетичний алгоритм для оптимізації ігрового середовища, такого як гра «Змійка». Він включає основні операції генетичного алгоритму, такі як оцінка популяції, вибір батьків, схрещування, мутація та управління поколіннями. Дані та методи класу наведені у таблиці 3.6.

Таблиця 3.6 – Опис полів та методів класу GeneticAlgorithm

Поля та методи класу	Опис
1	2
<i>public:</i>	
population_size	Змінна, що містить розмір популяції
genome_length	Змінна, що містить довжину геному кожної особини
generations	Змінна, що містить кількість поколінь
mutation_rate	Змінна, що містить мутації для кожного гена
population	Змінна, що містить список особин у поточній популяції
elitism_count	Змінна, що містить кількість елітних особин, які переходять до наступного покоління без змін
pool	Змінна, що містить пул процесів для паралельного обчислення пристосованості
current_generation	Змінна, що містить номер поточного покоління

Продовження таблиці 3.6

1	2
<code>__init__(self, population_size, genome_length, generations, mutation_rate=0.01, elitism_count=5)</code>	Функція, що ініціалізує генетичний алгоритм з вказаними параметрами: розміром популяції, довжиною геному, кількістю поколінь, ймовірністю мутації та кількістю елітних особин
<code>__del__(self)</code>	Функція, що закриває пул процесів після завершення роботи
<code>evaluate_population(self)</code>	Функція, що оцінює кожну особину в популяції за допомогою функції пристосованості
<code>select_parents(self, fitness_scores)</code>	Функція, що вибирає двох батьків на основі оцінок пристосованості з використанням зваженої вибірки
<code>crossover(self, parent1, parent2)</code>	Функція, що виконує схрещування між двома батьками для створення нової особини
<code>mutate(self, individual)</code>	Функція, що виконує мутацію геному особини з імовірністю, залежною від поточного покоління
<code>run(self)</code>	Функція, що запускає основний цикл генетичного алгоритму, повертає список найкращих оцінок пристосованості для кожного покоління, а також візуалізує гру з найкращою особиною

3.3.5 Клас GameEnvironment

Цей клас реалізує ігрове середовище для гри «Змійка». Він забезпечує взаємодію з симуляцією гри, надає інформацію про поточний стан гри, місцезнаходження змійки та їжі, а також дозволяє виконувати дії й отримувати відповідні винагороди. Дані та методи класу наведені у таблиці 3.7.

Таблиця 3.7 – Опис полів та методів класу GameEnvironment

Поля та методи класу	Опис
public:	
game	Змінна, що містить об'єкт симуляції гри, який використовується для управління станом гри та її оновлення
__init__(self)	Функція, що ініціалізує ігрове середовище та створює об'єкт симуляції гри
reset(self)	Функція, що скидає гру до початкового стану
get_snake_position(self)	Функція, що повертає координати голови змійки
get_snake_body(self)	Функція, що повертає координати всього тіла змійки
get_food_position(self)	Функція, що повертає координати їжі
step(self, action)	Функція, що виконує дію в грі, оновлює стан гри та повертає винагороду і статус завершення гри

3.4 Процес навчання

3.4.1 Параметри, які впливають на навчання

Розмір популяції впливає на різноманітність рішень та швидкість виконання алгоритму.

Малі розміри популяції (20–50):

- переваги: швидке виконання, менше споживання ресурсів;
- недоліки: високий ризик зупинення в локальному мінімумі.

Середні розміри популяції (100–200):

– найчастіше використовується для складних середовищ, забезпечуючи баланс між різноманітністю та швидкістю.

Великі розміри популяції (500+):

- переваги: висока різноманітність рішень;
- недоліки: значно зростає час виконання.

На рисунку 3.12 відображені результати навчання для різних розмірів популяції.

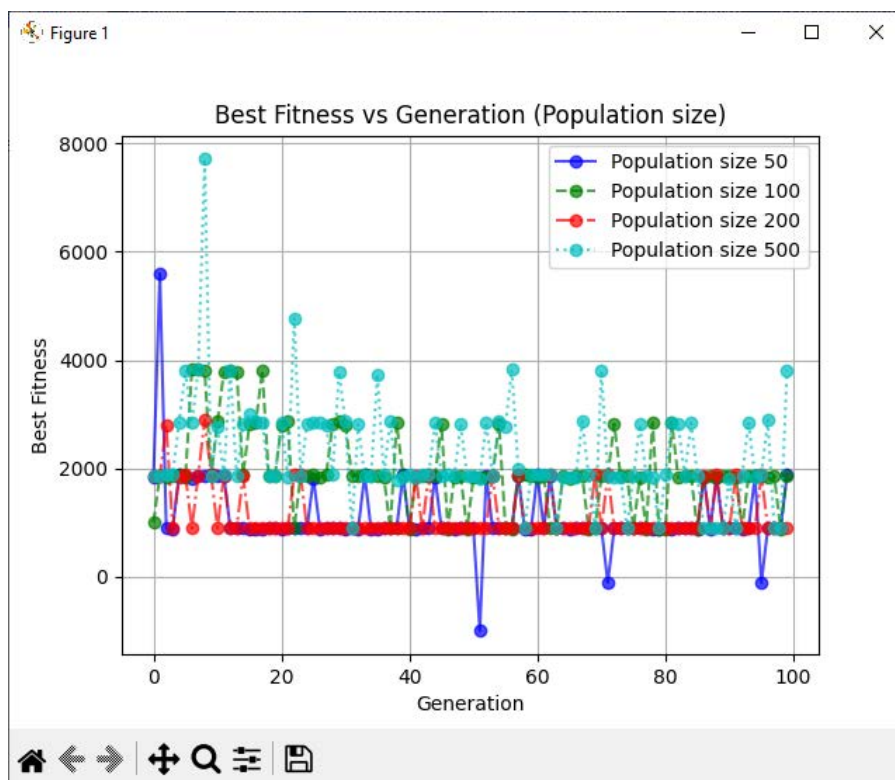


Рисунок 3.12 – Результати навчання для різних розмірів популяції

З рисунка 3.12 можна побачити, що оптимальний розмір популяції для подальших досліджень є 500. Збільшення розміру популяції до цього значення продемонструвало помітне підвищення ефективності навчання. Подальше збільшення популяції, хоча й може покращити результати, матиме значно менший приріст ефективності та суттєво збільшить час виконання алгоритму. Враховуючи ці фактори, розмір популяції 500 є оптимальним компромісом між точністю результатів і обчислювальними витратами.

Шансу мутації – цей параметр визначає, наскільки часто генетичні мутації вносяться в процесі еволюції.

Низькі значення (0.01–0.05): використовуються для стабільних алгоритмів, де потрібно зберегти більшість корисних характеристик, знайдених раніше.

Високі значення (0.1–0.3): підходять для більш хаотичних або складних середовищ, де потрібно знайти рішення в різних частинах простору пошуку. На рисунку 3.13 можна побачити результати навчання для різних коефіцієнтів мутацій.

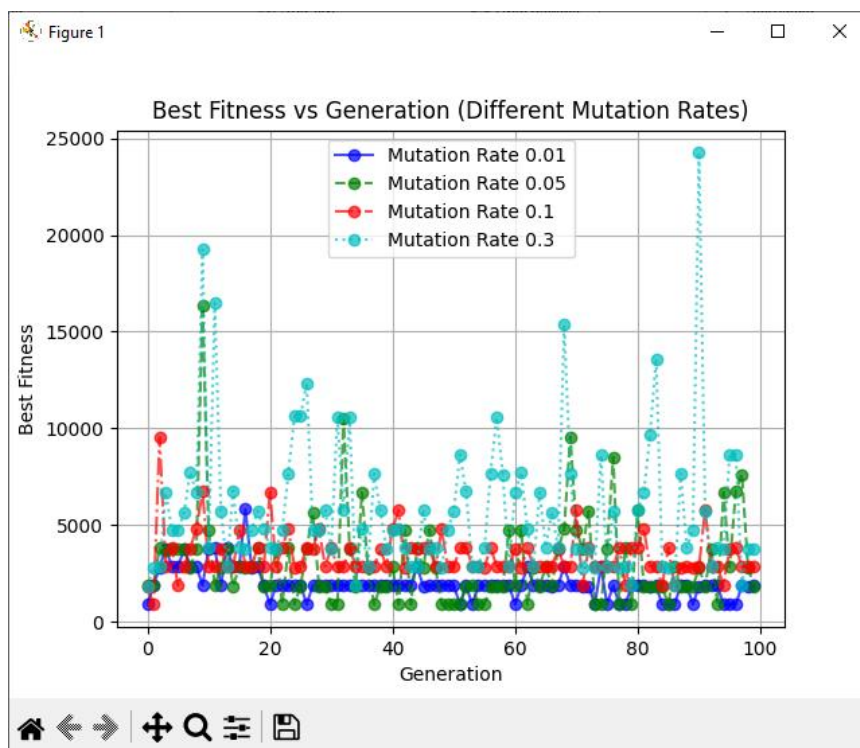


Рисунок 3.13 – Результати навчання для різних коефіцієнтів мутацій

З рисунку 3.13 видно, що оптимальний шанс мутації – 30%. Такий рівень мутації демонструє кращі результати підвищуючи різноманітність популяції.

Зміна результатів навчання по кількості елітних особин. Елітні особини зберігаються в наступному поколінні без змін, забезпечуючи збереження найкращих рішень.

Низькі значення (1–2 особини):

– підходять для алгоритмів, які потребують високого ступеня різноманітності.

Середні значення (5–10% від популяції):

– найкращий баланс між елітністю та різноманітністю.

Високі значення (20–30% від популяції):

– можуть спричинити застій, але добре працюють у стабільних середовищах.

На рисунку 3.14 можна побачити результати навчання для різної кількості елітних особин.

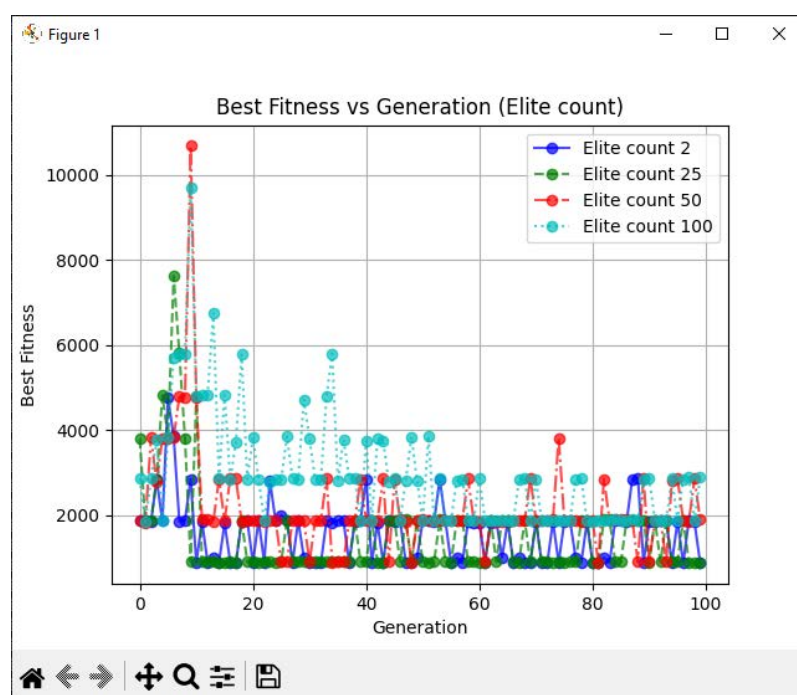


Рисунок 3.14 – Результати навчання для різної кількості елітних особин

З рисунку 3.14 видно, що оптимальна частка елітних особин – 20%. Такий вибір забезпечує збереження найкращих рішень у популяції, сприяючи швидшій збіжності, водночас підтримуючи достатню генетичну різноманітність для подальшої еволюції.

Кількості поколінь - цей параметр визначає, скільки разів алгоритм оновлює популяцію.

Малі значення (50–100):

- підходять для швидких тестів;
- недостатньо для складних задач.

Середні значення (200–500):

- найбільш підходять для балансування часу та якості рішення.

Великі значення (1000+):

- використовуються для високоточного пошуку в складних середовищах.

На рисунку 3.15 можна побачити результати навчання для різної кількості поколінь.

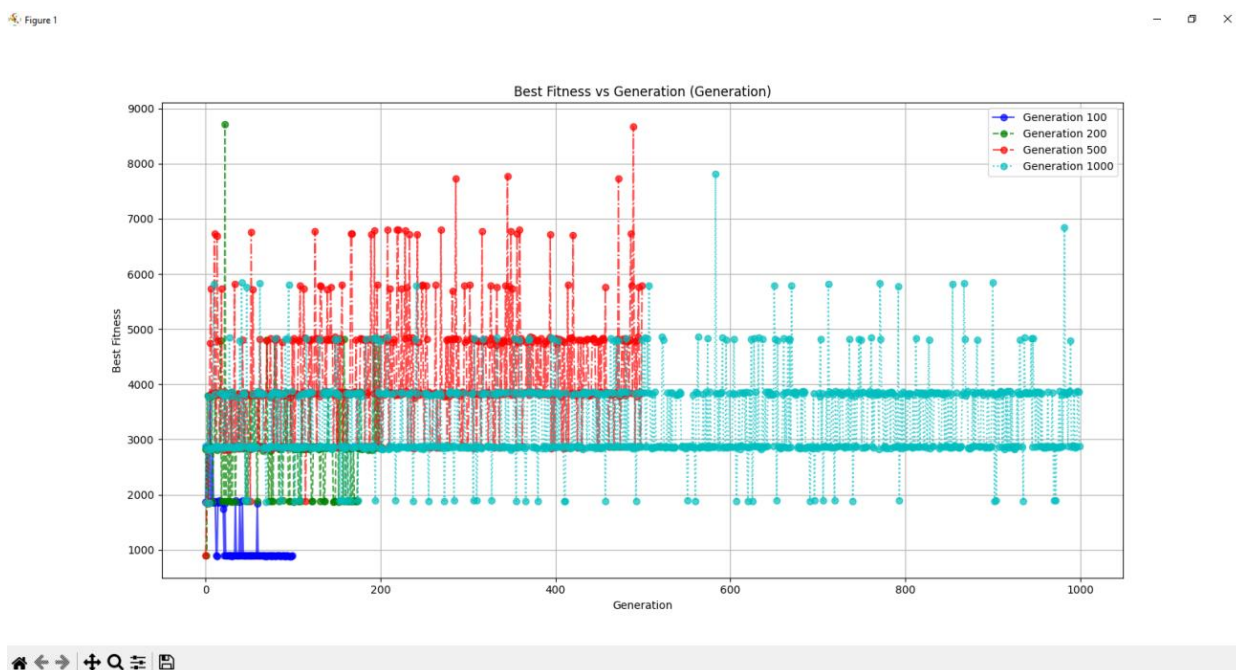


Рисунок 3.15 – Результати навчання для різної кількості елітних особин

З рисунку 3.15 видно, що оптимальна кількість поколінь – 500. Така кількість забезпечує достатній час для досягнення стабільного рівня адаптації, при цьому уникаючи надмірних витрат обчислювальних ресурсів.

Якщо інші всі параметри обирає користувач, то обчислення винагороди – параметр, який змінюється напряму втручанням у код

Приклад коду з визначенням винагороди :

```
base_reward = 1000

# Обчислення винагороди
if self.game.game_over:
    reward = base_reward * 0.1 # Штраф за завершення гри
    done = True
elif len(self.game.snake) > previous_length:
    reward = base_reward * 1.0 # Нагорода за їжу
    done = False
else:
    reward = -base_reward * 0.001 # Невеликий штраф за кожен хід
    done = False

# Винагорода/штраф за зміну відстані до їжі
distance_reward = base_reward * 0.001 if distance_after <=
distance_before else -base_reward * 0.001
```

Коли змійка програє, вона отримує штраф по очкам. Вплив:

- якщо цей штраф великий, агент буде більш обережним і прагнутиме уникати зіткнень зі стінами або власним тілом;
- якщо зменшити штраф, агент може частіше ризикувати, що може бути корисним на етапі дослідження середовища.

Нагорода за їжу, коли змійка з'їдає їжу. Вплив:

- висока нагорода за їжу мотивує агента активно рухатися до неї;
- якщо це значення занадто велике порівняно з іншими винагородами, агент може ігнорувати короткострокові цілі (наприклад, уникання стін), фокусуючись тільки на їжі.

За кожен хід змійка отримує невеликий штраф. Це створено, щоб змійка не потрапила у безкінечний цикл. Вплив:

- цей штраф заохочує агента знаходити їжу швидше;

- занадто великий штраф може змусити агента робити ризиковані рішення, щоб уникнути накопичення штрафу;
- занадто маленький або відсутній штраф може призвести до того, що агент діятиме повільно або хаотично.

Коли змійка наближається до їжі, вона отримує винагороду. Коли змійка віддаляється від їжі, отримує штраф. Вплив:

- позитивна винагорода за зменшення відстані мотивує агента рухатися в бік їжі;
- штраф за збільшення відстані змушує агента уникати «відходу» від їжі;
- якщо цей компонент має високе значення, агент може намагатися оптимізувати свою траєкторію навіть на шкоду іншим аспектам (наприклад, уникненню перешкод).

3.4.2 Результати навчання

На рисунку 3.16 можна побачити результати навчання з визначеними оптимальними параметрами.

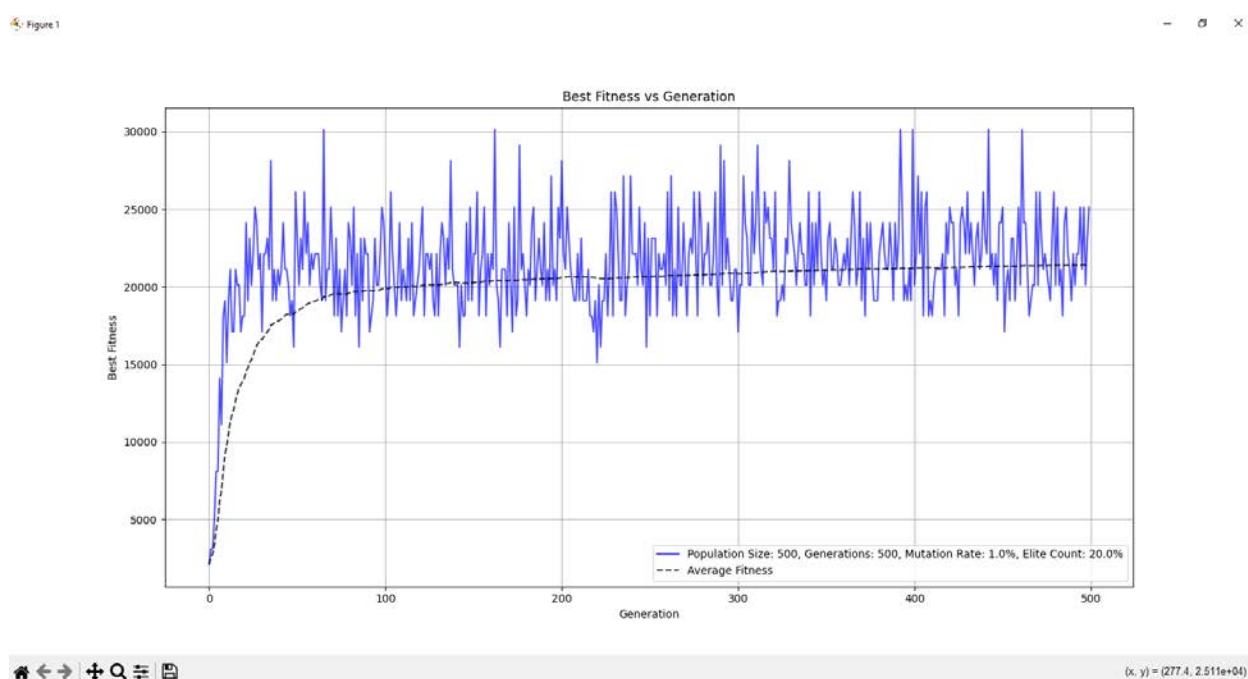


Рисунок 3.16 – Результати навчання

З рисунку 3.16 видно, що:

- прогрес на графіку, що спостерігається від низьких значень до стабільно високих, вказує на те, що модель вчиться з часом;
- коливання на графіку (особливо на початку) можуть бути пов'язані з тим, що агенти тільки починають адаптуватися до середовища, а алгоритм еволюційного навчання поступово поліпшується;
- після кількох поколінь відбувається помітне покращення, що є ознакою того, що вищі покоління мають кращу поведінку (вищу винагороду) завдяки оптимізації параметрів;
- після 200 поколінь спостерігається вихід на плато. Результати перестають помітно змінюватися.

3.5 Основні рішення щодо розробки інтерфейсу

В рамках розробки проєкту було вирішено створити консольний інтерфейс для гри. Такий підхід забезпечує простоту реалізації, високу швидкість роботи та мінімальні вимоги до ресурсів системи. Консольний інтерфейс дозволяє легко взаємодіяти з користувачем через текстові команди та виведення, що спрощує тестування й налагодження програми. Крім того, використання консольного інтерфейсу знижує складність розробки графічного інтерфейсу, зосереджуючи увагу на основній логіці гри.

Для побудови графіків та візуалізації результатів у рамках цього проєкту використовувалася бібліотека Matplotlib. Ця бібліотека є потужним інструментом для створення різноманітних графіків, що дозволяє ефективно представляти дані у візуальній формі. Використання Matplotlib значно полегшує аналіз результатів та допомагає візуалізувати динаміку змін параметрів у процесі роботи програми, зокрема в задачах, пов'язаних з генетичними алгоритмами та нейронними мережами.

З її допомогою можна легко налаштовувати елементи графіка: змінювати кольори, підписи осей, додавати легенди, анотації та інші компоненти.

Приклад вигляду графіків можна побачити на рисунку 3.17.

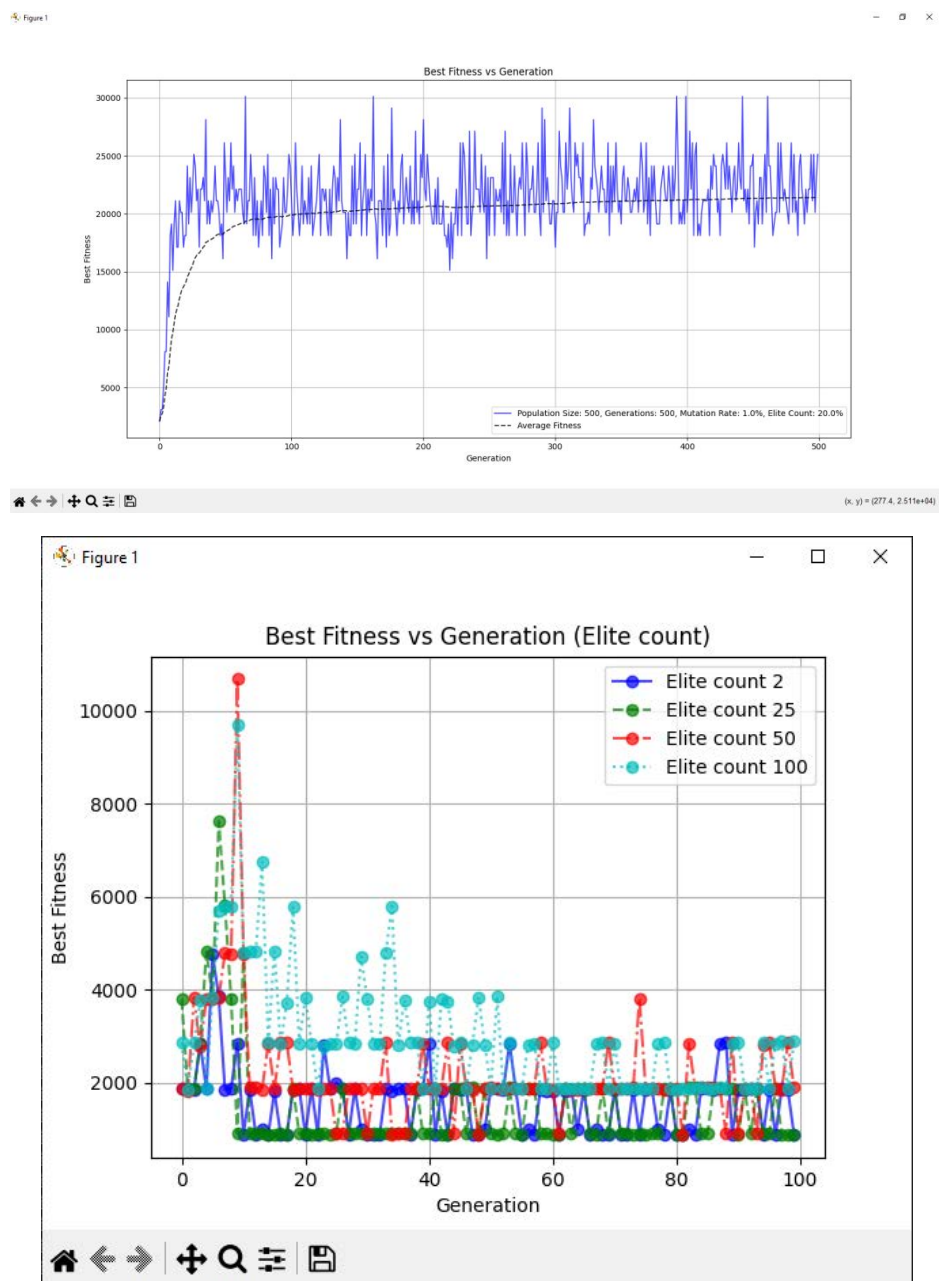


Рисунок 3.17 – Візуальний вигляд графіків

У цьому проєкті процес гри, тобто її візуалізація, були реалізовані за допомогою бібліотеки Pygame, яка є потужним інструментом для створення двовимірних ігор та інтерактивних симуляцій. Pygame надає всі необхідні

засоби для розробки графічного інтерфейсу: малювання об'єктів, обробки подій, анімацій та керування звуком.

Візуалізація гри дозволяє відображати стан програми в реальному часі, демонструючи, наприклад, рух змійки, поведінку алгоритму чи зміни в ігровому середовищі. За допомогою Pygame було створено простий і зрозумілий графічний інтерфейс, який допомагає оцінити результати роботи програми та забезпечити зручність взаємодії з користувачем.

Приклади візуалізації гри на рисунку 3.18

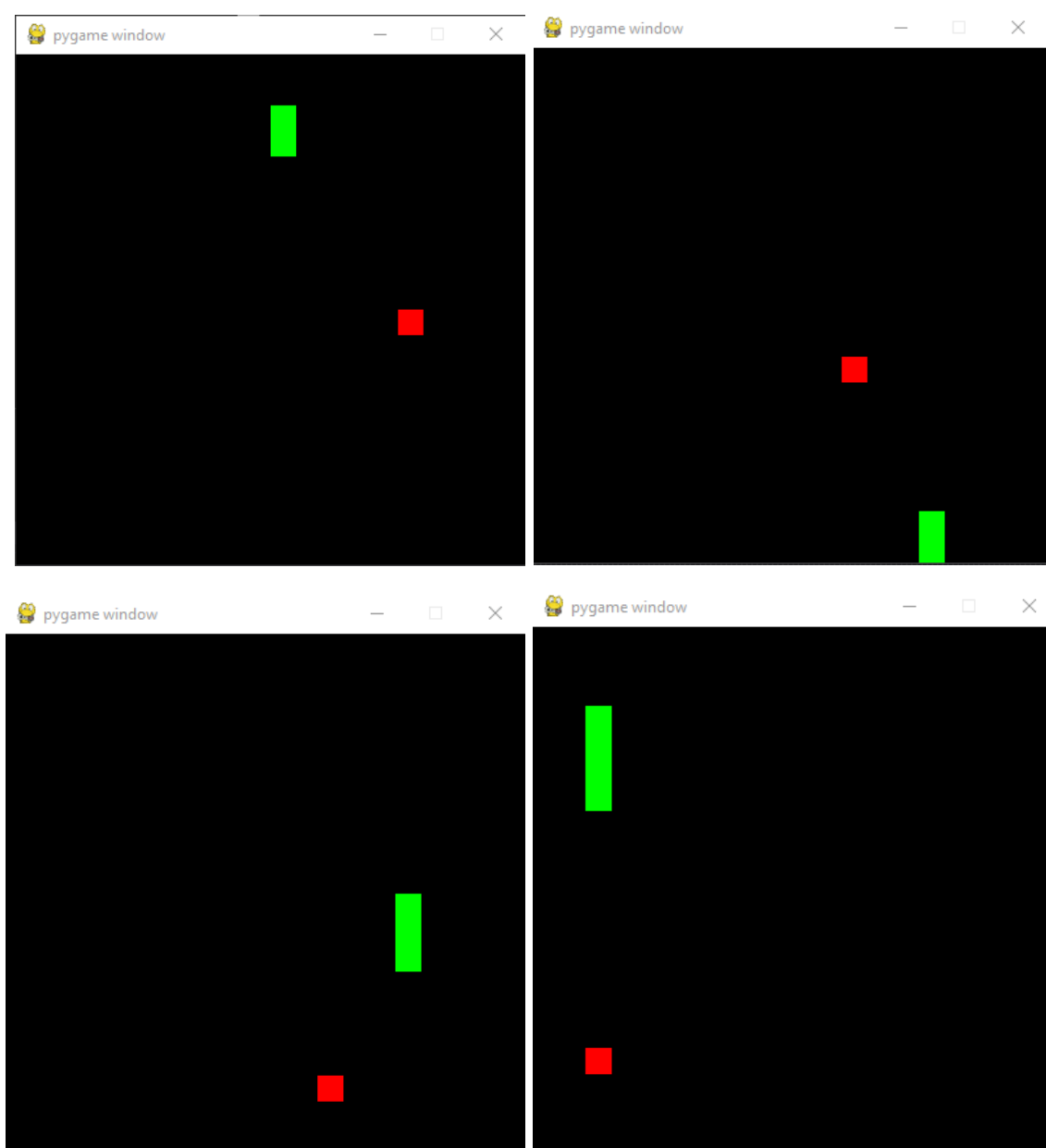


Рисунок 3.18 – Візуальний вигляд процесу гри

Таким чином, використання Pygame у поєднанні з Matplotlib для графіків забезпечило як якісну візуалізацію гри, так і ефективну демонстрацію аналітичних результатів.

3.6 Висновки за розділом 3

У цьому розділі було детально розглянуто та описано ключові компоненти розробленої програми, що включають функціональну схему, алгоритми, нейромережу, параметри навчання та інтерфейс користувача.

- створена функціональна схема дозволила чітко визначити взаємодію між різними частинами системи, забезпечивши зрозумілу та логічну структуру програмного забезпечення;

- описано низку алгоритмів, серед яких особливу увагу було приділено генетичному алгоритму, що використовує методи мутації та схрещування для пошуку оптимальних рішень у заданому ігровому середовищі. Алгоритми працюють на основі еволюційного підходу, що дозволяє знаходити ефективні стратегії гри;

- було розглянуто використання нейромережі для прийняття агентом рішення, що до наступних рухів в грі, на основі геному сформованого під час роботи генетичного алгоритму. Описано її архітектуру, параметри та процес навчання. Нейромережа дозволяє автоматизувати процес пошуку оптимальних рішень, покращуючи здатність програми адаптуватися до змінюваних умов гри;

- розглянуто параметри навчання нейромережі, а також результати цього процесу, зокрема, покращення ефективності гри з часом завдяки адаптації алгоритму до умов;

- було прийнято рішення використовувати консольний інтерфейс для взаємодії з користувачем. Цей вибір дозволяє зберегти простоту та ефективність, забезпечуючи зручність для розробника при налаштуванні та тестуванні системи.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

У цьому розділі представлено детальний аналіз призначення розробленого програмного забезпечення, умов його використання, основних функціональних можливостей, а також способів взаємодії з програмою. Розглянуто характеристики вхідних і вихідних даних, а також наведено перелік повідомлень, які застосунок може генерувати для програміста під час виконання операцій.

4.1 Призначення та умови застосування програми

4.1.1 Призначення програми

Програмне забезпечення розроблене з метою забезпечення користувача інструментами для аналізу впливу параметрів генетичного алгоритму на процес навчання агентів у симульованому ігровому середовищі гри «Змійка».

4.1.2 Функції програми

Функції, що виконує програма:

- створення графіків для аналізу ефективності навчання;
- реалізація генетичного алгоритму для пошуку оптимального геному агента «Змійки»;
- симуляція ігрового середовища для гри «Змійка»;
- візуальна симуляція ігрового середовища для гри «Змійка» для найкращих агентів;
- вибір дії у середовищі агентом, залежно від геному та інформації про оточуюче середовище, за допомоги нейромережевого алгоритму.

4.1.3 Умови застосування програми

Для забезпечення належного функціонування програмного забезпечення необхідно, щоб обране апаратне забезпечення відповідало встановленим мінімальним технічним характеристикам:

- потрібна оперативна пам'ять: 8 Гб;
- потрібна місце на жорсткому диску: 200 Мб;
- операційна система Windows 10;
- процесор Intel Core i5.

Для компіляції та внесення змін до програми рекомендується використовувати середовище Visual Studio 2022 та Python версії 3.10.

4.2 Характеристика програми

Програма створена мовою Python у середовищі Microsoft Visual Studio 2022.

Програмний продукт оснащений консольним інтерфейсом, який забезпечує можливість введення параметрів генетичного алгоритму. Крім того, передбачена функціональність для візуалізації: застосунок може відображати вікно з графічною симуляцією ігрового процесу агента, а також вікно з графічним відображенням динаміки результативності навчання.

4.2.1 Структура програми

Щоб працювати з програмою, потрібно запустити скрипт з файлу `SlytherSnake.Core.py`. Після запуску необхідне введення параметрів генетичного алгоритму у консольному інтерфейсі, після чого почнеться навчання.

Загальна структура програми складається з файлів програмного коду `.py`. Структура програми наведена на рисунку 4.1

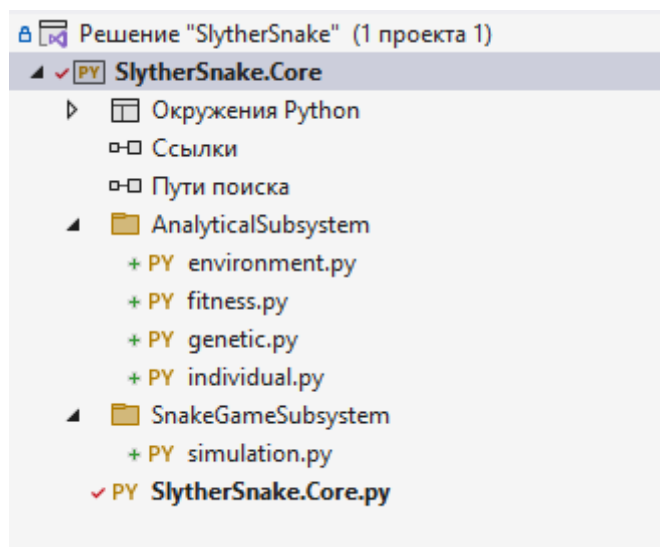


Рисунок 4.1 – Структура програми

4.3 Звертання до програми

Програмне забезпечення передбачає наступні способи взаємодії:

- використання клавіш клавіатури для введення параметрів у консольному інтерфейсі та взаємодії з графіками;
- застосування лівої кнопки комп'ютерної миші для взаємодії з графіками.

4.4 Вхідні та вихідні дані

4.4.1 Вхідні дані

Програма отримує такі вхідні дані як:

- сигнали про натискання користувачем клавіш клавіатури та миші;
- параметри генетичного алгоритму.

4.4.2 Вихідні дані

Програма видає такі вихідні дані як:

- інформація про поточне покоління та найкращий результат у ньому;
- графіки динаміки результативності навчання.

4.5 Повідомлення

Програма містить такі повідомлення як:

- «Generation 499: Best Fitness = 16117.0» – повідомлення, що містить інформацію про поточне покоління та його найкращий результат;
- «Population size (default: 500):» – запит на введення розміру популяції;
- «Mutation level (default: 0.01):» – запит на введення шансу мутації;
- «Number of elite individuals (default: 100):» – запит на введення кількості елітних особин;
- «Number of generations (default: 500):» – запит на введення розміру кількості поколінь;
- «Set default:» – означає, що поставлено значення параметра генетичного алгоритму за замовченням;
- «Error, set default:» – означає, що введено некоректні параметри, тому поставлено значення параметра генетичного алгоритму за замовченням.

5 КЕРІВНИЦТВО ОПЕРАТОРА

У цьому розділі здійснено аналіз основного призначення програмного забезпечення, умов його функціонування, а також процесу його виконання. Окрім того, розглянуті повідомлення, які застосунок може генерувати для користувача в ході роботи.

5.1 Призначення програми

Програма призначена для забезпечення користувача інструментами для аналізу впливу параметрів генетичного алгоритму на процес навчання агентів у симульованому ігровому середовищі гри «Змійка»

5.2 Умови виконання програми

5.2.1 Апаратні вимоги програми

Для забезпечення належного функціонування програмного забезпечення необхідно, щоб обране апаратне забезпечення відповідало встановленим мінімальним технічним характеристикам:

- потрібна оперативна пам'ять: 8 Гб;
- потрібна місце на жорсткому диску: 200 Мб;
- операційна система Windows 10;
- процесор Intel Core i5.

5.2.2 Вимоги до операторів

Для ефективного використання програмного забезпечення від користувача вимагаються такі компетенції:

- знання англійської мови на рівні, достатньому для розуміння інтерфейсу;

- практичні навички роботи з клавіатурою та комп'ютерною мишею;
- базові теоретичні знання у галузі штучного інтелекту та генетичних алгоритмів;
- досвід взаємодії з консольним інтерфейсом.

5.3 Виконання програми

5.3.1 Запуск програми

Програма запускається за допомогою скрипту , що знаходиться у файлі SlytherSnake.Core.py, який є основним запусковим елементом системи.

5.3.2 Виконання роботи з програмою

При запуску програми користувач бачить консольне вікно на рисунку 5.1.

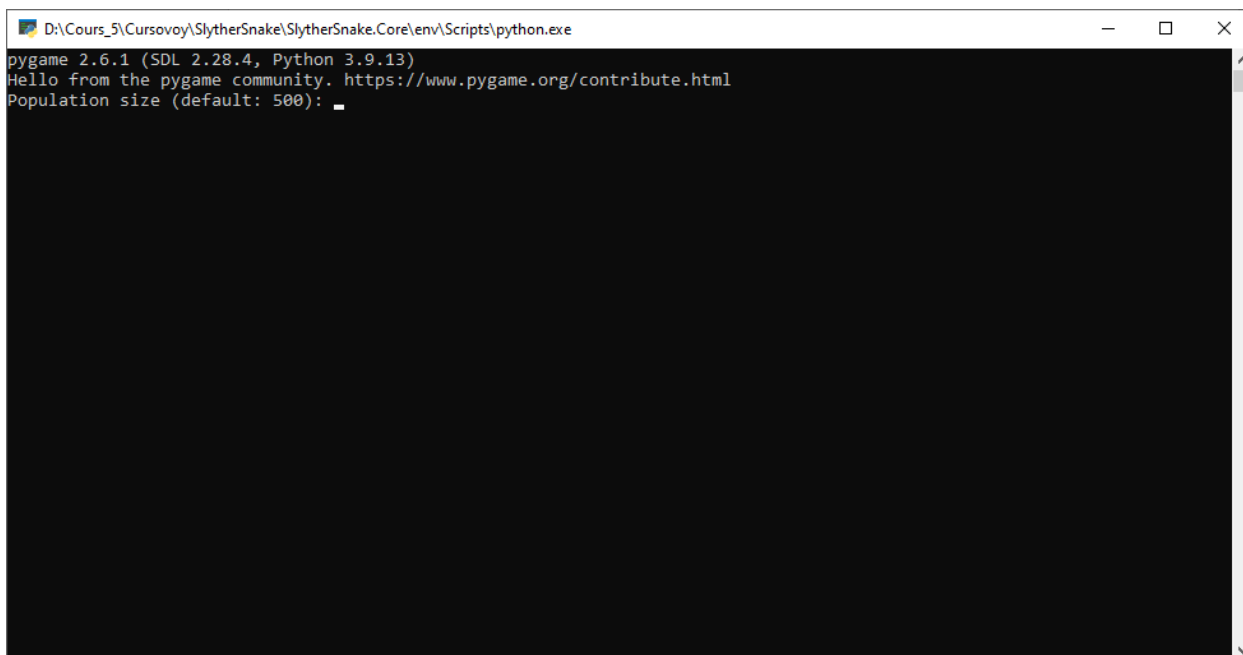
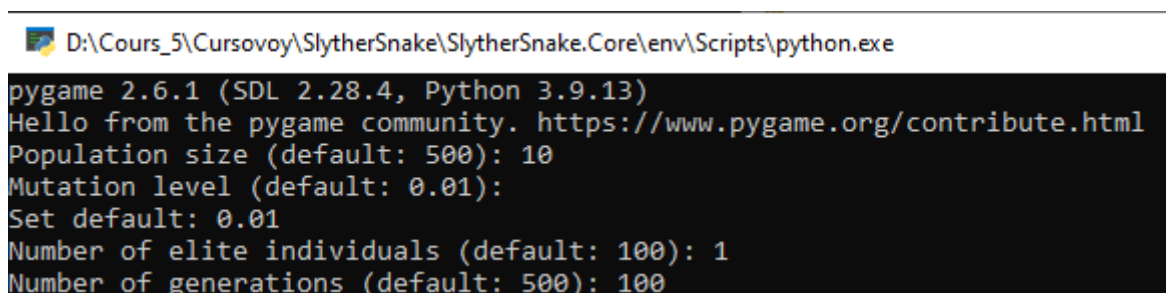


Рисунок 5.1 – Старт роботи програми

Після початку роботи програми користувачу необхідно ввести такі параметри як:

- розмір популяції;
- шанс мутації;
- кількість елітних осіб;
- кількість поколінь.

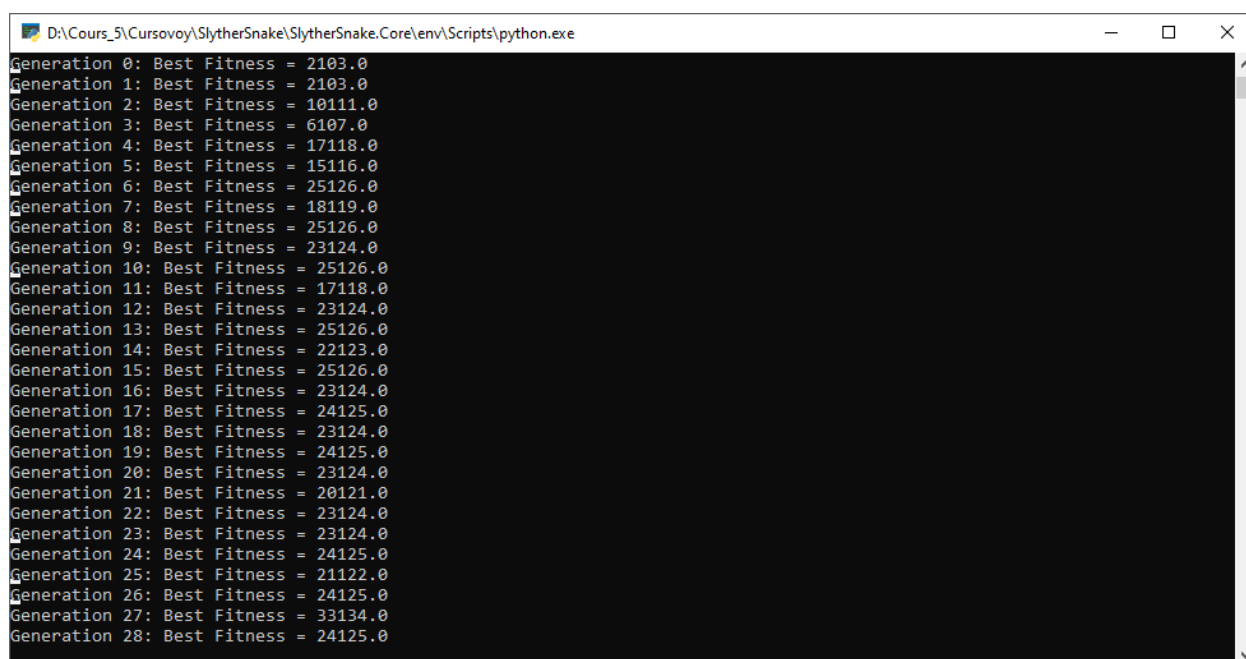
На рисунку 5.2 можна побачити приклад цієї дії



```
D:\Cours_5\Cursovoy\SlytherSnake\SlytherSnake.Core\env\Scripts\python.exe
pygame 2.6.1 (SDL 2.28.4, Python 3.9.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
Population size (default: 500): 10
Mutation level (default: 0.01):
Set default: 0.01
Number of elite individuals (default: 100): 1
Number of generations (default: 500): 100
```

Рисунок 5.2 – Ввід параметрів

Далі на консолі відображається у консолі найкращий результат у кожному поколінні. Це як раз показано на рисунку 5.3.



```
D:\Cours_5\Cursovoy\SlytherSnake\SlytherSnake.Core\env\Scripts\python.exe
Generation 0: Best Fitness = 2103.0
Generation 1: Best Fitness = 2103.0
Generation 2: Best Fitness = 10111.0
Generation 3: Best Fitness = 6107.0
Generation 4: Best Fitness = 17118.0
Generation 5: Best Fitness = 15116.0
Generation 6: Best Fitness = 25126.0
Generation 7: Best Fitness = 18119.0
Generation 8: Best Fitness = 25126.0
Generation 9: Best Fitness = 23124.0
Generation 10: Best Fitness = 25126.0
Generation 11: Best Fitness = 17118.0
Generation 12: Best Fitness = 23124.0
Generation 13: Best Fitness = 25126.0
Generation 14: Best Fitness = 22123.0
Generation 15: Best Fitness = 25126.0
Generation 16: Best Fitness = 23124.0
Generation 17: Best Fitness = 24125.0
Generation 18: Best Fitness = 23124.0
Generation 19: Best Fitness = 24125.0
Generation 20: Best Fitness = 23124.0
Generation 21: Best Fitness = 20121.0
Generation 22: Best Fitness = 23124.0
Generation 23: Best Fitness = 23124.0
Generation 24: Best Fitness = 24125.0
Generation 25: Best Fitness = 21122.0
Generation 26: Best Fitness = 24125.0
Generation 27: Best Fitness = 33134.0
Generation 28: Best Fitness = 24125.0
```

Рисунок 5.3 – Вивід найкращого результату кожного покоління

Також створено окреме вікно, де відбувається візуалізації самої гри. Приклад такого вікна можна побачити на рисунку 5.4.

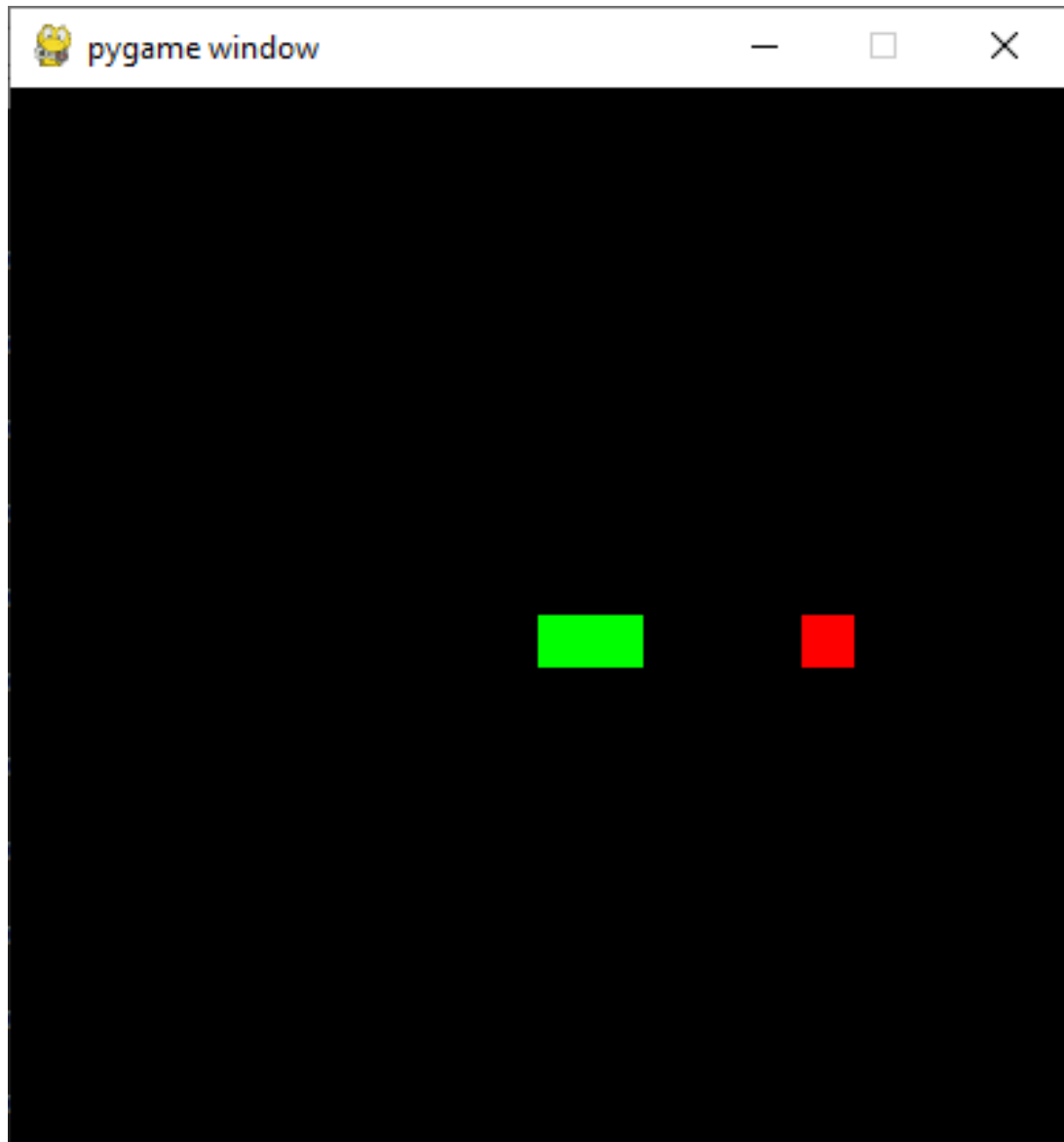


Рисунок 5.4 – Візуальна симуляція гри

Також після навчання кожного покоління відображаються графіки, які допомагають зрозуміти вплив кожного параметра на агента. Приклад графіка представлено на рисунку 5.5

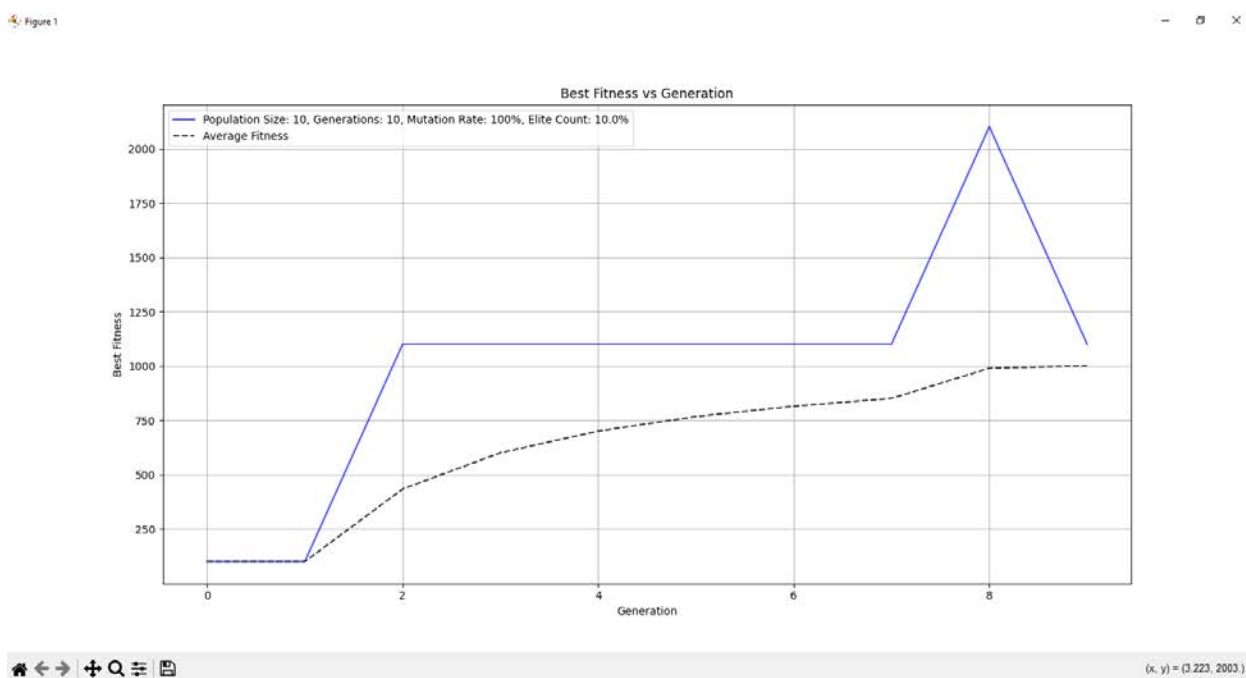


Рисунок 5.5 – Приклад діаграми

5.4 Повідомлення оператора

Програма містить такі повідомлення як:

- «Generation 499: Best Fitness = 16117.0» – повідомлення, що містить інформацію про поточне покоління та його найкращий результат;
- «Population size (default: 500):» – запит на введення розміру популяції;
- «Mutation level (default: 0.01):» – запит на введення шансу мутації;
- «Number of elite individuals (default: 100):» – запит на введення кількості елітних особин;
- «Number of generations (default: 500):» – запит на введення розміру кількості поколінь;
- «Set default:» – означає, що поставлено значення параметра генетичного алгоритму за замовченням;

- «Error, set default:» – означає, що введено некоректні параметри, тому поставлено значення параметра генетичного алгоритму за замовченням.

6 ТЕСТУВАННЯ І РЕЗУЛЬТАТИ

6.1 Опис методики проведення тестування

Тестування буде виконано згідно з наступним сценарієм:

а) основний скрипт:

- 1) введення коректних параметрів;
- 2) введення некоректних параметрів;
- 3) введення порожніх параметрів;
- 4) запуск генетичного алгоритму;
- 5) створення графіку динаміки навчання.

б) генетичний алгоритм:

- 1) створення початкового покоління;
- 2) оцінка поточної популяції;
- 3) застосування елітизму;
- 4) застосування мутації;
- 5) застосування схрещення;
- 6) вивід найкращого результату покоління в консоль.

в) симуляція гри:

- 1) обробка руху змійки;
- 2) обробка зміни напрямку змійки;
- 3) обробка збору їжі змійкою;
- 4) обробка зіткнення з стінкою;
- 5) обробка зіткнення з хвостом змійки;
- 6) обробка кінця симуляції.

г) візуальна симуляція гри:

- 1) обробка руху змійки;
- 2) обробка зміни напрямку змійки;
- 3) обробка збору їжі змійкою;
- 4) обробка зіткнення з стінкою;
- 5) обробка зіткнення з хвостом змійки;

- 6) обробка кінця симуляції;
- 7) відображення вікна гри;
- 8) оновлення вікна гри.

Для забезпечення коректного проведення тестування необхідно, щоб апаратне забезпечення, яке буде використовуватися, відповідало визначеним мінімальним технічним характеристикам:

- потрібна оперативна пам'ять: 8 Гб;
- потрібна місце на жорсткому диску: 200 Мб;
- операційна система Windows 10;
- процесор Intel Core i5.

6.2 Результати використання розробленого програмного забезпечення

У таблиці 6.1 представлено тестування програмного забезпечення.

Таблиця 6.1 – Тестування ПЗ

№	Назва	Опис результату	Оцінка тесту
1	2	3	4
Основний скрипт			
1	Введення коректних параметрів	Введені параметри успішно прийняті	Пройшов успішно
2	Введення некоректних параметрів	Введені параметри відхилені, виставлені параметри за замовченням, про це відправлено повідомлення у консоль	Пройшов успішно
3	Введення порожніх параметрів	Виставлені параметри за замовченням	Пройшов успішно
4	Запуск генетичного алгоритму	Алгоритм ініційовано для виконання	Пройшов успішно

Продовження таблиці 6.1

1	2	3	4
5	Створення графіку динаміки навчання	Графік успішно створено і відображено	Пройшов успішно
Генетичний алгоритм			
1	Створення початкового покоління	Покоління успішно згенеровано	Пройшов успішно
2	Оцінка поточної популяції	Оцінка поточної популяції успішно відбулась	Пройшов успішно
3	Застосування елітизму	Найкращі особини скопійовані у наступне покоління без змін	Пройшов успішно
4	Застосування мутації	Геном успішно мutowано	Пройшов успішно
5	Застосування схрещення	Геноми успішно схрещено	Пройшов успішно
6	Вивід найкращого результату покоління в консоль	Виведено результат покоління у консоль	Пройшов успішно
Симуляція гри			
1	Обробка руху змійки	Рух змійки успішно оброблено	Пройшов
2	Обробка зміни напрямку змійки	Зміна напрямку змійки успішно оброблена	Пройшов успішно
3	Обробка збору їжі змійкою	Збір їжі змійкою успішно оброблено, довжина змійки збільшилась, згенерована нова їжа	Пройшов успішно
4	Обробка зіткнення з стінкою	Гра успішно завершена	Пройшов успішно

Продовження таблиці 6.1

1	2	3	4
5	Обробка зіткнення з хвостом змійки	Гра успішно завершена	Пройшов успішно
6	Обробка кінця симуляції	Симуляція успішно завершена	Пройшов успішно
Візуальна симуляція гри			
1	Обробка руху змійки	Рух змійки успішно оброблено	Пройшов успішно
2	Обробка зміни напрямку змійки	Зміна напрямку змійки успішно оброблена	Пройшов успішно
3	Обробка збору їжі змієюю	Збір їжі змієюю успішно оброблено, довжина змійки збільшилась, згенерована нова їжа	Пройшов успішно
4	Обробка зіткнення з стінкою	Гра успішно завершена	Пройшов успішно
5	Обробка зіткнення з хвостом змійки	Гра успішно завершена	Пройшов успішно
6	Обробка кінця симуляції	Симуляція успішно завершена	Пройшов успішно
7	Відображення вікна гри	Вікно коректно відображено	Пройшов успішно
8	Оновлення вікна гри	Вікно коректно оновлює зображення	Пройшов успішно

6.3 Результати тестування генетичного алгоритму

Генетичний алгоритм демонструє високі результати з урахуванням використаних апаратних ресурсів та обмеженого часу навчання. На

початкових етапах обробки поколінь спостерігається стрімке підвищення якості ігрових стратегій агентів, що швидко досягають високого рівня ефективності, як це відображено на рисунку 6.1. Така динаміка підтверджує коректність функціонування алгоритму та його здатність ефективно оптимізувати поведінку агентів у заданому середовищі.

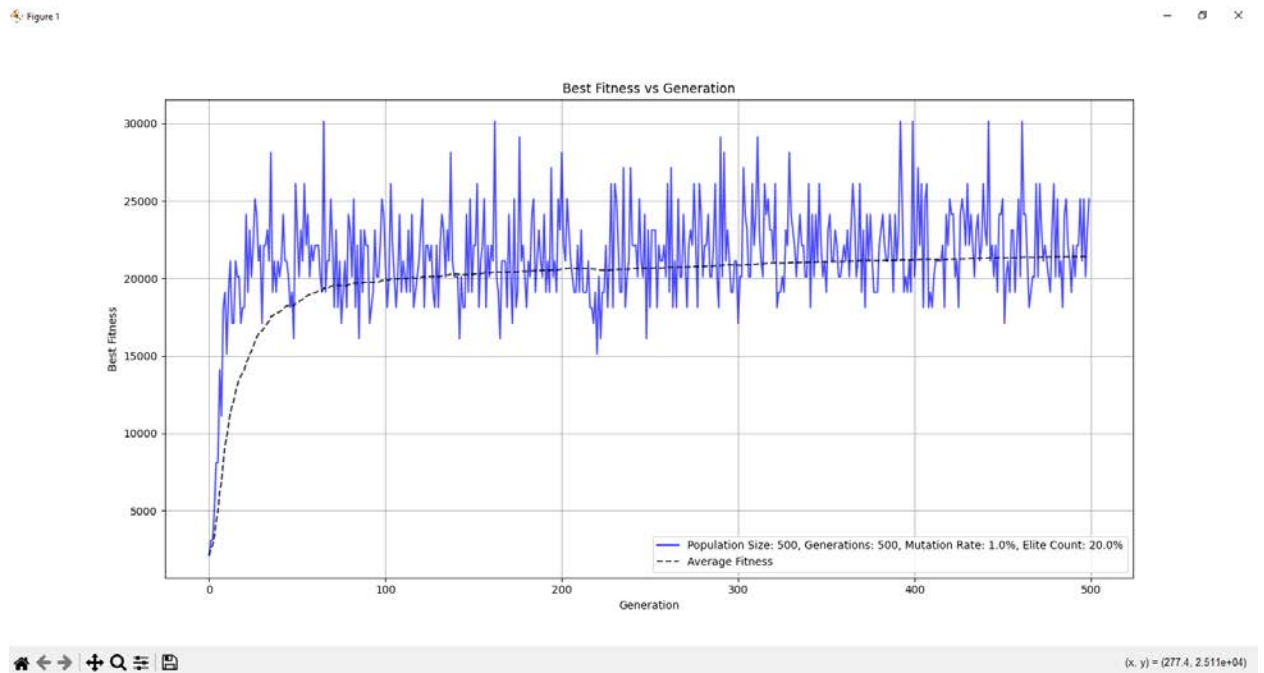


Рисунок 6.1 – Результат тестування генетичного алгоритму

ВИСНОВКИ

Під час виконання дипломної роботи було використано інтегроване середовище розробки Microsoft Visual Studio 2022 та мову програмування Python версії 3.10. Основною метою дослідження стала розробка клієнт-застосунку «SlytherSnake», призначеного для вивчення можливостей застосування генетичних алгоритмів у створенні ефективної стратегії поведінки в ігровому середовищі.

У першому розділі роботи проведено аналіз предметної області та здійснено огляд існуючих підходів до розв'язання аналогічних завдань. Було виконано експериментальний-порівняльний аналіз методів, таких як генетичні алгоритми в поєднанні з нейронними мережами, глибоке навчання з підкріпленням (DRL), метод Монте-Карло, евристичні алгоритми та алгоритм Q-Learning. В результаті було з'ясовано, що найбільш результативним виявився евристичний алгоритм, найкраще навчав метод DRL, але враховуючи ресурси витрачені на навчання та написання алгоритму найефективнішим виявився генетичний алгоритм в поєднанні з нейронною мережею. Крім того, було досліджено наявні програмні засоби для вирішення подібних завдань та обрано прототип для реалізації. У другому розділі описано вибір програмних інструментів, використаних під час розробки, та розроблено структурну схему ПЗ. Третій розділ присвячено детальному опису процесу реалізації програмного продукту. Зокрема, наведено опис створених класів і алгоритмів, консольного інтерфейсу, а також архітектури нейронної мережі, яка використовується в системі. Також була описана реалізація, обраного і модифікованого, під гру змійка, генетичного алгоритму. У четвертому та п'ятому розділах розроблено інструкції для користувачів та операторів щодо використання програмного продукту. У шостому розділі проведено тестування розробленої системи, описано процес перевірки її функціональності та представлено чек-лист виконаних тестів.

Результатом виконання роботи стало створення програмного

продукту, який надає можливість ініціалізації процесу навчання з варіативними параметрами генетичного алгоритму. Також забезпечено виведення графічного відображення динаміки навчання та реалізовано візуальну симуляцію ігрового процесу відповідно до правил гри «Змійка», що демонструє результати роботи навченого агента.

У процесі виконання були враховані вимоги до адаптивності, точності та ефективності розробленого алгоритму, що дозволило отримати результати, які можуть бути корисними для подальшого розвитку методів штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Holland, J. Adaptation in Natural and Artificial Systems / J. Holland. – Cambridge: MIT Press, 2019. – 228 p.
2. Yannakakis, G. Artificial Intelligence and Games / G. Yannakakis, J. Togelius. – Cham: Springer, 2018. – 376 p.
3. Deb, K. Optimization for Engineering Design: Algorithms and Examples / K. Deb. – Hoboken: Wiley, 2016. – 521 p.
4. Russell, S. Artificial Intelligence: A Modern Approach / S. Russell, P. Norvig. – New York: Pearson, 2020. – 1152 p.
5. Back, T. Evolutionary Computation 1: Basic Algorithms and Operators / T. Back, D. Fogel, Z. Michalewicz. – Boca Raton: CRC Press, 2018. – 593 p.
6. Wooldridge, M. An Introduction to MultiAgent Systems / M. Wooldridge. – Chichester: Wiley, 2020. – 484 p.
7. Sutton, R., Barto, A. Reinforcement Learning: An Introduction / R. Sutton, A. Barto. – Cambridge: MIT Press, 2018. – 552 p.
8. Genetic Algorithms vs Neural Networks [Electronic resource]. – Access mode: <https://www.baeldung.com/cs/genetic-algorithms-vs-neural-networks>.
9. A Beginner's Guide to Deep Reinforcement Learning [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/a-beginners-guide-to-deep-reinforcement-learning/>.
10. ML | Monte Carlo Tree Search (MCTS) [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/>.
11. Heuristic Search Techniques in AI [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/heuristic-search-techniques-in-ai/>.
12. Q-Learning vs. Deep Q-Learning vs. Deep Q-Network [Electronic resource]. – Access mode: <https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network>.

13. OpenAI Gym Beta [Electronic resource]. – Access mode: <https://openai.com/index/openai-gym-beta/>.
14. Getting Started With OpenAI Gym: The Basic Building Blocks [Electronic resource]. – Access mode: <https://www.digitalocean.com/community/tutorials/getting-started-with-openai-gym>.
15. gym-snake [Electronic resource]. – Access mode: <https://github.com/grantsrb/Gym-Snake>.
16. Deep Q-Learning (DQN) [Electronic resource]. – Access mode: <https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae>.
17. SnakeQ [Electronic resource]. – Access mode: https://github.com/0xArete/SnakeQ?utm_source=chatgpt.com.
18. Neuroevolution: Evolving Neural Network with Genetic Algorithms [Electronic resource]. – Access mode: <https://medium.com/@roopal.tatiwar20/neuroevolution-evolving-neural-network-with-genetic-algorithms-8ca2165ad04c>.
19. Snake-neuroevolution [Electronic resource]. – Access mode: <https://github.com/tadeaspaule/snake-neuroevolution>.
20. Octoverse 2024 GitHub [Electronic resource]. – Access mode: <https://github.blog/news-insights/octoverse/octoverse-2024/>.
21. The Python Programming Language [Electronic resource]. – Access mode: <https://www.tiobe.com/tiobe-index/python/>.
22. Рейтинг мов програмування 2024. TypeScript в трійці лідерів, Python з'являється у всіх нішах, а Rust — улюблена мова [Електронний ресурс]. – Режим доступу: <https://dou.ua/lenta/articles/language-rating-2024/>.
23. What is JavaScript? [Electronic resource]. – Access mode: https://developer.mozilla.org/What_is_JavaScript.
24. C++ Introduction [Electronic resource]. – Access mode: https://www.w3schools.com/cpp/cpp_intro.asp.

25. Introduction to Java [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/introduction-to-java/>
26. History of Python [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/history-of-python/>.
27. The History of Python Programming [Electronic resource]. – Access mode: <https://www.bocasay.com/history-python-programming/>.
28. Sweigart, A. Automate the Boring Stuff with Python / A. Sweigart. – San Francisco: No Starch Press, 2015. – 504 p.
29. Van Rossum, G. Python Programming Language / G. Van Rossum. – Beaverton: Python Software Foundation, 2020. – 352 p.
30. The Most Popular IDEs for Developers in 2025 [Electronic resource]. – Access mode: <https://outstaffyourteam.com/articles/most-popular-ides-for-developers>.
31. The Python IDE for data and web professionals [Electronic resource]. – Access mode: <https://www.jetbrains.com/pycharm/>.
32. Why did we build Visual Studio Code? [Electronic resource]. – Access mode: <https://code.visualstudio.com/docs/editor/whyvscode>.
33. Microsoft Visual Studio [Electronic resource]. – Access mode: <https://visualstudio.microsoft.com/>.
34. History and Evolution of Visual Studio and Supported .NET Versions [Electronic resource]. – Access mode: <https://www.c-sharpcorner.com/article/history-and-evolution-of-visual-studio-and-supported-net-versions/>.
35. What is Visual Studio [Electronic resource]. – Access mode: <https://www.arimetrics.com/en/digital-glossary/visual-studio>.
36. Glossary [Electronic resource]. – Access mode: <https://docs.python.org/3/glossary.html#term-library>.
37. Pygame Front Page [Electronic resource]. – Access mode: <https://www.pygame.org/docs/>.

38. <https://realpython.com/pygame-a-primer/> PyGame: A Primer on Game Programming in Python [Electronic resource]. – Access mode:
39. Matplotlib 3.10.0 documentation [Electronic resource]. – Access mode: <https://matplotlib.org/stable/index.html>.
40. Python Plotting With Matplotlib (Guide) [Electronic resource]. – Access mode: <https://realpython.com/python-matplotlib-guide/>.
41. NumPy Documentation [Electronic resource]. – Access mode: <https://numpy.org/doc/>.
42. What is NumPy? [Electronic resource]. – Access mode: <https://numpy.org/doc/stable/user/whatisnumpy.html>.
43. multiprocessing — Process-based parallelism [Electronic resource]. – Access mode: <https://docs.python.org/3/library/multiprocessing.html>.
44. Simon, D. Evolutionary Optimization Algorithms / D. Simon. – Hoboken: Wiley, 2016. – 772 p.
45. Eiben, A.E. Introduction to Evolutionary Computing / A.E. Eiben, J.E. Smith. – 2nd ed. – Springer, 2015. – 300 p.

ДОДАТОК А
Технічне завдання

A.1 Вступ

У сучасному світі комп'ютерні ігри стали невід'ємною частиною розвитку технологій та розваг. Однією з найвідоміших класичних ігор є «Змійка» – проста за своєю концепцією, але водночас захоплююча гра, що вже багато років залишається популярною серед різних поколінь гравців. Її основна ідея полягає в управлінні змією, яка переміщується по ігровому полю, збираючи об'єкти та збільшуючи свою довжину, уникаючи зіткнень зі стінами чи власним тілом.

Головною метою розробки є створення інтелектуальної версії гри, яка не лише забезпечує цікавість і динаміку геймплею, а й використовує сучасні алгоритми оптимізації та штучного інтелекту для вдосконалення процесу гри.

A.1.1 Назва програми

Назва програми: «SlytherSnake».

A.1.2 Призначення та область застосування

Програма призначена для забезпечення користувача інструментами для аналізу впливу параметрів генетичного алгоритму на процес навчання агентів у симульованому ігровому середовищі гри «Змійка»

Область застосування: приватні особи.

A.2 Підстави для розробки

Проведення розробки затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

A.3 Призначення розробки

Основне призначення розробки полягає у створенні вдосконаленої версії класичної гри «Змійка», яка поєднує в собі традиційний ігровий процес із сучасними підходами до оптимізації та інтеграції штучного інтелекту.

Ця розробка має на меті:

- надати користувачам захоплюючий ігровий досвід;
- продемонструвати можливості використання генетичних алгоритмів і нейронних мереж у процесі автоматизації та вдосконалення гри;
- створити базу для досліджень у галузі алгоритмів оптимізації та навчання ШІ в інтерактивних середовищах.
- Проєкт орієнтований на ентузіастів програмування, дослідників у галузі штучного інтелекту, студентів, а також усіх, хто цікавиться інтеграцією новітніх технологій у сферу ігрової індустрії.

A.4 Вимоги до програми

A.4.1 Вимоги до функціональних характеристик

Основні функціональні характеристики:

- створення графіків для аналізу ефективності навчання;
- реалізація генетичного алгоритму для пошуку оптимального геному агента «Змійки»;
- симуляція ігрового середовища для гри «Змійка»;
- візуальна симуляція ігрового середовища для гри «Змійка» для найкращих агентів;
- вибір дії у середовищі агентом, залежно від геному та інформації про оточуюче середовище, за допомоги нейромережевого алгоритму.

A.4.2 Вимоги до надійності

Щоб забезпечити коректну роботу програмного забезпечення, обране апаратне забезпечення повинно відповідати визначеним мінімальним технічним вимогам:

- алгоритми, зокрема генетичні, нейромережеві, а також модулі обробки даних, повинні бути ретельно протестовані для запобігання критичних помилок, які можуть вплинути на функціональність гри;
- інтерфейс і візуальні елементи повинні працювати коректно без графічних артефактів або зависань;
- бібліотеки, що використовуються для обробки даних мають забезпечувати стабільну роботу програми навіть за високого навантаження;
- система повинна пройти багатоетапне тестування.
- Виконання цих вимог гарантує, що програма буде працювати стабільно, надійно та відповідати сучасним стандартам якості.

A.4.3 Вимоги до технічних та програмних засобів

Для коректної роботи з готовим програмним забезпеченням потрібно, щоб пристрій, який буде використовуватися, відповідав мінімальним технічним вимогам:

- потрібна оперативна пам'ять: 8 Гб;
- потрібна місце на жорсткому диску: 200 Мб;
- операційна система Windows 10;
- процесор Intel Core i5.

A.5 Вимоги до програмної документації

Програмна документація повинна містити:

- технічне завдання;

- пояснювальна записка.

A.6 Стадії і етапи розробки

A.6.1 Стадії розробки

Основні стадії розробки:

- аналіз вимог до розроблюваного програмного забезпечення;
- формування технічного завдання для проєкту;
- проєктування системи програмного забезпечення;
- створення розроблюваного програмного забезпечення;
- тестування створеного програмного забезпечення;
- оформлення необхідної програмної документації.

A.6.2 Етапи розробки та зміст роботи на кожному етапі

Етапи розробки та зміст роботи на кожному етапі:

- аналіз предметної області — дослідження актуальності тематики роботи, основних понять, пов'язаних з грою «Змійка», а також правил гри;
 - огляд існуючих методів вирішення завдання — огляд методів, що використовуються для знаходження оптимальних стратегій у складних ігрових середовищах, таких як гра «Змійка»;
 - огляд існуючих програмних засобів, що вирішують аналогічні завдання — аналіз аналогічних програмних рішень, виконано їх порівняння, виявлення переваг й недоліків для вибору оптимального прототипу;
 - визначення вимог до програмного продукту — опис необхідної функціональності, характеристик та обмежень, які мають бути враховані під час розробки програмного забезпечення;
 - визначення вимог до надійності — встановлення та формулювання вимог та критеріїв, які забезпечують безвідмовну роботу програмного продукту, його стабільну продуктивність та очікувану функціональність;

- визначення вимог до технічних і програмних засобів – складання вимог до технічних характеристик апаратного забезпечення та програмних засобів для розробки подальшої експлуатації системи;
- визначення вимог до програмної документації – визначення та опис склад супровідної документації;
- проектування структури застосунку – визначення архітектурної структури, яка включає основні компоненти, модулі та взаємозв'язки між ними;
- проектування структури нейромережі – визначення типу, архітектури та параметрів нейронної мережі, яка інтегрується в систему для прийняття рішень;
- розробка застосунку – створення консольного інтерфейсу та реалізація основної програмної логіки, що забезпечує навчання генетичним алгоритмом і відображення результатів;
- розробка симуляції ігрового середовища по правилах гри «Змійка» для навчання агентів – створення симуляції гри «Змійка» у візуальному та невізуальному режимах для навчання агентів і демонстрації їх результатів;
- розробка нейромережі для прийняття рішень – створення нейромережі, яка інтегрується із симуляційним середовищем для прийняття рішень агентами;
- завершення розробки програмного продукту – інтеграція всіх компонентів системи та забезпечення їх коректної взаємодії;
- тестування програмного забезпечення – тестування функціональності, продуктивності, надійності та відповідності вимогам програмного продукту;
- виправлення знайдених проблем – виправлення помилок, виявлених під час тестування, з метою підвищення якості та стабільності системи;

- оформлення опису програми – створення технічної документації, яка детально описує функціональність, архітектуру та використані технологій й інші аспекти роботи програмного продукту;
- завершення розробки пояснювальної записки – оформлення остаточного варіанту пояснювальної записки, який охоплює всі етапи проєкту, їх результати та детальний аналіз виконаної роботи.

ДОДАТОК Б
Текст програми

Б.1 Файл SlytherSnake.Core.py

```

from AnalyticalSubsystem.genetic import GeneticAlgorithm
import numpy as np
import matplotlib.pyplot as plt
import pygame

if __name__ == '__main__':
    pygame.init()

    # Списки можливих параметрів
    population_sizes = [50, 100, 200, 500]
    mutation_rates = [0.01, 0.05, 0.1, 0.3]
    elite_counts = [2, 25, 50, 100]
    generations = [100, 200, 500, 1000]

    # Список обраних параметрів
    main_population_size = 500
    main_mutation_rate = 0.01
    main_elite_count = 100
    main_generations = 500

    default_main_population_size = 500
    default_main_mutation_rate = 0.01
    default_main_elite_count = 100
    default_main_generations = 500

    # Параметри графіків
    line_styles = ['-', '--', '-.', ':'] # Різні стилі ліній
    colors = ['b', 'g', 'r', 'c']      # Різні кольори ліній

    def mutation_rates_test():
        results_mutation_rates = {}
        # Прогоняємо алгоритм
        for rate in mutation_rates:
            sesion = GeneticAlgorithm(main_population_size, 48,
main_generations, rate, main_elite_count)
            best_fitness_results = sesion.run()
            results_mutation_rates[rate] = best_fitness_results
        # Побудова графіка
        for idx, (rate, fitness) in enumerate(results_mutation_rates.items()):
            plt.plot(fitness, linestyle=line_styles[idx % len(line_styles)],
color=colors[idx % len(colors)],
                    marker='o', label=f'Mutation Rate {rate}', alpha=0.7)
        plt.title("Best Fitness vs Generation (Different Mutation Rates)")
        plt.xlabel("Generation")
        plt.ylabel("Best Fitness")
        plt.legend()
        plt.grid(True)
        plt.show()

    def population_sizes_test():
        population_sizes_rates = {}
        # Прогоняємо алгоритм
        for size in population_sizes:
            sesion = GeneticAlgorithm(size, 48, main_generations,
main_mutation_rate, main_elite_count)
            best_fitness_results = sesion.run()
            population_sizes_rates[size] = best_fitness_results
        # Побудова графіка

```

```

        for idx, (size, fitness) in enumerate(population_sizes_rates.items()):
            plt.plot(fitness, linestyle=line_styles[idx % len(line_styles)],
                    color=colors[idx % len(colors)],
                    marker='o', label=f'Population size {size}', alpha=0.7)
        plt.title("Best Fitness vs Generation (Population size)")
        plt.xlabel("Generation")
        plt.ylabel("Best Fitness")
        plt.legend()
        plt.grid(True)
        plt.show()

    def elite_counts_test():
        elite_counts_rates = {}
        # Прогоняємо алгоритм
        for count in elite_counts:
            session = GeneticAlgorithm(main_population_size, 48,
main_generations, main_mutation_rate, count)
            best_fitness_results = session.run()
            elite_counts_rates[count] = best_fitness_results
        # Побудова графіка
        for idx, (count, fitness) in enumerate(elite_counts_rates.items()):
            plt.plot(fitness, linestyle=line_styles[idx % len(line_styles)],
                    color=colors[idx % len(colors)],
                    marker='o', label=f'Elite count {count}', alpha=0.7)
        plt.title("Best Fitness vs Generation (Elite count)")
        plt.xlabel("Generation")
        plt.ylabel("Best Fitness")
        plt.legend()
        plt.grid(True)
        plt.show()

    def generations_test():
        generations_rates = {}
        # Прогоняємо алгоритм
        for generation in generations:
            session = GeneticAlgorithm(main_population_size, 48, generation,
main_mutation_rate, main_elite_count)
            best_fitness_results = session.run()
            generations_rates[generation] = best_fitness_results
        # Побудова графіка
        for idx, (generation, fitness) in
enumerate(generations_rates.items()):
            plt.plot(fitness, linestyle=line_styles[idx % len(line_styles)],
                    color=colors[idx % len(colors)],
                    marker='o', label=f'Generation {generation}', alpha=0.7)
        plt.title("Best Fitness vs Generation (Generation)")
        plt.xlabel("Generation")
        plt.ylabel("Best Fitness")
        plt.legend()
        plt.grid(True)
        plt.show()

    def test():
        rates = {}
        # Прогоняємо алгоритм
        session = GeneticAlgorithm(main_population_size, 48, main_generations,
main_mutation_rate, main_elite_count)
        best_fitness_results = session.run()
        rates["test"] = best_fitness_results

        # Обчислення середнього значення для кожного покоління

```

```

        average_fitness_results = [np.mean(best_fitness_results[:i+1]) for i
in range(len(best_fitness_results))]

    # Побудова графіка
    params_info = (f'Population Size: {main_population_size}, '
                  f'Generations: {main_generations}, '
                  f'Mutation Rate: {main_mutation_rate*100}%, '
                  f'Elite                               Count:
{main_elite_count/main_population_size*100}%')

    for idx, (generation, fitness) in enumerate(rates.items()):
        plt.plot(fitness, linestyle=line_styles[idx % len(line_styles)],
color=colors[idx % len(colors)],
                label=f'{params_info}', alpha=0.7)
    # Додаємо лінію для середнього значення
    plt.plot(average_fitness_results, linestyle='--', color='black',
label='Average Fitness', alpha=0.8)
    plt.title("Best Fitness vs Generation")
    plt.xlabel("Generation")
    plt.ylabel("Best Fitness")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Запитуємо параметри у користувача з можливістю використати значення за
замовчуванням
    try:
        population_size_input = input("Population size (default: 500):
").strip()
        # Якщо ввід порожній, використовується значення за замовчуванням
        if not population_size_input:
            main_population_size = default_main_population_size
            print(f"Set default: {main_population_size}")
        else:
            # Спроба перетворити на ціле число
            main_population_size = int(population_size_input)
    except ValueError:
        # У разі некоректного вводу використовується значення за замовчуванням
        main_population_size = default_main_population_size
        print(f"Error, set default: {main_population_size}")

    try:
        mutation_rate_input = input("Mutation level (default: 0.01):
").strip()
        # Якщо ввід порожній, використовується значення за замовчуванням
        if not mutation_rate_input:
            main_mutation_rate = default_main_mutation_rate
            print(f"Set default: {main_mutation_rate}")
        else:
            # Спроба перетворити на ціле число
            main_mutation_rate = int(mutation_rate_input)
    except ValueError:
        # У разі некоректного вводу використовується значення за замовчуванням
        main_mutation_rate = default_main_mutation_rate
        print(f"Error, set default: {main_mutation_rate}")

    try:
        elite_count_input = input("Number of elite individuals (default: 100):
").strip()
        # Якщо ввід порожній, використовується значення за замовчуванням
        if not elite_count_input:
            main_elite_count = default_main_elite_count

```

```

        print(f"Set default: {main_elite_count}")
    else:
        # Спроба перетворити на ціле число
        main_elite_count = int(elite_count_input)
except ValueError:
    # У разі некоректного вводу використовується значення за замовчуванням
    print(f"Error, set default: {main_elite_count}")
    main_elite_count = default_main_elite_count

try:
    generations_input = input("Number of generations (default: 500):
").strip()
    # Якщо ввід порожній, використовується значення за замовчуванням
    if not generations_input:
        main_generations = default_main_generations
        print(f"Set default: {main_generations}")
    else:
        # Спроба перетворити на ціле число
        main_generations = int(generations_input)
except ValueError:
    # У разі некоректного вводу використовується значення за замовчуванням
    main_generations = default_main_generations
    print(f"Error, set default: {main_generations}")

test()

```

Б.2 Файл genetic. py

```

from AnalyticalSubsystem.environment import GameEnvironment
from AnalyticalSubsystem.fitness import fitness_function
from AnalyticalSubsystem.individual import Individual
from SnakeGameSubsystem.simulation import SnakeGameVisualSimulation
from multiprocessing import Pool
import numpy as np
import random
import time

# Симуляція гри для однієї особи
def evaluate_individual(individual, game_environment):
    return fitness_function(individual, game_environment)

class GeneticAlgorithm:
    def __init__(self, population_size, genome_length, generations,
mutation_rate=0.01, elitism_count=5):
        # Ініціалізація генетичного алгоритму.
        # :param population_size: Розмір популяції.
        # :param genome_length: Довжина геному кожної особи.

        # :param generations: Кількість поколінь.
        # :param mutation_rate: Імовірність мутації гена.

        self.population_size = population_size
        self.genome_length = genome_length

        self.generations = generations
        self.mutation_rate = mutation_rate
        self.population = [Individual(genome_length) for _ in
range(population_size)]

```

```

self.elitism_count = elitism_count
self.pool = Pool()

def __del__(self):
    self.pool.close()
    self.pool.join()

def evaluate_population(self):
    # Оцінює кожну особину в популяції за допомогою функції
    # пристосованості.
    # :return: Список оцінок для кожної особини. return
    [fitness_function(individual, self.game_environment) for individual in
    self.population]
    return self.pool.starmap(
        evaluate_individual,
        [(ind, GameEnvironment()) for ind in self.population]
    )

def select_parents(self, fitness_scores):
    # Вибирає батьків для схрещування на основі оцінок пристосованості.
    # :param fitness_scores: Оцінки пристосованості популяції.
    # :return: Вибрані батьки.

    total_fitness = sum(fitness_scores)
    probabilities = [score / total_fitness for score in fitness_scores]
    parents = random.choices(self.population, weights=probabilities, k=2)
    return parents

def crossover(self, parent1, parent2):
    # Виконує схрещування між двома батьками.
    # :param parent1: Перша особина-батько.
    # :param parent2: Друга особина-батько.
    # :return: Нова особина-дитина.

    child_genome = []
    child_genome = [
        parent1.genome[i] * 0.6 + parent2.genome[i] * 0.4 # Ваги можуть
змінюватися
        for i in range(self.genome_length)
    ]
    child = Individual(self.genome_length)
    child.genome = np.clip(child_genome, -1, 1) # Обмеження значень
    return child

def mutate(self, individual):
    # Виконує мутацію геному особини з певною імовірністю.
    # :param individual: Особина, яку потрібно мутувати.

    for i in range(self.genome_length):
        if random.uniform(0, 1) < self.mutation_rate:
            sigma = max(0.1 * (1 - self.current_generation /
self.generations), 0.01)
            individual.genome[i] += np.random.normal(0, sigma) # зміна
            individual.genome[i] = np.clip(individual.genome[i], -1, 1) #
Обмежуємо значення

def run(self):
    # Основний цикл генетичного алгоритму.
    best_fitness_per_generation = []

    for generation in range(self.generations):
        self.current_generation = generation

```

```

        # Оцінка поточної популяції
        fitness_scores = self.evaluate_population()
        print(f"Generation {generation}: Best Fitness = {max(fitness_scores)}")

        best_fitness = max(fitness_scores)
        best_fitness_per_generation.append(best_fitness)

        # Створення нового покоління
        new_population = []

        # Зберігаємо кращі elitism_count особини
        sorted_population = sorted(zip(self.population, fitness_scores),
key=lambda x: x[1], reverse=True)
        elite_individuals = [ind for ind, score in
sorted_population[:self.elitism_count]]
        new_population.extend(elite_individuals)

        while len(new_population) < self.population_size:
            parent1, parent2 = self.select_parents(fitness_scores)
            child = self.crossover(parent1, parent2)
            self.mutate(child)
            new_population.append(child)

        self.population = new_population
        best_index = fitness_scores.index(max(fitness_scores))
        best_game = SnakeGameVisualSimulation(self.population[best_index])
        best_game.play()
        while best_game.game_over == False:
            time.sleep(100)

        # Повертаємо найкращу особину після завершення покоління
        best_index = fitness_scores.index(max(fitness_scores))
        best_game = SnakeGameVisualSimulation(self.population[best_index])
        best_game.play()
        while best_game.game_over == False:
            time.sleep(100)

    return best_fitness_per_generation

```

Б.3 Файл environment.py

```

from SnakeGameSubsystem.simulation import SnakeGameSimulation, UP, DOWN, LEFT,
RIGHT

class GameEnvironment:
    def __init__(self):
        # Ініціалізація середовища гри.
        # :param game: Об'єкт гри (наприклад, SnakeGame).
        self.game = SnakeGameSimulation()

    def reset(self):
        # Скидає гру до початкового стану.
        self.game.reset_game()

    def get_snake_position(self):

```

```

# Повертає інформацію про місцезнаходження голови змійки.
# :return: Кординати голови змійки.
return self.game.snake[0]

def get_snake_body(self):
    # Повертає інформацію про місцезнаходження тіла змійки.
    # :return: Кординати тіла змійки (з головою).
    return self.game.snake

def get_food_position(self):
    # Повертає інформацію про місцезнаходження їжі.
    # :return: Кординати їжі.
    return self.game.food

def step(self, action):
    # Виконує дію у грі та повертає винагороду і стан завершення гри.
    # :param action: Дія, яку необхідно виконати (напрямок руху змійки).
    # :return: Винагорода та статус завершення гри (reward, done).

    directions = [UP, DOWN, LEFT, RIGHT]
    self.game.direction = directions[action]

    food_position = self.game.food
    snake_position = self.game.snake[0]
    distance_before = abs(snake_position[0] - food_position[0]) +
abs(snake_position[1] - food_position[1])

    # Оновлюємо гру
    previous_length = len(self.game.snake)
    self.game.update()

    # Обчислюємо відстань до їжі після виконання дії
    distance_after = abs(snake_position[0] - food_position[0]) +
abs(snake_position[1] - food_position[1])

    base_reward = 1000

    # Обчислення винагороди
    if self.game.game_over:
        reward = base_reward * 0.1 # Штраф за завершення гри
        done = True
    elif len(self.game.snake) > previous_length:
        reward = base_reward * 1.0 # Нагорода за їжу
        done = False
    else:
        reward = -base_reward * 0.001 # Невеликий штраф за кожен хід
        done = False

    # Винагорода/штраф за зміну відстані до їжі
    distance_reward = base_reward * 0.001 if distance_after <=
distance_before else -base_reward * 0.001
    reward = reward + distance_reward

    return reward, done

```

Б.4 Файл simulation. py

```

import pygame
import random

```

```

# Розміри гри
CELL_SIZE = 20
GRID_WIDTH = CELL_SIZE
GRID_HEIGHT = CELL_SIZE

# Напрямки руху
UP = (0, -1)
DOWN = (0, 1)
LEFT = (-1, 0)
RIGHT = (1, 0)

class SnakeGameSimulation:
    def __init__(self):
        self.reset_game()

    def reset_game(self):
        # Ініціалізація початкових умов
        self.snake = [(5, 5)] # Початкова позиція змійки
        self.direction = RIGHT
        self.food = self.place_food()
        self.game_over = False

    def place_food(self):
        # Розміщення їжі в випадковій позиції
        while True:
            food = (random.randint(0, GRID_WIDTH - 1), random.randint(0,
GRID_HEIGHT - 1))
            if food not in self.snake:
                return food

    def update(self):
        if self.game_over:
            return

        # Рух змійки
        head_x, head_y = self.snake[0]
        delta_x, delta_y = self.direction
        new_head = (head_x + delta_x, head_y + delta_y)

        # Перевірка на зіткнення зі стінами або з собою
        if not (0 <= new_head[0] < GRID_WIDTH and 0 <= new_head[1] <
GRID_HEIGHT):
            self.game_over = True
            return
        if new_head in self.snake:
            self.game_over = True
            return

        # Додавання нової голови
        self.snake.insert(0, new_head)

        # Перевірка, чи з'їли їжу
        if new_head == self.food:
            self.food = self.place_food()
        else:
            self.snake.pop()

    def update_direction(self, direction):
        # Оновлює напрямок руху змійки
        if direction == "UP" and self.direction != DOWN:
            self.direction = UP

```

```

elif direction == "DOWN" and self.direction != UP:
    self.direction = DOWN
elif direction == "LEFT" and self.direction != RIGHT:
    self.direction = LEFT
elif direction == "RIGHT" and self.direction != LEFT:
    self.direction = RIGHT

# Параметри вікна
SCREEN_WIDTH = GRID_WIDTH * GRID_WIDTH
SCREEN_HEIGHT = GRID_HEIGHT * GRID_HEIGHT
BLOCK_SIZE = CELL_SIZE

# Кольори
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLACK = (0, 0, 0)

class SnakeGameVisualSimulation:
    def __init__(self, genome):
        self.snake = [(200, 200), (180, 200)] # Початкова позиція змійки
        self.direction = (BLOCK_SIZE, 0) # Напрямок руху (права)
        self.apple = (300, 200) # Початкова позиція яблука
        self.genome = genome # Геном топ-гравця
        self.game_over = False

    def move_snake(self):
        # Змійка рухається відповідно до геному (наприклад, кожен крок - це
        одна дія)
        head = self.snake[0]
        new_head = (head[0] + self.direction[0], head[1] + self.direction[1])
        self.snake = [new_head] + self.snake[1:]

        # Перевірка зіткнень з яблуком
        if self.snake[0] == self.apple:
            self.snake.append(self.last) # Додаємо нову частину тіла
            food_is_not_place = True
            while food_is_not_place:
                self.apple = (random.randint(0, (SCREEN_WIDTH // BLOCK_SIZE) -
1) * BLOCK_SIZE,
                                random.randint(0, (SCREEN_HEIGHT // BLOCK_SIZE)
- 1) * BLOCK_SIZE)
                if self.apple not in self.snake:
                    food_is_not_place = False
            self.last = self.snake[-1]

        # Перевірка зіткнення з кордонами екрану або собою
        if (self.snake[0][0] < 0 or self.snake[0][0] >= SCREEN_WIDTH or
            self.snake[0][1] < 0 or self.snake[0][1] >= SCREEN_HEIGHT or
            self.snake[0] in self.snake[1:]):
            self.game_over = True

    def update_direction(self, direction):
        # Оновлює напрямок руху змійки
        if direction == 0 and self.direction != (0, BLOCK_SIZE): # UP
            self.direction = (0, -BLOCK_SIZE)
        elif direction == 1 and self.direction != (0, -BLOCK_SIZE): # DOWN
            self.direction = (0, BLOCK_SIZE)
        elif direction == 2 and self.direction != (BLOCK_SIZE, 0): # LEFT
            self.direction = (-BLOCK_SIZE, 0)
        elif direction == 3 and self.direction != (-BLOCK_SIZE, 0): # RIGHT
            self.direction = (BLOCK_SIZE, 0)

```

```

def render(self, screen):
    # Очищаємо екран
    screen.fill(BLACK)

    # Малюємо змійку
    for segment in self.snake:
        pygame.draw.rect(screen, GREEN, pygame.Rect(segment[0],
segment[1], BLOCK_SIZE, BLOCK_SIZE))

    # Малюємо яблуко
    pygame.draw.rect(screen, RED, pygame.Rect(self.apple[0],
self.apple[1], BLOCK_SIZE, BLOCK_SIZE))

    # Оновлюємо екран
    pygame.display.flip()

def play(self):
    screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
    clock = pygame.time.Clock()

    generation = 0
    steps = 0
    while not self.game_over:
        # Оновлення геному (дії на кожному кроці)
        action = self.genome.genome_to_action(self.snake[0], self.apple,
self.snake)
        self.update_direction(action)
        self.move_snake()
        self.render(screen)

        # Пауза між кадрами
        clock.tick(10) # FPS (кількість кадрів на секунду)
        steps = steps + 1
        generation += 1
        if steps == 500:
            self.game_over = True

```

Б.5 Файл individual.py

```

from SnakeGameSubsystem.simulation import GRID_WIDTH, GRID_HEIGHT
import numpy as np

class Individual:
    def __init__(self, genome_length):
        # Ініціалізація особини з випадковим геномом.
        # :param genome_length: Довжина геному (кількість параметрів).

        self.genome = np.random.uniform(-1, 1, genome_length) # Геном:
випадкові числа між -1 і 1
        self.fitness = 0

    def genome_to_action(self, snake_position, food_position, snake_body):
        # Перетворює геном і стан гри на дію (напрямок руху змійки).
        # :return: Дія у вигляді індексу (0 - вгору, 1 - вниз, 2 - ліворуч, 3
- праворуч).

        self.snake_position = snake_position
        self.snake_body = snake_body

```

```

# Витягуємо просторові ознаки
distance_to_food_x = food_position[0] - snake_position[0]
distance_to_food_y = food_position[1] - snake_position[1]

distance_to_wall_up = snake_position[1]
distance_to_wall_down = GRID_HEIGHT - snake_position[1] - 1
distance_to_wall_left = snake_position[0]
distance_to_wall_right = GRID_WIDTH - snake_position[0] - 1

# Чи їжа зверху/знизу та зліва/справа
food_vertical = np.sign(distance_to_food_y) # -1, 0, 1
food_horizontal = np.sign(distance_to_food_x) # -1, 0, 1
distances_to_obstacles = self.calculate_distances_to_obstacles()

# Вхідні характеристики
input_features = np.array([
    distance_to_food_x / GRID_WIDTH, distance_to_food_y / GRID_HEIGHT,
    distance_to_wall_up / GRID_HEIGHT, distance_to_wall_down /
GRID_HEIGHT,
    distance_to_wall_left / GRID_WIDTH, distance_to_wall_right /
GRID_WIDTH,
    food_vertical, food_horizontal, *(np.array(distances_to_obstacles)
/ max(GRID_WIDTH, GRID_HEIGHT))
])

# Використання нелінійної активації
transformed_features = np.tanh(input_features)

# Обчислення оцінки дій
action_scores = [
    np.dot(transformed_features, self.genome[i::4]) for i in range(4)
]

# Повертаємо найкращу дію
return action_scores.index(max(action_scores))

def calculate_distances_to_obstacles(self):
    # Обчислює відстань до перешкод (стіна або тіло змійки) в кожному
    напрямку.
    distances = []

    # Визначення напрямків
    directions = {
        "UP": (0, -1),
        "DOWN": (0, 1),
        "LEFT": (-1, 0),
        "RIGHT": (1, 0),
    }
    for direction, (dx, dy) in directions.items():
        distance = 0
        current_x, current_y = self.snake_position
        while True:
            current_x += dx
            current_y += dy
            distance += 1
            # Перевірка виходу за межі поля
            if current_x < 0 or current_x >= GRID_WIDTH or current_y < 0
or current_y >= GRID_HEIGHT:
                break
            # Перевірка на зіткнення з тілом змійки
            if (current_x, current_y) in self.snake_body:
                break

```

```

        distances.append(distance)

    return distances

```

Б.6 Файл `fitness.py`

```

def fitness_function(individual, game_environment):
    # Обчислення оцінки придатності для конкретної особини.
    # :param individual: Особина, геном якої потрібно оцінити.
    # :param game_environment: Об'єкт середовища гри для симуляції.
    # :return: Числове значення оцінки придатності.

    # 1. Скидаємо гру до початкового стану
    game_environment.reset()

    # 2. Виконуємо симуляцію гри для цієї особини
    score = 0
    max_steps = 1000 # Ліміт кроків для уникнення зависання
    for _ in range(max_steps):
        # Використовуємо геном особини для прийняття рішення
        action =
individual.genome_to_action(game_environment.get_snake_position(),
game_environment.get_food_position(), game_environment.get_snake_body())
        reward, done = game_environment.step(action)

        # Додаємо винагороду до загального балу
        score += reward

        # Якщо гра закінчилася (зіткнення або перемога), виходимо з симуляції
        if done:
            break

    return score

```

ДОДАТОК В
Слайди презентації

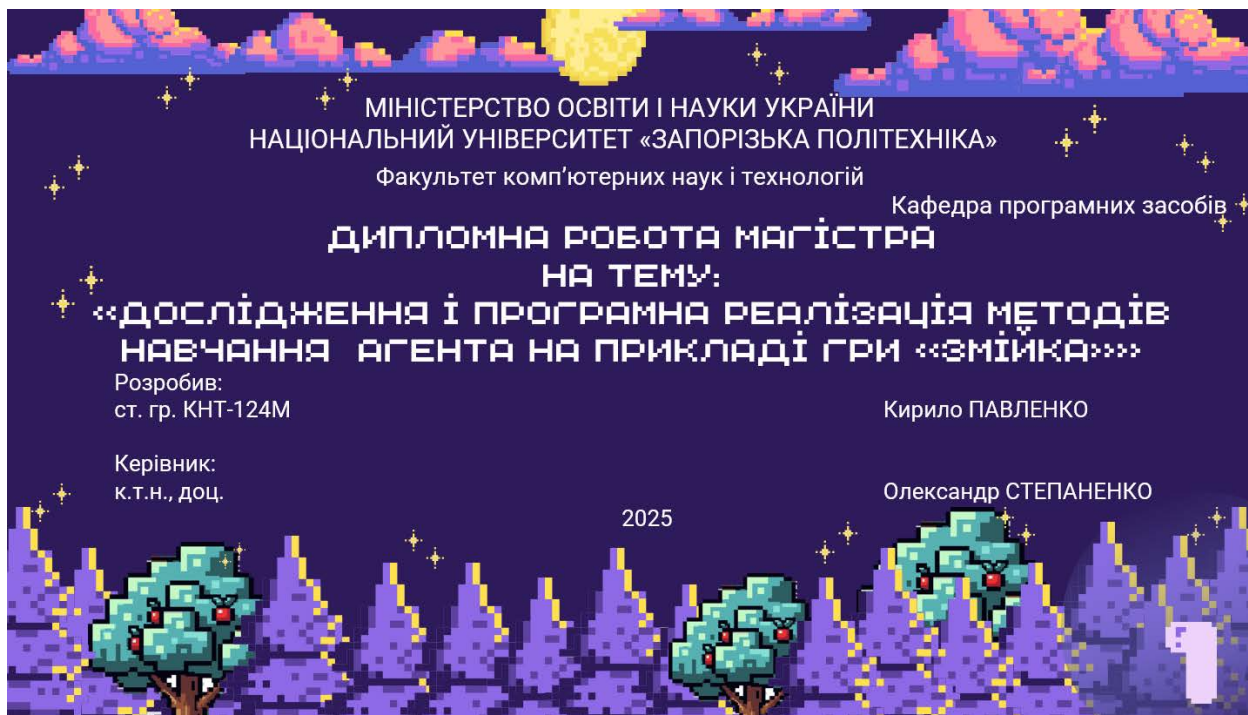


Рисунок В.1 – Перший слайд

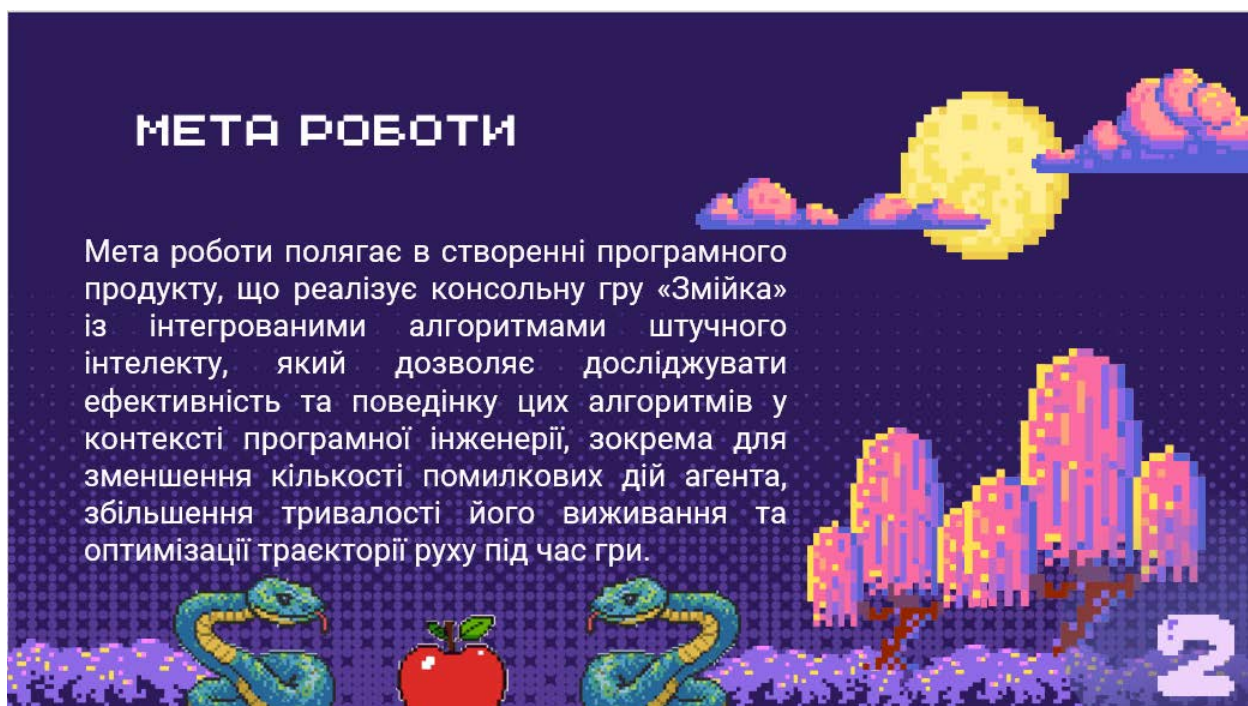


Рисунок В.2 – Другий слайд

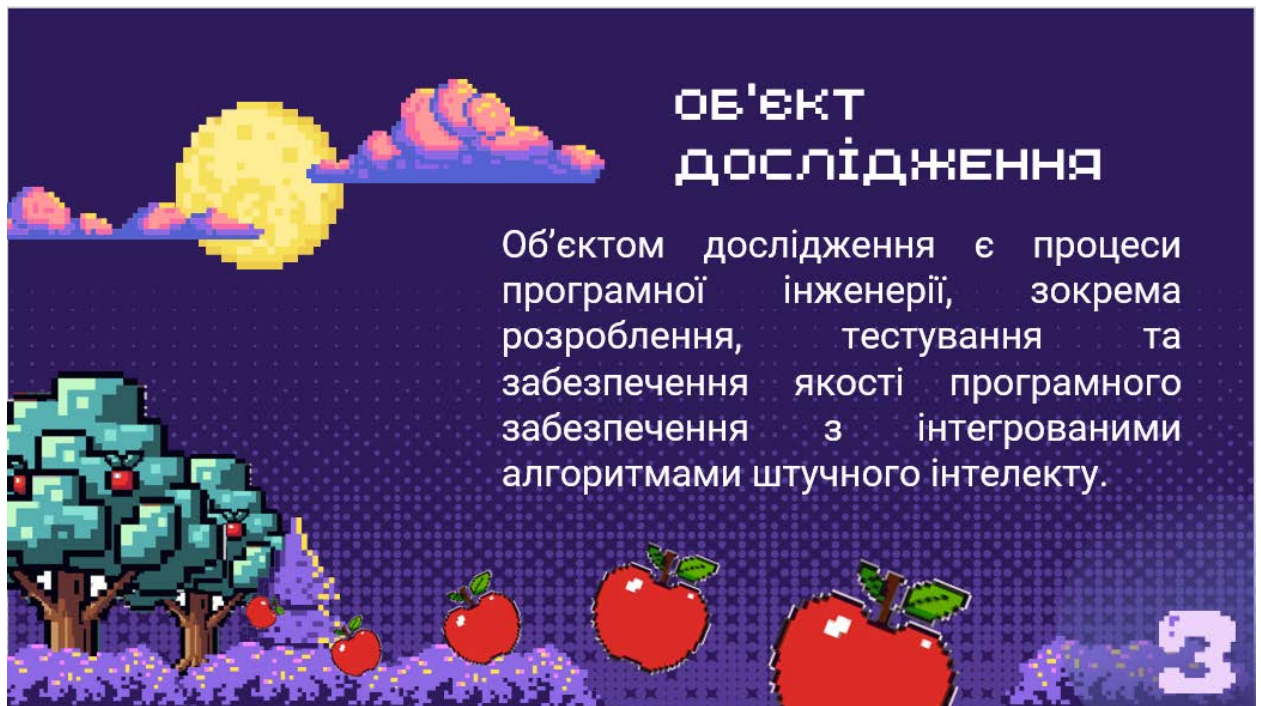


Рисунок В.3 –Третій слайд

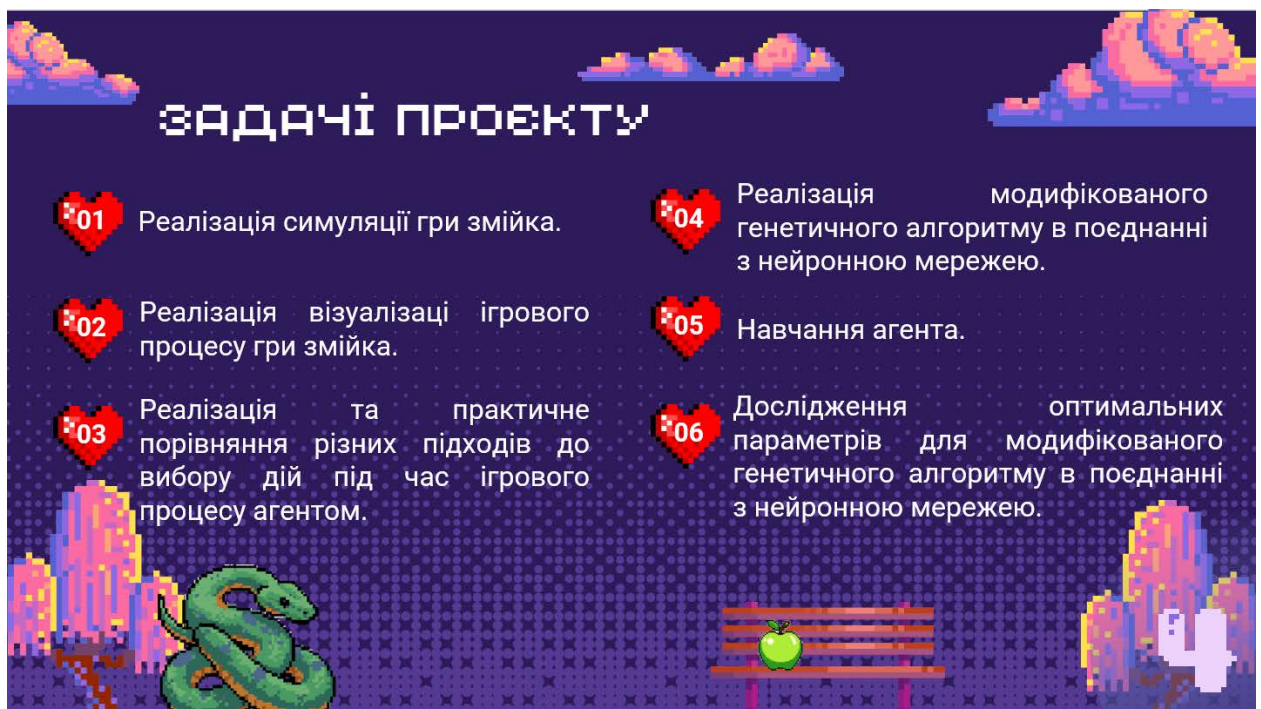


Рисунок В.4 –Четвертий слайд

ПОРІВНЯННЯ ІСНУЮЧИХ АНАЛОГІВ

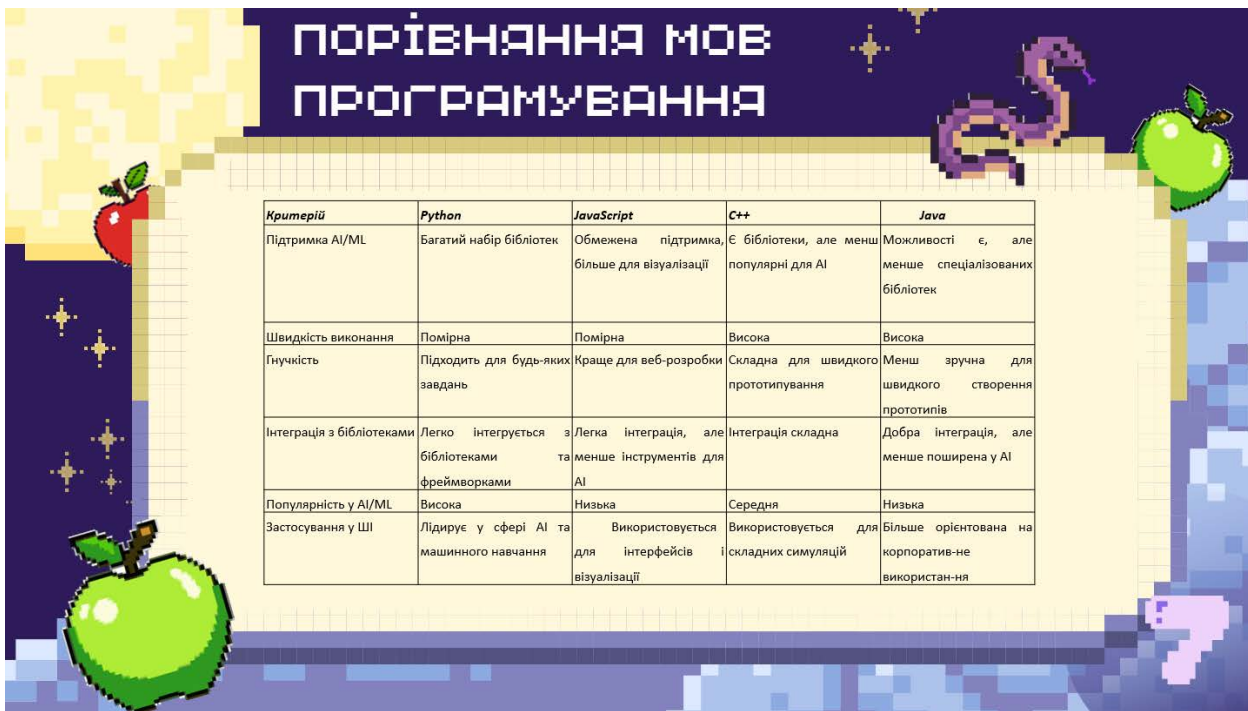
Можливості	SlytherSnake	Gym-Snake	SnakeQ	Snake-Neuroevolution
Методологія навчання	Гібридний метод	Мультиагентна система; класична гра з варіаціями	Deep Q-Learning	Neuroevolution (еволюція нейромереж)
Можливість симуляції середовища	Так (повна візуалізація найкращих агентів)	Так, з підтримкою мультиагентності	Лише для одного агента	Так
Візуалізація результатів	Симуляція найкращих агентів та створення графіків для аналізу ефективності навчання	Мінімальна візуалізація	Обмежена візуалізація	Мінімальна візуалізація
Можливість роботи з кількома агентами	Ні	Так	Ні	Ні
Гнучкість налаштувань середовища	Висока	Обмежена, фіксована структура гри	Фіксована структура гри	Фіксована структура гри
Простота реалізації	Середня, потребує налаштування ГА та нейронних мереж	Висока, готове середовище	Середня, вимагає налаштування DQL	Висока, основна реалізація на основі ГА

Рисунок В.5 –П’ятий слайд

ПОРІВНЯННЯ ПІДХОДІВ

Характеристики	ГА та нейронні мережі	DRL	Метод Монте-Карло	Евристичні алгоритми
Середній бал	4979.87	7650.54	2961.72	25433.31
Середнє відхилення	3886.56	4889.36	3096.10	9853.20
Середня довжина змійки	5.87	8.54	3.87	26.31
Найкращий результат	25122.00	27128.00	21122.00	59160.00

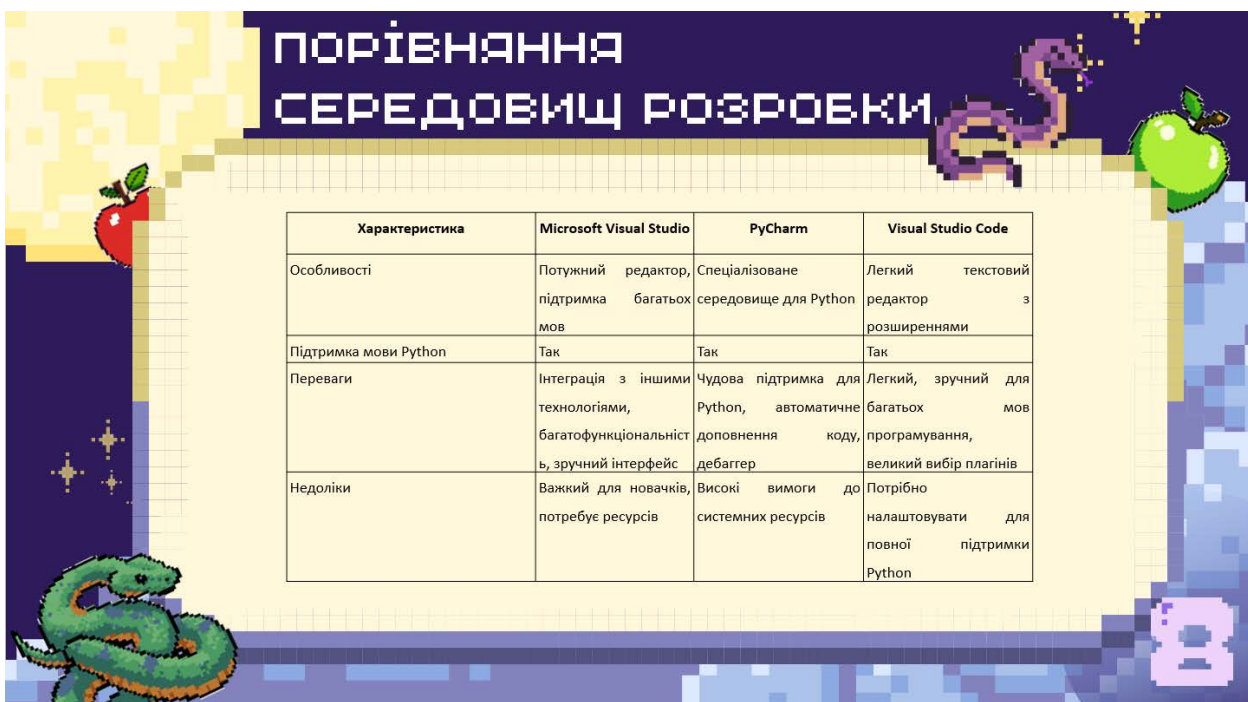
Рисунок В.6 – Шостий слайд



ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ

Критерій	Python	JavaScript	C++	Java
Підтримка AI/ML	Багатий набір бібліотек	Обмежена підтримка, більше для візуалізації	Є бібліотеки, але менш популярні для AI	Можливості є, але менше спеціалізованих бібліотек
Швидкість виконання	Помірна	Помірна	Висока	Висока
Гнучкість	Підходить для будь-яких завдань	Краще для веб-розробки	Складна для швидкого прототипування	Менш зручна для швидкого створення прототипів
Інтеграція з бібліотеками	Легко інтегрується з бібліотеками та фреймворками	Легка інтеграція, але менше інструментів для AI	Інтеграція складна	Добра інтеграція, але менше поширена у AI
Популярність у AI/ML	Висока	Низька	Середня	Низька
Застосування у ШІ	Лідирує у сфері AI та машинного навчання	Використовується для інтерфейсів візуалізації	Використовується для складних симуляцій	Більше орієнтована на корпоративне використання

Рисунок В.7 –Сьомий слайд



ПОРІВНЯННЯ СЕРЕДОВИЩ РОЗРОБКИ

Характеристика	Microsoft Visual Studio	PyCharm	Visual Studio Code
Особливості	Потужний редактор, підтримка багатьох мов	Спеціалізоване середовище для Python	Легкий текстовий редактор з розширеннями
Підтримка мови Python	Так	Так	Так
Переваги	Інтеграція з іншими технологіями, багатофункціональність, зручний інтерфейс	Чудова підтримка для Python, автоматичне доповнення коду, дебаггер	Легкий, зручний для багатьох мов програмування, великий вибір плагінів
Недоліки	Важкий для новачків, потребує ресурсів	Високі вимоги до системних ресурсів	Потрібно налаштовувати для повної підтримки Python

Рисунок В.8 –Восьмий слайд

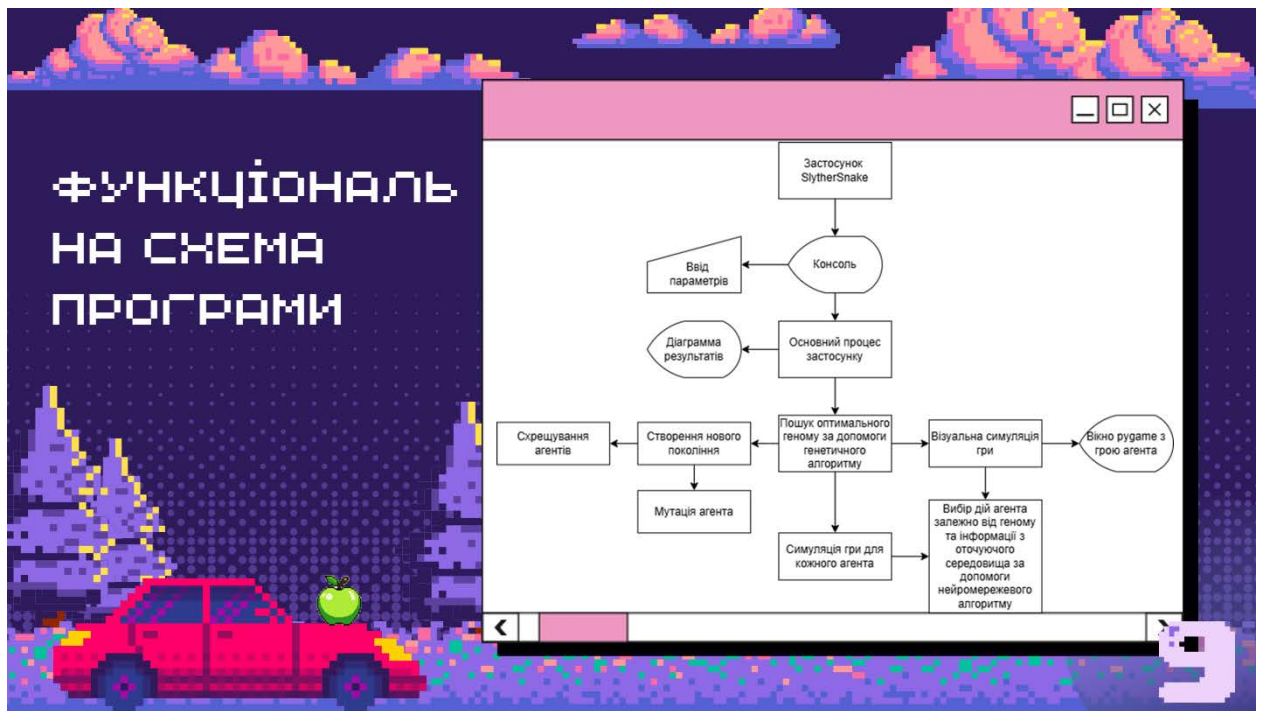


Рисунок В.9 – Дев'ятий слайд

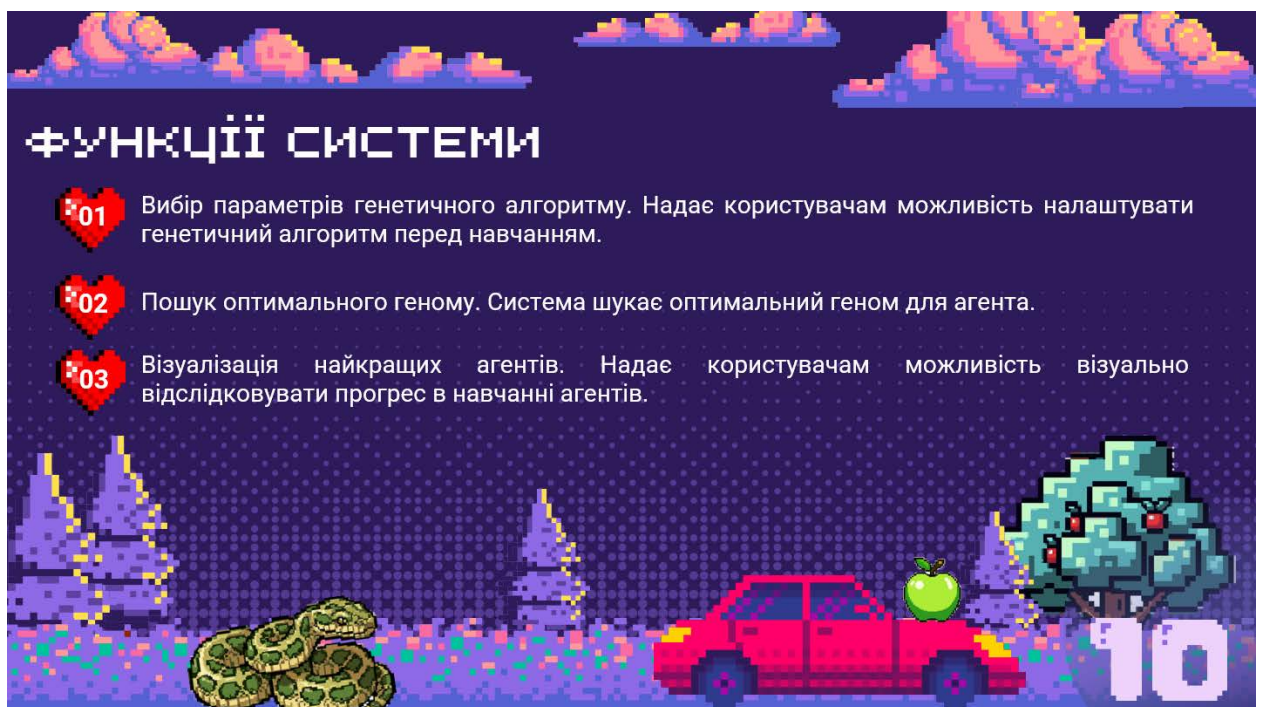


Рисунок В.10 – Десятий слайд

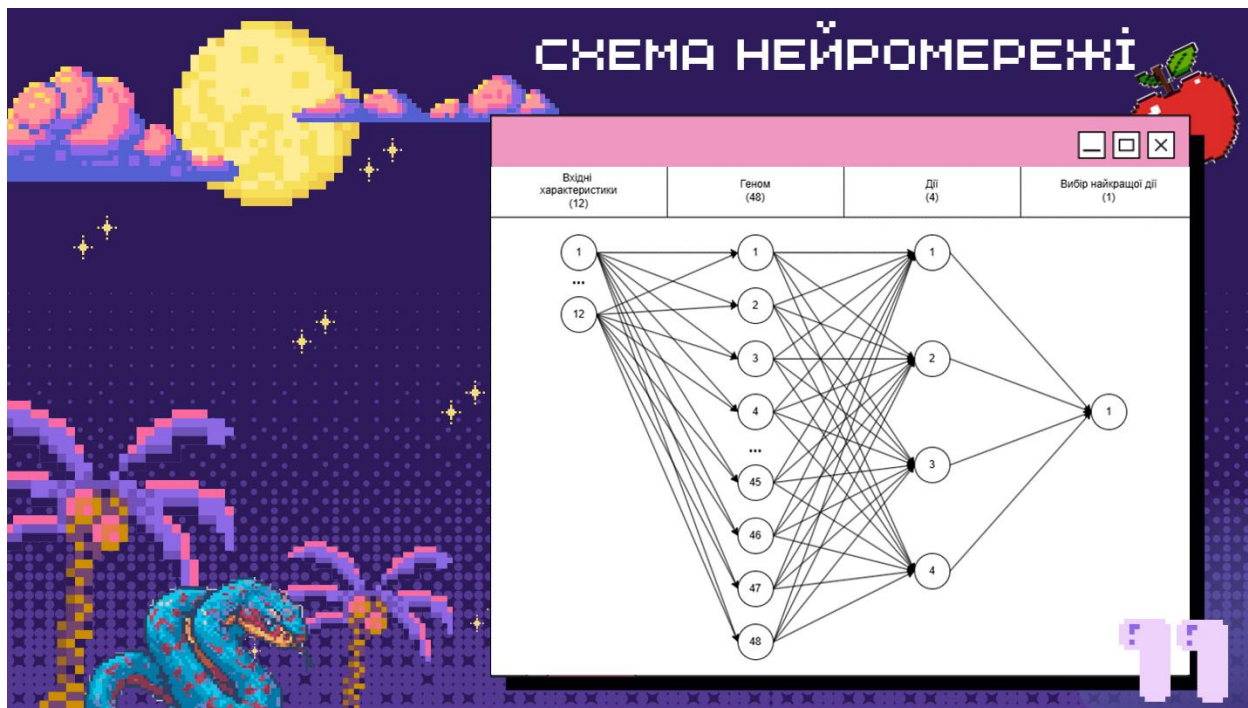


Рисунок В.11 – Одинадцятий слайд

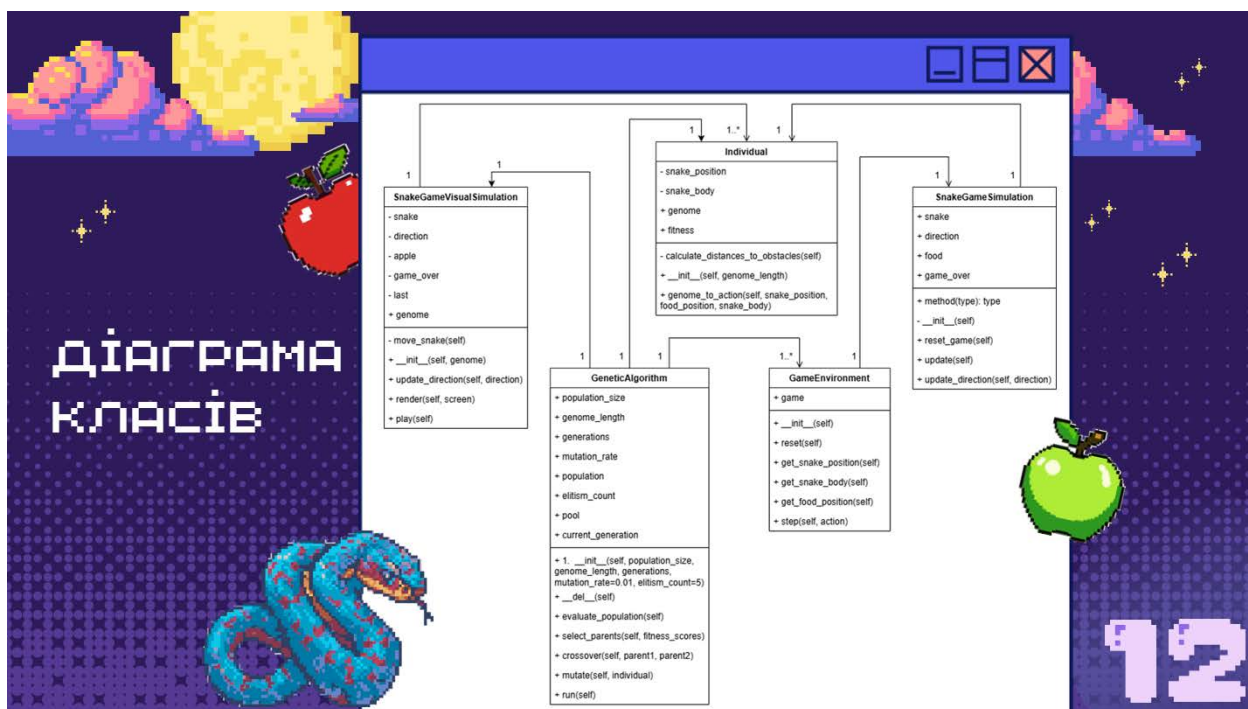


Рисунок В.12 – Дванадцятий слайд

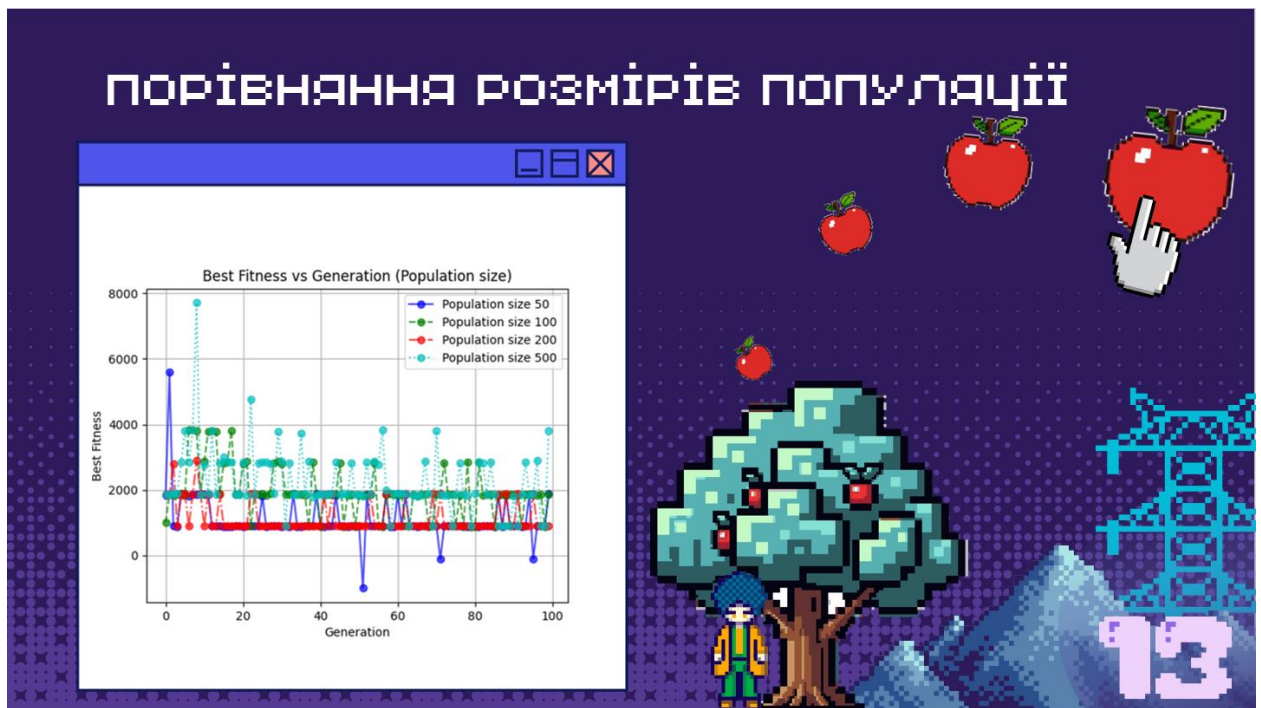


Рисунок В.13 – Тринадцятий слайд

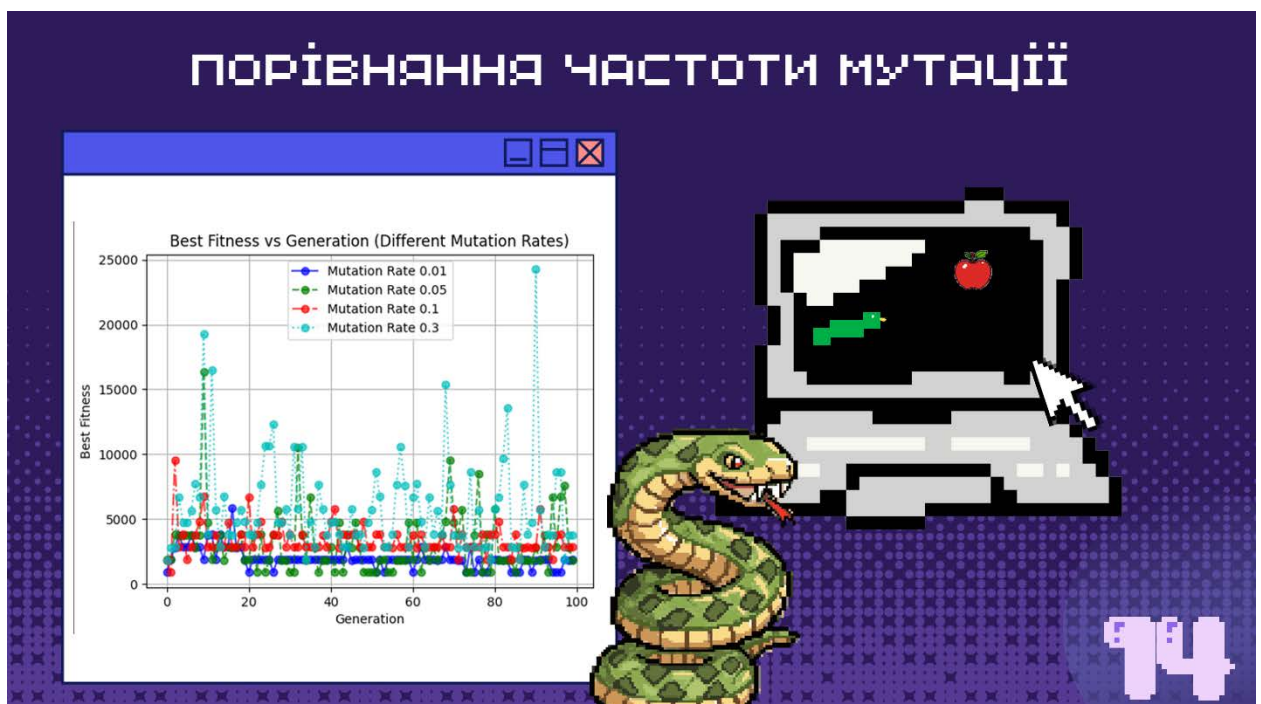


Рисунок В.14 – Чотирнадцятий слайд

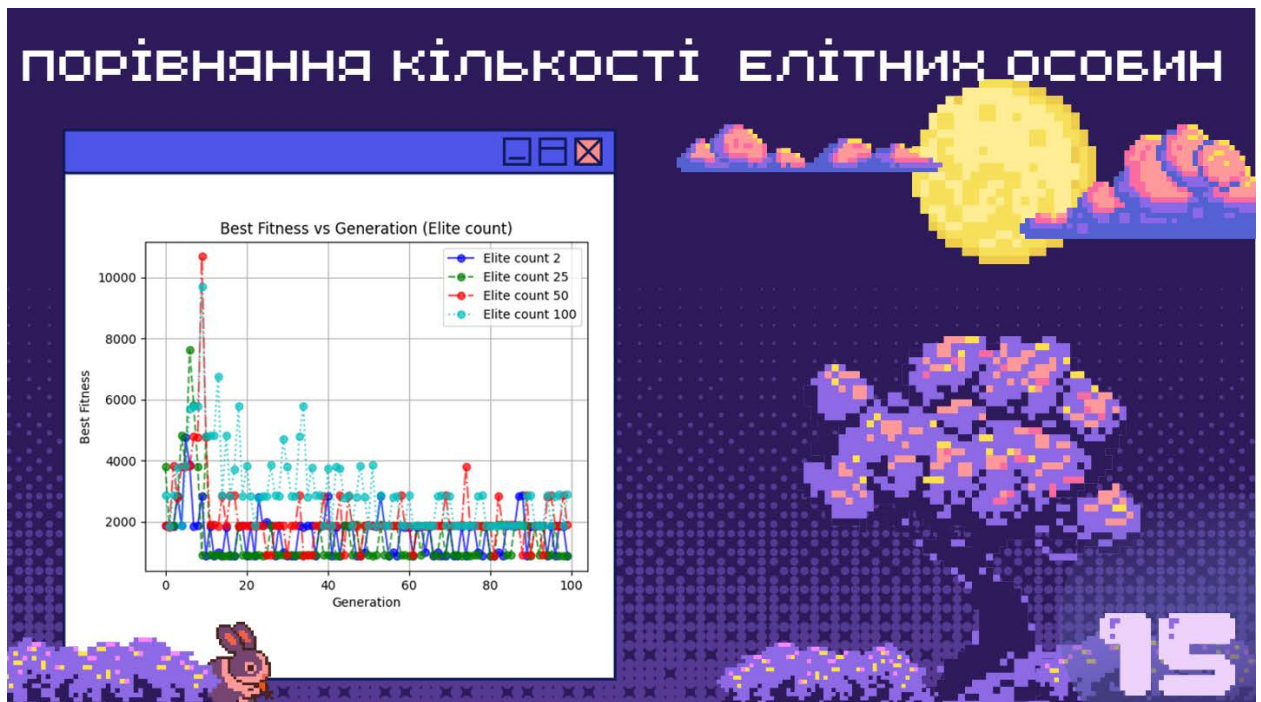


Рисунок В.15 – П'ятнадцятий слайд

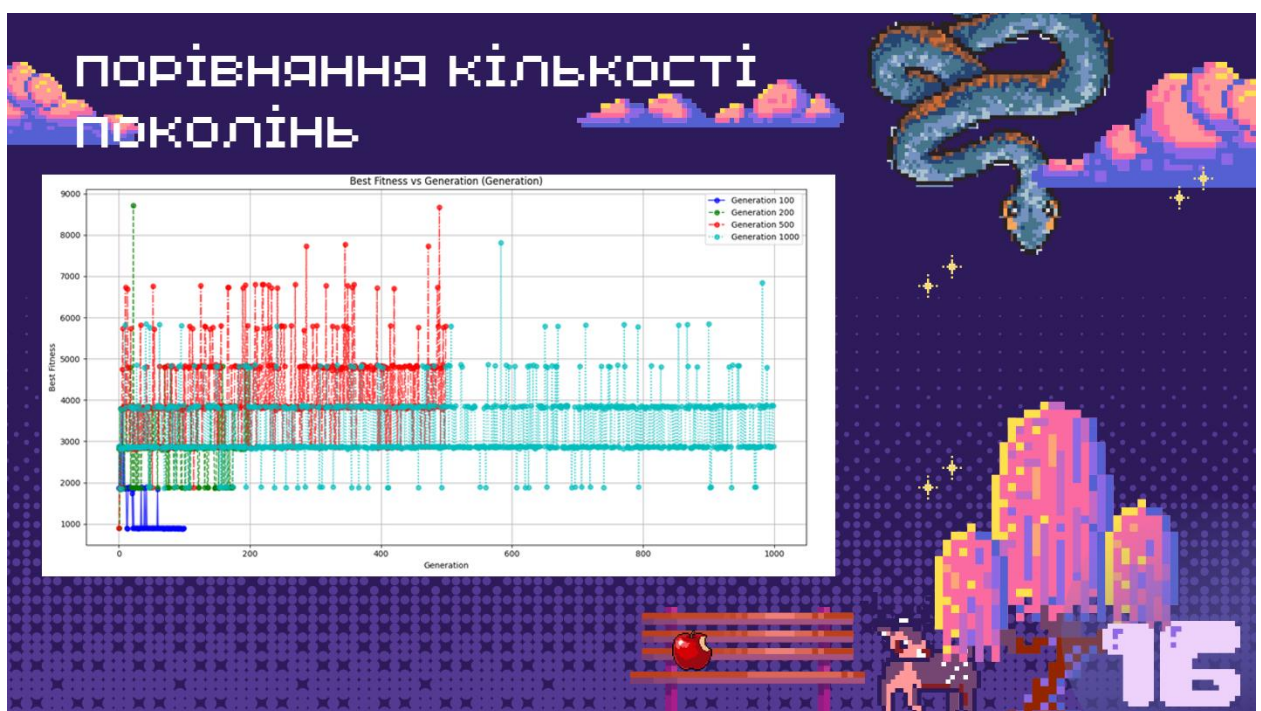


Рисунок В.16 – Шістнадцятий слайд

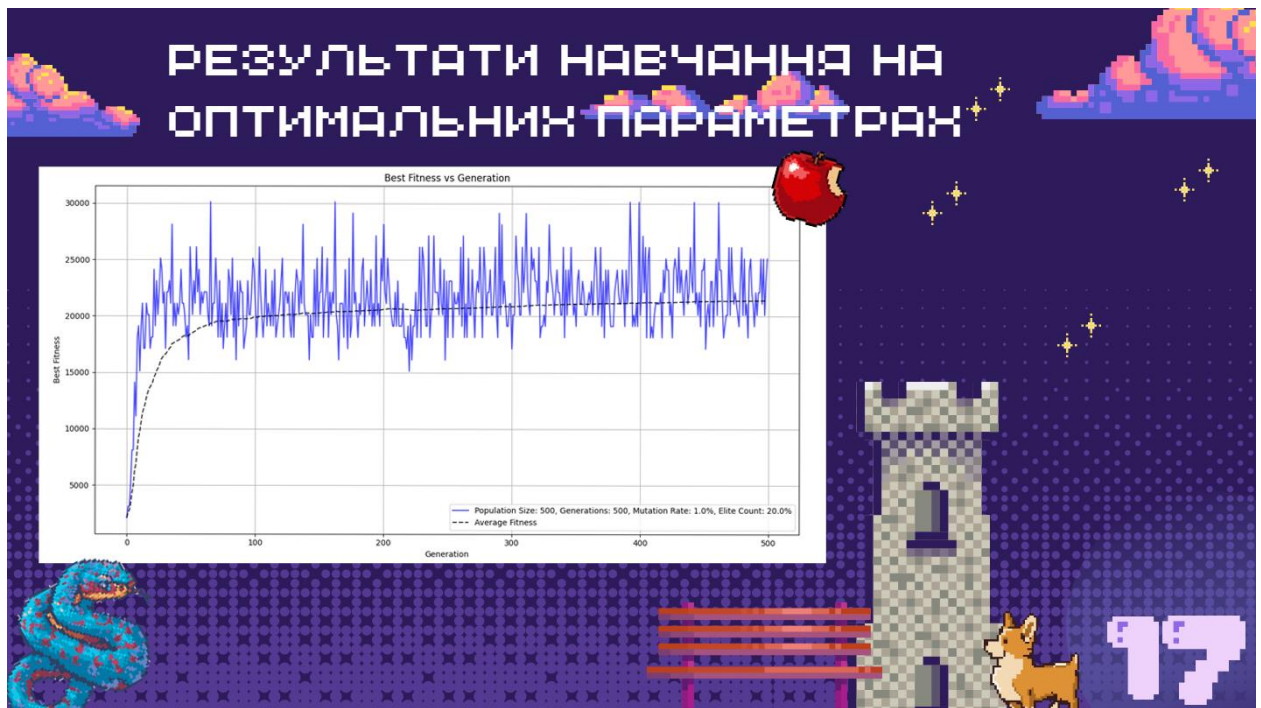


Рисунок В.17 –Сімнадцятий слайд

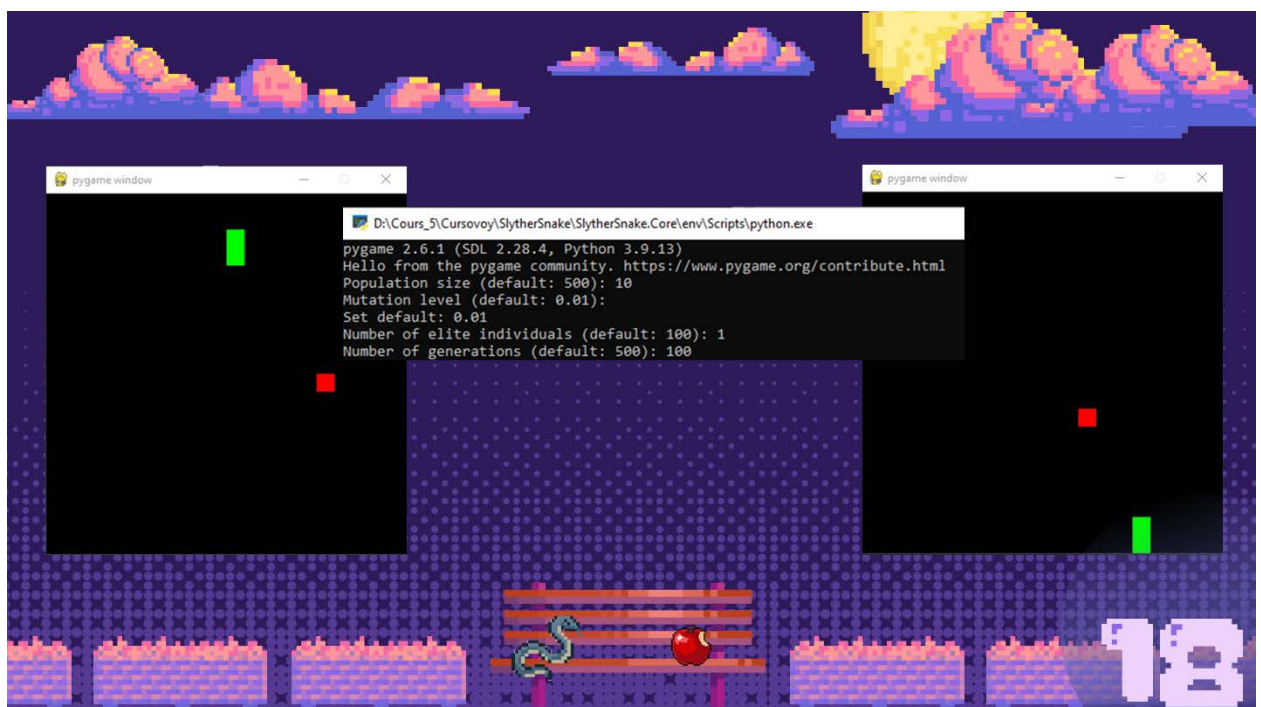


Рисунок В.18 –Вісімнадцятий слайд

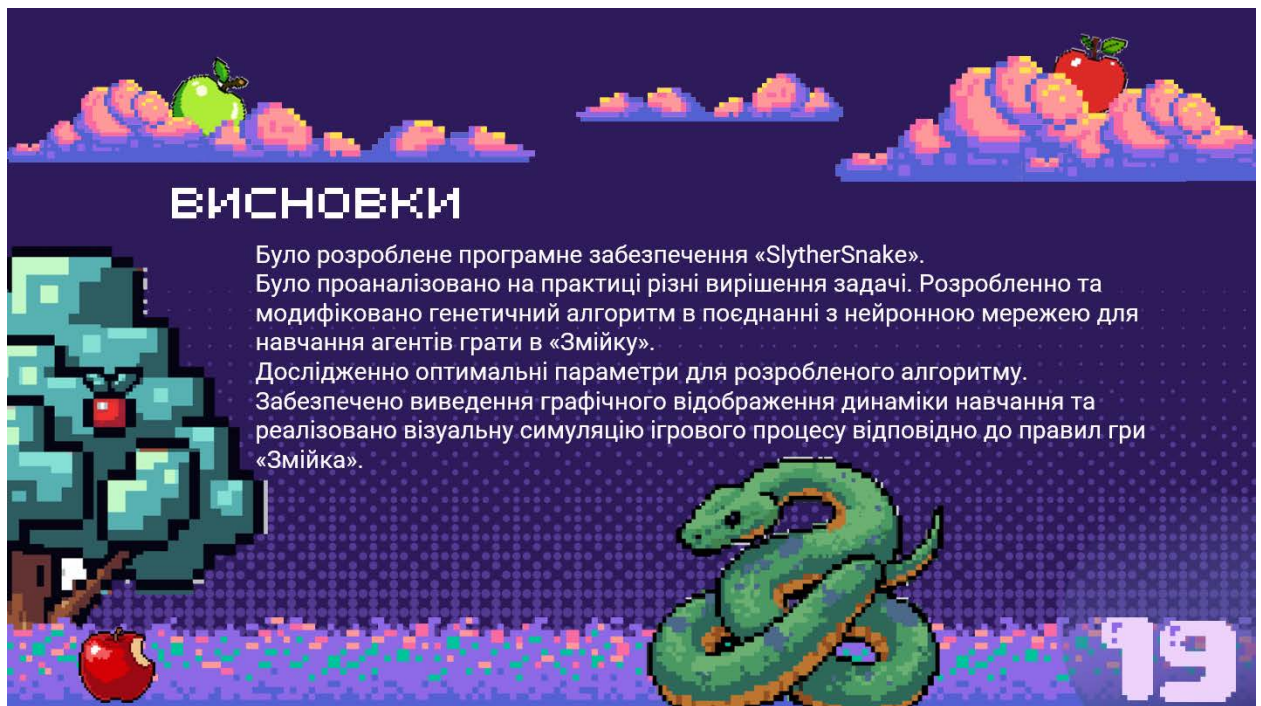


Рисунок В.19 –Дев’ятнадцятий слайд

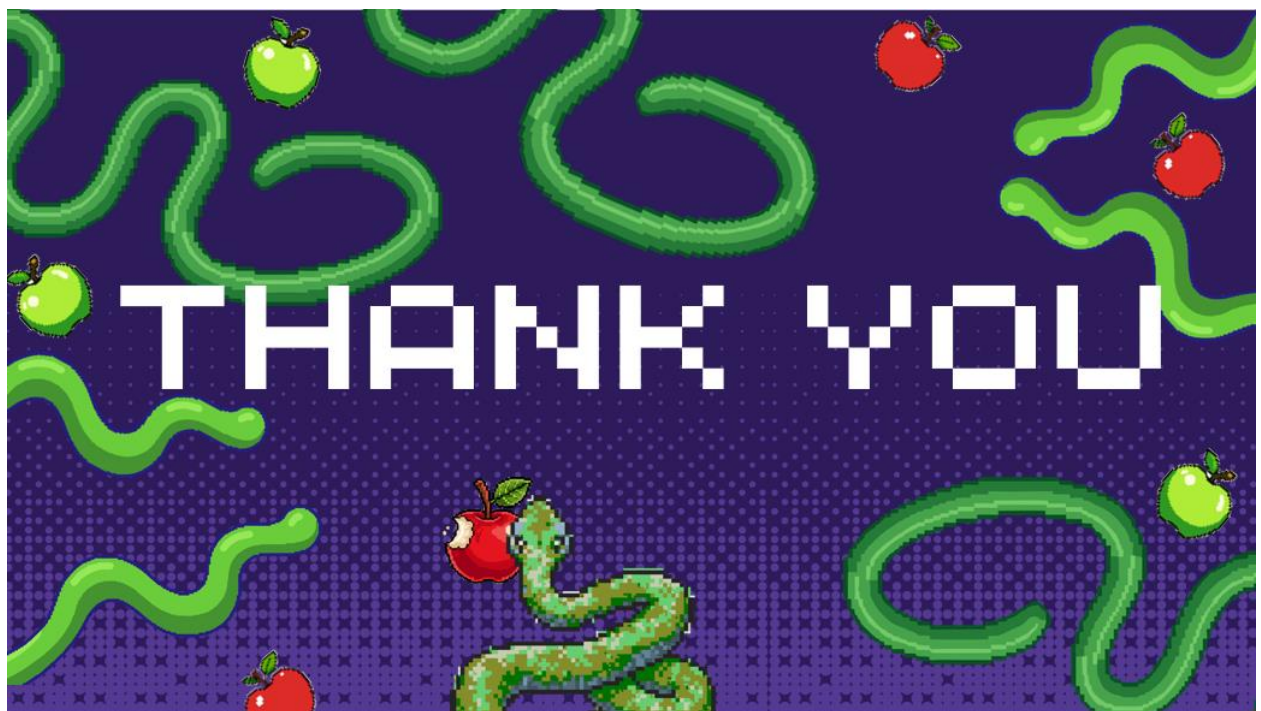


Рисунок В.20 –Двадцятий слайд