Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“___” _____ 20__ р.

# ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
**за другим (магістерським) рівнем вищої освіти**
на тему
**“Дослідження та програмна реалізація нейронних мереж для управління ігровими об’єктами”**



Виконав здобувач вищої освіти
ІІ курсу, групи КІ-22М-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Конончук М.С.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Єлизавета МЕЛЕШКО
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

**Центральноукраїнський національний технічний університет**

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____Олексій СМІРНОВ
« __ » _____ 20__ року

# ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

*Кононучуку Максиму Сергійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація нейронних мереж для управління ігровими об'єктами*

2. Керівник роботи *Мелешко Єлизавета Владиславівна, доктор техн. наук, професор*
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу №__-__ від __.__.20__ року

3. Строк подання роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Дослідження та програмна реалізація нейронних мереж для управління ігровими об'єктами*

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

*1. Призначення та область використання.*
*2.Перегляд аналогічних існуючих систем.*
*3. Опис і обґрунтування проектних рішень.*
*4. Етапи програмування системи.*
*5. Впровадження системи в промислову експлуатацію.*
*6. Наукова новизна*
*7. Економічна ефективність розробленої програми.*
*8. Заходи з охорони праці та техніки безпеки.*
*9. Висновки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

| | |
|---|---|
| *Наукова новизна* | *1 аркуш* |
| *Структурна схема системи* | *1 аркуш* |
| *Функціональна схема системи* | *1 аркуш* |
| *Блок-схема алгоритму роботи додатку* | *2 аркуша* |
| *Діаграма процесів* | *1 аркуш* |
| *Показники економічної ефективності* | *1 аркуш* |

6. Консультанти по роботі, із зазначенням розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Економічний | Савеленко Г.В., к.т.н., доцент | 25.10.2023 | 10.11.2023 |
| Охорона праці | Оришака О.В., к.т.н., доцент | 03.11.2023 | 21.11.2023 |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти | Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти | Примітка |
|---|---|---|---|
| 1. | Аналіз існуючих систем | 10.10.2023 р. | |
| 2. | Постановка задачі, оформлення ТЗ | 16.10.2023 р. | |
| 3. | Розробка моделі компонента | 20.10.2023 р. | |
| 4. | Розробка структур даних | 25.10.2023 р. | |
| 5. | Розробка алгоритмів зв’язку та відображення | 30.10.2023 р. | |
| 6. | Програмування алгоритмів | 10.11.2023 р. | |
| 7. | Розрахунок економічної ефективності | 13.11.2023 р. | |
| 8. | Розрахунки з охорони праці та техніки безпеки | 15.11.2023 р. | |
| 9. | Оформлення ПЗ | 17.11.2023 р. | |
| 10. | Попередній захист роботи | 10.12.2023 р. | |
| | | | |



Дата видачі завдання
«__»_____20 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
«__»_____20 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

# АНОТАЦІЯ

**Конончук М.С. Дослідження та програмна реалізація нейронних мереж для управління ігровими об'єктами. Центральноукраїнський національний технічний університет. Кропивницький. 2023.**

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації системи моделювання комп'ютерних мереж та процесів передачі даних.

Метою розробки є дослідження та програмна реалізація нейронних мереж для управління ігровими об'єктами.

Об'єктом дослідження є процес управління об'єктами у віртуальному ігровому середовищі.

Предметом дослідження є методи управління ігровими об'єктами за допомогою багатошарових нейронних мереж.

Методи дослідження базуються на теорії штучного інтелекту, теорії математичної статистики, методах розробки програмного забезпечення, методах об'єктно-орієнтованого програмування та методах розробки комп'ютерних ігор.

Результат роботи – програмна реалізація нейронних мереж для управління ігровими об'єктами.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування Python у середовищі розробки програмного забезпечення Anaconda.

**Ключові слова**: комп'ютерна інженерія, нейронні мережі, комп'ютерні ігри, управління ігровими об'єктами, інтелектуальна система.

# ABSTRACT

**Kononchuk M.S. Research and software implementation of neural networks for controlling game objects. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2023.**

In this master's thesis, software designed neural networks for controlling game objects.

The purpose of the development is the research and program implementation of neural networks for controlling game objects.

The object of the research is the process of controlling objects in a virtual gaming environment.

The subject of the research is the methods of controlling game objects using multilayer neural networks.

Research methods are based on the theory of artificial intelligence, the theory of mathematical statistics, software development methods, object-oriented programming methods, and computer game development methods.

The result of the work is the software implementation of neural networks for controlling game objects.

In the process of working on a software model an analysis of existing hardware and software was performed. All components of the software developed are fully described.

User-friendly user interface is developed. These are instructions for working with software.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the high-level programming language – Python, in the software development environment – Anaconda.

**Keywords**: computer engineering, neural networks, computer games, controlling game objects, intelligent system.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

**CPU (Central Processing Unit) –** центральний процесор, основний компонент комп'ютера, який виконує інструкції програмного забезпечення та керує іншими апаратними компонентами.

**GPU (Graphics Processing Unit) –** графічний процесор, пристрій, оптимізований для обробки графіки та паралельних обчислень.

**Батч (batch) –** підмножина навчального набору даних, яка використовується для тренування моделі нейронної мережі в одній ітерації.

**БД** – база даних; організований набір електронних даних.

**Інтелектуальний агент –** програмний агент, який має здатність до навчання, адаптації та виконання завдань у певному середовищі; часто використовуються в областях, де потрібно автоматичне прийняття рішень.

**Навчальне середовище –** у контексті машинного навчання, це середовище, в якому моделі штучного інтелекту (наприклад, нейронні мережі) вчаться через взаємодію з середовищем, отримуючи винагороди за вірні дії та покарання за невірні.

**ОС** – операційна система; програмне забезпечення, яке керує апаратними та програмними ресурсами комп'ютера та надає загальні послуги для комп'ютерних програм.

**Тензор –** у контексті програмування та машинного навчання, це багатовимірний масив даних; тензори використовуються для представлення даних, які подаються на вхід або виходять з моделі нейронної мережі.

**Штучна нейронна мережа –** комп'ютерна модель, створена за аналогією до структури та функцій біологічних нейронних мереж, яка використовується для вирішення складних задач, таких як розпізнавання образів, обробка мови, прогнозування та управління об'єктами.

# ВСТУП

**Актуальність теми**. В останні роки галузь штучного інтелекту зазнала значного прогресу, зокрема у сфері машинного навчання та нейронних мереж. Цей розвиток відкриває нові можливості для застосування штучного інтелекту у різноманітних областях, включаючи індустрію комп'ютерних ігор.

Сучасні гравці прагнуть до більш реалістичних і занурювальних ігрових досвідів. Управління ігровими об'єктами за допомогою нейронних мереж може значно підвищити реалізм та адаптивність ігор, що є ключовим фактором у задоволенні запитів сучасних гравців. Індустрія комп'ютерних ігор постійно шукає нові технологічні рішення для створення інноваційних продуктів. Використання нейронних мереж для управління ігровими об'єктами відкриває нові горизонти для геймдизайну та розробки ігор.

Адаптивні ігрові системи, здатні реагувати на дії та зміни у поведінці гравців, стають все більш актуальними. Нейронні мережі можуть забезпечити високий рівень адаптивності, роблячи ігровий процес більш гнучким та відповідним до індивідуальних переваг гравців.

Розвиток та впровадження нейронних мереж у іграх також сприяє загальному прогресу в галузі штучного інтелекту, відкриваючи нові можливості для досліджень та розвитку технологій.

**Мета й завдання дослідження.** Метою роботи є дослідження та програмна реалізація нейронних мереж для управління ігровими об'єктами.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих підходів та методів для управління ігровими об'єктами.

– Розробка методів та алгоритмів системи управління ігровими об'єктами на основі нейронних мереж.

– Програмна реалізація комп’ютерної гри та системи управління ігровими об’єктами на основі нейронних мереж.

*Об’єктом дослідження* є процес управління об’єктами у віртуальному ігровому середовищі.

*Предметом дослідження* є методи управління ігровими об’єктами за допомогою багатошарових нейронних мереж.

*Методи дослідження* базуються на теорії штучного інтелекту, теорії математичної статистики, методах розробки програмного забезпечення, методах об’єктно-орієнтованого програмування та методах розробки комп’ютерних ігор.

**Наукова новизна отриманих результатів.** У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод управління ігровими агентами, що є супротивниками гравця у казуальній аркадній грі, за допомогою багатошарових нейронних мереж та безперервного навчання у віртуальному ігровому середовищі.

2. Розроблено вітчизняний продукт для інтелектуального управління ігровими об’єктами за допомогою нейронних мереж, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний при розробці комп’ютерних ігор.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтелектуального управління агентами-супротивниками у комп’ютерній грі.

**Достовірність наукових результатів** підтверджена теоретичними викладками, результатами тестування розробленого програмного забезпечення, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи управління ігровими об’єктами на основі нейронних мереж, є актуальною задачею, яка потребує вирішення у даній магістерській роботі.

# 1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

## 1.1 Призначення системи

Програмне забезпечення нейронних мереж для управління ігровими об’єктами призначене для створення, навчання та інтеграції нейронних мереж у комп’ютерні ігри з метою підвищення інтерактивності, реалістичності та адаптивності ігрових сценаріїв. Система може використовуватися для розробки ігор, де поведінка ігрових персонажів та елементів управляється штучним інтелектом.

Ключові функції такого програмного забезпечення:

– Автоматизоване управління ігровими агентами та об’єктами. Використання нейронних мереж дозволяє автоматично керувати поведінкою ігрових об’єктів, забезпечуючи більш реалістичну та динамічну гру.

– Адаптація до дій гравців. Система дозволяє ігровим об’єктам адаптуватися до дій та стратегій гравців, зробивши ігровий процес більш складним та непередбачуваним.

– Поліпшення ігрового дизайну. Використання нейронних мереж може допомогти розробникам ігор створювати більш складні ігрові сценарії та дизайни рівнів.

Потенційне застосування розроблюваної системи:

– Комп’ютерні ігри. Створення ігор з використанням штучного інтелекту, що забезпечує нові та унікальні ігрові враження.

– Тренувальні симулятори. Використання у тренувальних симуляторах різних сфер діяльності для моделювання реалістичних сценаріїв.

– Навчальні платформи. Розробка навчальних ігор, де адаптивна поведінка ігрових об’єктів може підвищити ефективність навчання.

Ця система відкриває широкі можливості для інновацій у сфері розробки ігор, забезпечуючи більш глибоку взаємодію з ігровим середовищем та надаючи розробникам потужний інструмент для створення більш продуманих та вражаючих ігрових досвідів.

### 1.2 Область застосування

Розроблену систему можна використовувати у наступних сферах діяльності:

– Розробка комп’ютерних та мобільних ігор.

– Розробка ігрових та навчальних симуляторів.

– Дослідження в галузі штучного інтелекту.

Можливі користувачі розробленої системи:

– ігрові розробники та дизайнери;

– розробники у галузі ігрових та навчальних симуляцій;

– освітні інституції та розробники навчальних програм;

– дослідники в галузі штучного інтелекту.

Програмне забезпечення нейронних мереж для управління ігровими об’єктами відкриває широкі можливості для різних галузей та професій, що займаються розробкою цифрових інтерактивних продуктів та дослідженнями у галузі штучного інтелекту, пропонуючи інноваційні підходи до створення динамічних та реалістичних ігрових середовищ.

# 2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

## 2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи

Нейронні мережі в іграх застосовуються вже досить давно, але досі це лише дуже слабкі спроби впровадження. Тому що контролювати навчання нейронних мереж в іграх дуже важко. Хоча потенціал застосування нейромереж досить великий. Тому їх використання спочатку несе непередбачуваний ефект. З одного боку, вони можуть відмінно вдосконалювати гру, але з іншого боку, вони можуть повністю зруйнувати її геймплей, тому що алгоритми починають дуже дивно поводитися.

Нейронна мережа в іграх використовується для об’єктів у грі, які не керуються людиною, тобто для ботів. Суть застосування зводиться до того, щоб боти вели себе більш природно при ігровому сценарії, що змінюється навколо них.

Нейронна мережа в іграх навчається динамічно під час ігрового процесу. Саме таке динамічне навчання важко піддається контролю. Плюс динамічне навчання ботів потребує постійного підключення користувачів до гри та постійного обміну інформацією з ігровими серверами. А це дуже сильно знижує продуктивність ігор і потребує більш високих характеристик апаратного забезпечення користувачів. Але найважливіше – динамічне навчання нейронних мереж мало контролюється розробниками ігор.

На сьогоднішній день можна розрізнити декілька способів застосування нейронних мереж в іграх:

– *Нейронна мережа на основі набору обмежуючих правил*. Це найпростіша форма застосування нейронних мереж, коли поведінка ботів регулюється заздалегідь запрограмованими алгоритмами, які «включаються» за певних

характеристик ігрового сценарію.

– *Адаптивна нейронна мережа*. Це вже складніша система для складних ігор. Вона застосовується, коли у гру потрібно внести «краплю» непередбачуваності. Такі боти в іграх також запрограмовані алгоритмами поведінки. Але алгоритми активуються з різною інтенсивністю і різною силою, тому що їх застосування залежить не просто від ігрового сценарію, а ще й від інших індикаторів, наприклад: ігровий час, що залишився, наявність переваг у суперника, стан здоров'я ігрового об'єкта, наявність додаткових бонусів, досягнення гравця у попередніх боях тощо. Таким чином, бот здатний максимально адаптуватися під кожного окремого «живого» гравця, враховуючи навіть його стиль гри.

– *Повна свобода*. Таке застосування нейромереж в іграх вважається експериментальним варіантом. У цьому випадку боти в грі лише програмуються алгоритмами для обробки інформації, що отримується. Їм довіряється повна свобода у її інтерпретації та прийнятті рішень. Саме таке застосування штучного інтелекту в іграх є практично неконтрольованим та непередбачуваним.

Також є випадки, коли застосування нейронних мереж в комп'ютерних іграх є непрямим. Наприклад:

– *Нейронні мережі для оптимізації ігор*. Вимоги до ігор постійно змінюються, тому ігри стають дедалі складнішими і важчими. Хоча зараз і існує безліч технологій для швидкої розробки ігор, все одно сам процес залишається дуже нудним і довгим, як і в 90-х або 80-х роках. «Тяжкість» сучасних ігор накладає свої відбитки на характеристики комп'ютерів користувачів – потрібні пристрої потужніше. Нейронна мережа дозволяє використовувати потужні та важкі ігри на слабких комп'ютерах. Наприклад, у грі ви встановлюєте налаштування вашого комп'ютера або задаєте числове значення fps, а нейронна мережа самостійно оптимізує гру під ваш комп'ютер або під ваші налаштування, природно урізуючи її візуальну якість. Другий метод оптимізації – це коли об'єкти з гри, сцени, карти тощо пропускають через нейронну мережу, щоб

надати їм максимальну реалістичність.

– *Генерація квестів*. Таке застосування нейронної мережі стає доступним в іграх з відкритим світом, де штучний ітелект може самостійно генерувати шматочки карт, квести, завдання гравцям і т.д.

– *Генерація рівнів*. Таке застосування нейронних мереж можна побачити в іграх-симуляторах, коли в симулятор переноситься реальне місто з усіма будівлями, дорогами, інфраструктурою і т.д. Реальна карта в грі – результат роботи штучного інтелекту. Якщо все це робити «руками», то знадобиться багато часу.

– *Нейронна мережа для домалювання*. У момент дизайну зовнішнього вигляду карт гри часто застосовується нейронна мережа. Суть її роботи полягає в тому, щоб «домальовувати» за дизайнером. Наприклад, дизайнер малює карту загалом, а нейронна мережа опрацьовує всі дрібні деталі.

– *Нейронна мережа – модератор*. Нейронні мережі вчаться контролювати та модерувати поведінку гравців у іграх та спільних чатах, щоб зняти таку рутинну роботу з живих людей.

– *Реставрація ігор*. Нейронні мережі застосовуються, коли потрібно покращити якість старої цікавої гри. Працює це за простим принципом: стару гру пропускають через нейронну мережу і на виході отримують покращену модель старої гри.

Розглянемо ігри, що уже використовують штучні нейронні мережі. Станом на квітень 2023 року, нейронні мережі все частіше використовуються в ігровій індустрії, але зазвичай деталі їхнього використання в іграх не розкриваються широкому загалу. Однак, є декілька відомих прикладів та експериментів, де нейронні мережі застосовувались для управління супротивниками або об’єктами:

**StarCraft II.** DeepMind, підрозділ компанії Google, розробив систему штучного інтелекту на ім'я AlphaStar, яка здатна грати в "StarCraft II" на рівні професійних гравців. AlphaStar використовує нейронні мережі для ухвалення стратегічних рішень та мікроуправління юнітами.

Рисунок 2.1 – Приклад ігрового процесу у StarCraft II

**Dota 2.** OpenAI розробила систему під назвою OpenAI Five, яка може грати в Dota 2 проти професійних команд. Ця система також використовує нейронні мережі для прийняття рішень, планування дій та координації між різними героями в грі.

Рисунок 2.2 – Приклад ігрового процесу у StarCraft II

**The Last of Us Part II.** У цій грі нейронні мережі використовуються для підвищення реалістичності поведінки неігрових персонажів (NPC). Це дозволяє NPC адаптуватися до дій гравця та приймати більш складні рішення.

Рисунок 2.3 – Приклад ігрового процесу у The Last of Us Part II

**Middle-earth: Shadow of Mordor** та **Shadow of War.** Ці ігри використовують систему «Nemesis», яка адаптує поведінку NPC в залежності від дій гравця, використовуючи елементи машинного навчання для створення унікальних вражень від взаємодії з персонажами гри.

Рисунок 2.4 – Приклад ігрового процесу у Middle-earth: Shadow of Mordor

Рисунок 2.5 – Приклад ігрового процесу у Shadow of War

Хоча використання нейронних мереж у комерційних іграх все ще є відносно новим і в багатьох випадках експериментальним, це напрямок набирає популярності і має великий потенціал для розвитку в майбутньому.

Британська компанія DeepMind, яка займається питаннями розробки штучного інтелекту, нещодавно показала, яким чином ІІ можуть самостійно вчитися грати в ігри, засвоювати їх правила та знаходити способи пройти гру або перемогти в ній – правда, поки що лише на прикладі простих ігор, таких як ранні ігри Atari – наприклад, шахи та японська логічна гра. Результати, отримані їм, показують, що штучний інтелект може сформувати собі адекватну оцінку що відбувається на полі. Якщо ж говорити про адаптацію ІІ до різних стилів гри опонента, то результати поки що не такі вражаючі.

Для створення нейронних мереж існує декілька відомих бібліотек, які можуть бути корисними при розробці штучного інтелекту для ігор. Ось декілька з них:

– TensorFlow – це одна з найпопулярніших бібліотек для машинного навчання, розроблена Google. TensorFlow підходить для широкого спектру

завдань, включаючи глибоке навчання та нейронні мережі. Вона має гнучкий API, що дозволяє інтегрувати її з ігровими двигунами.

– Keras – це високорівневий API, який працює на основі TensorFlow. Keras робить процес проектування та тренування моделей більш інтуїтивно зрозумілим і менш вимогливим до коду. Його можна використовувати для швидкого прототипування.

– Torch – розроблений Facebook, Torch широко використовується у наукових дослідженнях. Він пропонує легку інтеграцію, що робить його дуже зручним для експериментів і прототипування нейронних мереж.

– Microsoft Cognitive Toolkit (CNTK) – ця бібліотека від Microsoft оптимізована для високопродуктивних завдань машинного навчання. Вона особливо добре підходить для масштабних та розподілених навчальних середовищ.

## 2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Було прийнято рішення реалізувати систему на мові високого рівня Python у середовищі розробки Anaconda та з застосуванням бібліотек pygame, gym, torch. Бібліотека pygame використана для створення казуальної аркадної комп'ютерної гри та інтерфейсу користувача. Бібліотека gym використана для створення віртуального навчального середовища для інтелектуальних агентів – супротивників гравця в розробленій казуальній аркадній грі. Бібліотека torch використана для реалізації нейронних мереж, які здійснюють управління агентами у комп'ютерній грі.

Мова програмування Python була обрана з наступних причин:

– Підтримка та спільнота. Python – одна з найпопулярніших мов програмування з великою та активною спільнотою. Це забезпечує доступ до великої кількості ресурсів, документації та форумів для вирішення проблем.

– Бібліотеки для штучного інтелекту та машинного навчання. Мова Python відома своїми потужними бібліотеками для машинного навчання та штучного інтелекту, що є ключовим для розробки та дослідження нейронних мереж для управління ігровими об’єктами.

– Гнучкість та читабельність. Мова Python відома своєю гнучкістю та легкістю читання, що спрощує процес розробки та спільної роботи над проектами.

Бібліотека pygame була обрана з наступних причин:

– Інтерфейс для розробки ігор. Pygame є простим у використанні інтерфейсом для створення ігор на Python. Це дозволяє швидко розробляти прототипи та ігрові додатки.

– Підтримка різних ігрових елементів. Pygame підтримує графіку, звук, та інші ігрові елементи, що необхідні для створення комплексного ігрового середовища.

– використання в навчанні. Pygame дозволяє легко інтегрувати нейронні мережі для управління ігровими об’єктами, що робить її ідеальною для виконання поставлених задач.

Бібліотека gym була обрана з наступних причин:

– Симуляційне середовище для навчання агентів. Gym від OpenAI надає широкий спектр симуляційних середовищ для тренування та тестування алгоритмів штучного інтелекту.

– Стандартизація тестування. Використання Gym дозволяє стандартизувати процес навчання та тестування агентів, забезпечуючи консистентність та порівняльність результатів.

– Легка інтеграція з іншими бібліотеками. Gym легко інтегрується з іншими бібліотеками машинного навчання, зокрема з torch, що є важливим для розроблюваної системи.

Бібліотека torch була обрана з наступних причин:

– Гнучкість у розробці нейронних мереж. Torch відома своєю гнучкістю та інтуїтивно зрозумілою архітектурою, що дозволяє ефективно розробляти та налаштовувати нейронні мережі.

– Підтримка глибокого навчання. Як одна з провідних бібліотек для глибокого навчання, torch забезпечує потужні інструменти для створення та навчання складних моделей нейронних мереж.

– Спільнота та ресурси. Torch підтримується великою спільнотою розробників, що забезпечує доступ до великої кількості навчальних матеріалів та прикладів коду.

Середовище розробки програмного забезпечення Anaconda було обране через наступні його переваги:

– Інтегроване середовище для розробки додатків та аналізу даних. Anaconda є одним з найпопулярніших дистрибутивів Python, спеціально розробленим для науки про дані, машинного навчання та наукових досліджень. Це робить його ідеальним вибором для проектів, які включають роботу з нейронними мережами та обробку даних.

– Широкий набір попередньо встановлених бібліотек. Anaconda включає велику кількість попередньо встановлених бібліотек, таких як numpy, pandas, scipy, matplotlib, та інших, що спрощує процес налаштування середовища для розробки.

– Управління залежностями та віртуальними середовищами. Anaconda надає потужні інструменти для управління залежностями та віртуальними середовищами, що дозволяє легко створювати та управляти ізольованими середовищами для різних проектів.

– Сумісність з різними ОС. Anaconda підтримує різні операційні системи, включаючи Windows, macOS та Linux, що робить його доступним для широкого кола розробників.

– Інтеграція з інструментами розробки. Anaconda легко інтегрується з популярними інструментами розробки та середовищами, такими як Jupyter

Notebook, PyCharm, Visual Studio Code, що забезпечує гнучкість та зручність у використанні.

– Спрощення командної роботи. Завдяки своїй стандартизації та широкому використанню, Anaconda спрощує спільну роботу над проектами, оскільки всі члени команди можуть мати однакове налаштоване середовище розробки.

– Підтримка дата-інженерії. Anaconda підтримує інструменти для дата-інженерії, що є критично важливим для проектів, де потрібно обробляти та аналізувати великі обсяги даних, як це часто буває у галузі штучного інтелекту та машинного навчання.

– Ресурси для навчання та підтримки. Anaconda пропонує широкий спектр ресурсів для навчання та підтримки, включаючи документацію, навчальні матеріали, та активну спільноту користувачів.

Вибір мови програмування Python разом з бібліотеками pygame, gym та pytorch для дослідження та програмної реалізації нейронних мереж для управління ігровими об’єктами заснований на їх гнучкості, підтримці спільноти, ефективності у використанні для розробки ігор та можливостях у сфері машинного навчання і штучного інтелекту. Ці інструменти забезпечують ідеальну основу для розробки, навчання та інтеграції нейронних мереж в ігровому середовищі. Використання середовища розробки Anaconda дає можливість максимально ефективно використовувати ресурси Python, забезпечує легкість управління залежностями та сприяє спрощенню роботи. Це робить Anaconda ідеальним вибором для високоефективної та гнучкої розробки в галузі штучного інтелекту.



### 2.3 Розгорнута постановка завдання

Ця робота має на меті дослідження та розробку нейронних мереж для управління ігровими об’єктами в комп’ютерних іграх. Вона передбачає створення

нейронної мережі, її навчання та інтеграцію з ігровим середовищем для управління поведінкою ігрових об’єктів, таких як супротивники гравця.

Основні задачі, які потребують виконання у даній роботі:

1. Огляд літератури та дослідження теоретичних основ:

– Дослідження сучасних методів та технік використання нейронних мереж у геймдеві.

– Аналіз можливостей та обмежень нейронних мереж для управління ігровими об’єктами.

2. Розробка програмного забезпечення:

– Проектування архітектури нейронної мережі, оптимальної для управління ігровими об’єктами.

– Розробка програмного коду для реалізації нейронної мережі.

– Розробка казуальної аркадної гри як начального та ігрового середовища для нейронної мережі.

– Інтеграція нейронної мережі для управління ігровими об’єктами у комп’ютерну гру.

3. Навчання та тестування нейронної мережі:

– Одержання даних нейронною мережею з ігрового середовища.

– Вибір дій нейронною мережею для виконання агентом, яким вона управляє.

– Одержання винагород чи покарань від ігрового середовища.

– Підлаштування вагових коефіцієнтів нейронної мережі на основі отриманих винагород чи покарань за виконані дії.

– Тестування ефективності мережі в управлінні ігровими об’єктами в різних сценаріях гри.

4. Аналіз результатів та оптимізація:

– Оцінка ефективності мережі в різних ігрових умовах.

– Вдосконалення алгоритму та параметрів нейронної мережі на основі аналізу результатів.

5. Документування та презентація результатів:

Підготовка пояснювальної записки кваліфікаційної магістреської роботи, включаючи опис алгоритмів, архітектуру мережі та результати тестування.

Очікувані результати:

– Розроблена та налаштована нейронна мережа для ефективного управління ігровими об’єктами.

– Демонстрація покращення поведінки ігрових об’єктів завдяки застосуванню нейронних мереж.

– Документація та звіт про роботу, який включає детальний опис процесу розробки, навчання мережі, тестування та отримані результати.

– Підтвердження потенціалу використання нейронних мереж у розробці ігор та управлінні поведінкою ігрових об’єктів.

Ресурси:

– Програмне забезпечення для розробки та начання нейронних мереж.

– Програмне забезпечення для розробки комп’ютерної гри та інтеграції і тестування нейронної мережі.

# 3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

## 3.1 Опис функціонування системи

Для розробки багатошарової нейронної мережі було використано бібліотеку torch. Комп'ютерна гра з «агентами», якими має управляти нейронна мережа, створена з використанням бібліотеки pygame. Інтеграція нейронної мережі в комп'ютерну гру здійснена з використанням бібліотеки gym.

Розглянемо принципи створення та роботи з нейронними мережами при використанні бібліотеки torch.

Прості нейронні мережі з послідовною архітектурою можна створювати за допомогою стопкового інтерфейсу. Наприклад, двошарова повнозв'язна мережа з двома входами (*nX*=2 ознаки), одним (*nY*=1 ) сигмоїдним виходом (2 класу) і п'ятьма нейронами (*nH*=5 ) у прихованому шарі має вигляд (рис. 3.1):

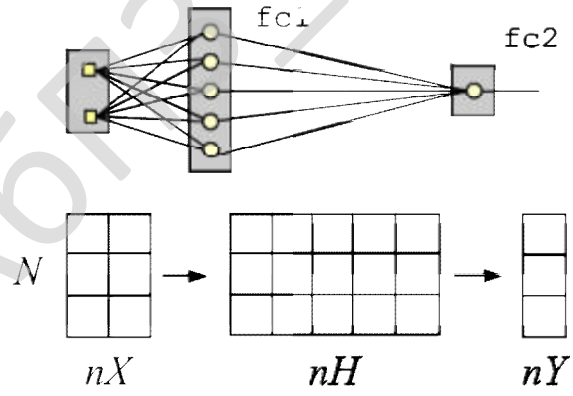


Рисунок 3.1 – Двошарова повнозв'язна нейронна мережа

Створити таку мережу можна за допомогою наступного коду:

```
import torch
import torch.nn as nn

nX, nH, nY = 2, 5, 1
```

```
model = nn.Sequential(
          nn.Linear(nX, nH),     # перший прошарок
          nn.Sigmoid(),          # активація прихованого прошарку
          nn.Linear(nH, nY),     # другий, вихідний прошарок
          nn.Sigmoid() )         # функція активації другого прошарку
```

Прошарок Linear робить лінійне перетворення вхідного тензора форми (*N*, *nX*) де *N* – число прикладів. На виході шару виходить тензор форми (*N*, *nH*) який пропускається через сигмоїд (нові *nH* ознаки). Другий шар дає тензор (*N*, *nY*) , від якого значення також береться через сигмоїду. Наприклад, якщо його значення менше 0.5 – це перший клас, якщо більше 0.5 – другий.

При проектуванні складних нейронних мереж їх архітектуру зручніше задавати у функціональному вигляді.

Для цього створюється спадкоємець класу nn.Module. У ньому перевизначається конструктор і метод forward. У конструкторі задаються необхідні шари та ініціалізуються їх параметри. Роздянемо реалізацію виклику конструктора предка якому необхідно передати ім’я класу. У методі forward (пряме поширення) будується архітектура мережі (шари зв’язуються в обчислювальний граф):

```
class TwoLayersNet(nn.Module):
    def __init__(self, nX, nH, nY):
        super(TwoLayersNet, self).__init__()  # конструктор предка з цим
ім’ям
        self.fc1 = nn.Linear(nX, nH)      # створюємо параметри моделі
        self.fc2 = nn.Linear(nH, nY)      # у повнозв’язних шарах

    def forward(self, x):                 # задається прямий прохід
        x = self.fc1(x)                   # вихід першого шару
        x = nn.Sigmoid()(x)               # пропускаємо через Sigmoid
        x = self.fc2(x)                   # вихід другого шару
        x = nn.Sigmoid()(x)               # пропускаємо через сигмоїд
        return x

model = TwoLayersNet(2, 5, 1)             # екземпляр мережі
```

Виклик nn.Line(in_features, out_features) створює два екземпляри лінійного повнозв'язного шару fc1, fc2 (fully connected). Одночасно з цим задаються випадкові значення їх ваги та зсувів. Скобочний оператор fc1(x) у методі forward запускає обчислення всередині шару і на виході видає результуючий тензор.

Створимо на torch модельні дані для об'єктів двох видів (два класи), що характеризуються двома ознаками. Нехай на 2D площині об'єкти першого класу заповнюють одиничний квадрат. $[0...1]^2$ крім кола в центрі квадрата, де знаходяться об'єкти другого класу:

```
X = torch.rand (1200,2)
Y = (torch.sum((X - 0.5)**2, axis=1) < 0.1).float().view(-1,1)
```

Метод float перетворює логічні значення оператора менше на дійсні числа (0.0 або 1.0). Потім за допомогою view перетворюємо Y на матрицю з одного стовпчика (в якому знаходиться 0 або 1). По рядках йдуть класи прикладів.

За допомогою бібліотеки matplotlib намалюємо те, що в результаті вийшло:

```
import matplotlib.pyplot as plt      # побудова графіків
plt.figure (figsize=(5, 5))          # розміри (квадрат)
plt.scatter(X.numpy()[:,0], X.numpy()[:,1], c=Y.numpy()[:,0],
            s=30, cmap=plt.cm.Paired, edgecolors='k')
plt.show()                                  # виводимо малюнок
```

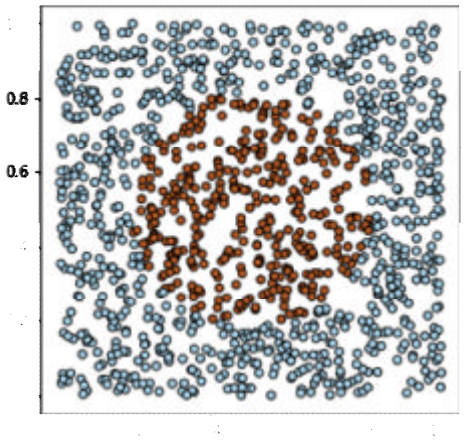


Рисунок 3.2 – Приклад модельних даних для нейронної мережі

Для навчання мережі необхідно створити функцію помилки (те, що слід мінімізувати) та оптимізатор (який мінімізуватиме помилку). Оптимізатору передаються параметри моделі. Як помилку виберемо бінарну крос-ентропію (BCELoss), а оптимізатором буде SGD (Stochastic Gradient Descent):

```
model = TwoLayersNet(2, 5, 1)     # екземпляр мережіи
loss      = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(),    # параметри моделі
                            lr=0.5, momentum=0.8)   # параметри оптимізатора
```

На кожній ітерації з навчальних прикладів формується масив батчів, запускається пряме поширення y = model(bx) і обчислюється помилка loss (з порівняння виходу моделі з "правильними" значеннями yb.

На обчислювальному графі помилки метод loss.backward() дає градієнти параметрів моделі, за допомогою яких оптимізатор optimizer методом step() отримує нові значення параметрів:

```
def fit(model, X,Y, batch_size=100, train=True):
      model.train(train)          # важливо для Dropout, BatchNorm
      sumL, sumA, numB = 0, 0, int( len(X)/batch_size )  # помилка,
точність, батч

      for i in range(0, numB*batch_size, batch_size):
        xb = X[i: i+batch_size]                # поточний батч,
        yb = Y[i: i+batch_size]                # X,Y - torch тензори

        y = model(xb)                          # пряме поширення
        L = loss(y, yb)                        # обчислюємо помилку

        if train:                              # у режимі навчання
            optimizer.zero_grad()              # обнулюємо градієнти
            L.backward()                       # обчислюємо градієнти
            optimizer.step()                   # підправляємо параметри

        sumL += L.item()                  # сумарна помилка (item із графа)
        sumA += (y.round() == yb).float().mean()# точність визначення класу
    return sumL/numB,  sumA/numB            # середня помилка та точність
```

Функція fit робить прохід за всіма даними. Це називається епохою навчання. Якщо параметр leran=True, відбувається навчання моделі (зміна її параметрів). Значення leran=False використовується для оцінки якості моделі без її зміни. Як метрик якості використовується середня помилка за всіма прикладами і середня точність (частка правильно передбачених класів). Зазвичай потрібно досить багато епох навчання:

```
    # Режим оцінки моделі:
    print( "before:      loss: %.4f accuracy: %.4f" %  fit(model, X,Y,
    train=False) )
       epochs = 1000                                          # число епох
       for epoch in range(epochs):                            # епоха - прохід
за всіма прикладами
           L,A = fit(model, X, Y)                             # одна епоха
           if epoch % 100 == 0 or epoch == epochs-1:
               print(f'epoch: {epoch:5d} loss: {L:.4f} accuracy: {A:.4f}' )
       В результаті отримаємо щось на кшталт:
       before:      loss: 0.6340 accuracy: 0.6950
       epoch:     0 loss: 0.6327 accuracy: 0.6550
       ....
       epoch:   999 loss: 0.0334 accuracy: 0.9908
```

Для поліпшення навчання зазвичай варто перемішувати дані. Це можна зробити двома способами. Перший запускається перед функцією fit на початку кожної епохи:

```
       idx = torch.randperm( len(X) )      # перемішаний список індексів
       X = X[idx]
       Y = Y[idx]
```

У цьому вся методі створюється нова пам'ять всім даних, що з великих навчальних масив який завжди добре. Другий спосіб робить випадкову вибірку тільки для поточного батча всередині функції fit :

```
       idx = torch.randint(high = len(X), size = (batch_size,) )
       xb = X[idx]
       yb = Y[idx]
```

Пам’яті при цьому використовується менше, але при проході по одній епосі навчання можуть потрапити не всі приклади, а в батч можуть потрапити однакові приклади. Цей метод використовується в класі DataLoader (параметр shuffle).

Таким чином, порівняно з Keras у TensorFloor, процедуру навчання необхідно написати самостійно. Однак у складних випадках це дозволяє втручатися у процес будь-якому етапі. Наприклад, можна вставити свій оптимізатор:

```
if train:                               # у режимі навчання
            L.backward()                # обчислюємо градієнти
            with torch.no_grad():
                for p in model.parameters():
                    p.add_(p.grad, alpha=-0.7)          # p += -0.7*grad
                    p.grad.zero_()
```

Розглянемо способи візуалізації структури нейронної мережі.

Найпростіший спосіб побачити шари нейронної мережі, це просто вивести на друк модель:

```
print(model)     # текстовое представление модели
```

Зручніше скористатися стороннім модулем modelsummary. У його функцію summary передається вхідний тензор з будь-якими значеннями (але правильної форми) з одного прикладу (у shape це буде перша розмірність -1):

```
from modelsummary import summary
summary(model, torch.zeros(1, 2), show_input=False)      # аналог summary в keras
----------------------------------------------------------------
            Layer (type)               Output Shape         Param #
================================================================
                Linear-1                      [-1, 5]              15
                Linear-2                      [-1, 1]               6
----------------------------------------------------------------
Всього параметрів: 21
Тренувані параметри: 21
Параметри, які не підлягають навчанню: 0
```

Якщо встановити show_input=True, замість форми виходу буде друкуватись форма входу кожного шару. Для складених моделей можна запускати функцію summary з параметром show_hierarchical=True.

Вивести параметри моделі при стопковому (Sequential) способі її завдання можна пошарово:

```
for layer in model:
    print("***", layer)
    for param in layer.parameters():
        print(param.data.numpy())
```

Загалом параметри моделі виводяться таким чином:

```
for param in model.parameters():
    print(param.numel(), param.size(), param.data.numpy())
```

Доступ параметрів через атрибут data необхідний, тому що вони є вузлами графа (можна написати param.detach()). Ще один спосіб виведення навчальних параметрів зі своїми іменами, числом параметрів і формами:

```
tot = 0
for k, v in model.state_dict().items():
    pars = np.prod(list(v.shape)); tot += pars
    print(f'{k:20s} :{pars:7d}  shape: {tuple(v.shape)} ')
print(f"{'total':20s} :{tot:7d}")
fc1.weight : 10 форма: (5, 2)
fc1.bias : 5 форма: (5,)
fc2.weight : 5 форма: (1, 5)
fc2.bias : 1 форма: (1,)
всього: 21
```

Нарешті, бібліотека torchviz дозволяє зобразити обчислювальний граф моделі:

```
import torchviz

torchviz.make_dot(model(X),
                  params = dict(model.named_parameters()) )
```

На рис. 3.3 показано приклад вивеведення обчислювального графу моделі нейронної мережі.

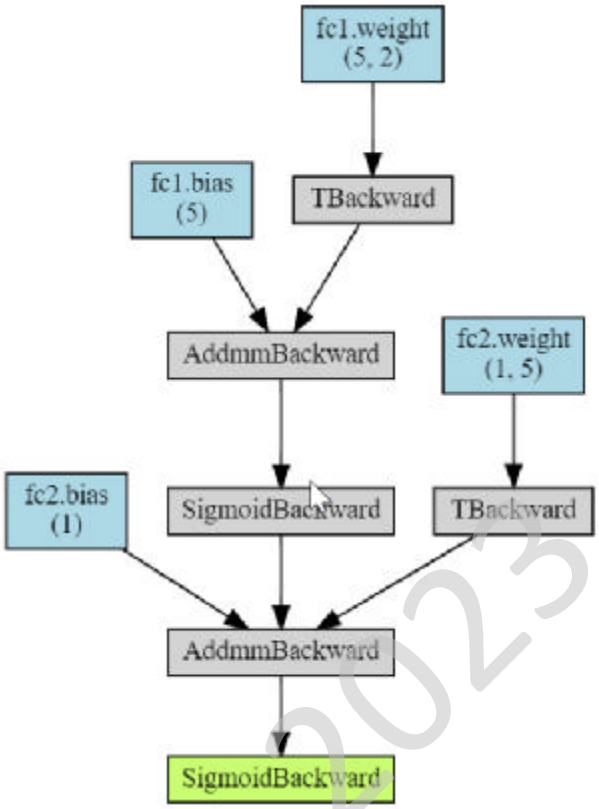


Рисунок 3.3 – Відображення структури нейронної мережі

У першому аргументі функції make_dot передається корінь графа моделі, який виходить при подачі до неї вхідного тензора (правильної форми, з будь-яким числом прикладів).

Вузол AddmmBackward відповідає функції torch.addmm(v, m1, m2, b=1, a=1), яка перемножує матриці та додає до них вектор: bv + a (m1 @ m2).

Після навчання нейронної мережі навчену модель треба зберігти для того, щоб можна її було використовувати в подальшому.

Метод torch.save зберігає у бінарному вигляді будь-який словник, у тому числі стану моделі та оптимізатора:

```
import datetime
state = {'info':      "Це моя мережа",            # опис
         'date':      datetime.datetime.now(),    # дата та час
```

```
            'model' :    model.state_dict(),        # параметри моделі
            'optimizer': optimizer.state_dict()}    # стан оптимізатора

    torch.save(state, 'state.pt')                   # зберігаємо файл
```

Потім їх можна завантажити, створивши попередньо модель та оптимізатор (його парметри можуть бути будь-якими, тому що вони завантажаться з файлу):

```
    state = torch.load('state.pt')                  # завантажуємо файл

    m = TwoLayersNet(2, 5, 1)                       # екземпляр мережі
    optimizer = torch.optim.SGD(m.parameters(),lr=1) # оптимізатор (будь-які
параметри)

    m.        load_state_dict(state['model'])       # отримуємо параметри
моделі
    optimizer.load_state_dict(state['optimizer'])   # отримуємо стан
оптимізатора

    print(state['info'], state['date'])             # допоміжна інформація
```

У torch прийнято обертати навчальні дані до класу. Він є спадкоємцем класу Dataset і перевизначає метод __len__ числа прикладів та метод __getitem__ отримання прикладу за індексом idx:

```
    class MyDataset(torch.utils.data.Dataset):      # Спадкоємець класу

        def __init__(self, N = 10, *args, **kwargs):
            super().__init__(*args,**kwargs)

            self.x = torch.rand(N,1)                # випадкові дані

        def __len__(self):                          # Кількість прикладів
            return len(self.x)

        def __getitem__(self, idx):                 # отримати idx-тий приклад
            return {'input' : self.x[idx],
                    'target': 2*self.x[idx]}
```

```
data = MyDataset()

for sample in data:
    print(sample)      # {'input': tensor([0.1545]), 'target':
tensor([0.3090])} ...
```

Створивши датасет, його можна передати об'єкту DataLoader, який видаватиме батчі, перемішуватиме, розбиватиме на навчання та тест, нормалізуватиме дані тощо.

```
train_loader = torch.utils.data.DataLoader(dataset=data, batch_size=12,
shuffle=False)

for batch in train_loader:          # отримуємо батчі для тренування
    print(batch['input'], batch['target'])
```

Для обчислення на GPU (як і в загальному випадку) необхідно створити відповідні обчислювальні пристрої:

```
gpu = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
cpu = torch.device("cpu")
```

Після створення моделі її можна відправити на графічну картку і (якщо дозволяє її пам'ять) теж зробити з навчальними даними:

```
model = TwoLayersNet(2, 5, 1)     # екземпляр мережі
model.to(gpu)                     # відправляємо його на GPU

X =  X.to(gpu)                    # відправляємо навчальні дані на GPU
Y =  Y.to(gpu)
```

Якщо для навчальних даних місця на GPU не вистачає, їх можна відправляти туди батчами.

При навчанні помилка пересилається з GPU в CPU. Зауважимо, що для маленьких моделей та навчальних масивів даних використання GPU не доцільно і може навіть уповільнити навчання порівняно з CPU.

Шари, як і вся мережа, є спадкоємцем класу nn.Module. Приклад реалізації лінійного шару:

```
import math
from torch.nn.parameter import Parameter

class My_Linear(nn.Module):

    def __init__(self, in_F, out_F):
        super(My_Linear, self).__init__()
        self.weight = Parameter(torch.Tensor(out_F, in_F))
        self.bias   = Parameter(torch.Tensor(out_F))

        self.reset_parameters()

    def reset_parameters(self):
        for p in self.parameters():
            stdv =  1.0 / math.sqrt(p.shape[0])
            p.data.uniform_(-stdv, stdv)

    def forward(self, x):
        return  x @ self.weight.t() + self.bias
```

Нижче наведено код простого SGD-оптимізатора. Звернімо увагу на декоратор @torch.no_grad() перед методом step. Він означає, що етап зміни параметрів виконуватиметься при вимкненому режимі побудови графа.

```
class SGD(torch.optim.Optimizer):

    def __init__(self, params, lr=0.1, momentum=0):
        defaults = dict(lr=lr, momentum=momentum)
        super(SGD, self).__init__(params, defaults)

    @torch.no_grad()
    def step(self):
        for group in self.param_groups:
            momentum = group['momentum']
            lr       = group['lr']
```

```
                    for p in group['params']:
                        if p.grad is None:
                            continue

                        grad = p.grad

                        if momentum != 0:
                            p_state = self.state[p]
                            if 'momentum_buf' not in p_state:
                                buf = p_state['momentum_buf'] =  torch.clone(grad)
                            else:
                                buf = p_state['momentum_buf']
                                buf.mul_(momentum).add_( grad
)
                            grad = buf

                        p.add_(grad, alpha = -lr)

        optimizer = SGD(model.parameters(), lr=0.1, momentum=0.9)
```

Параметри готових оптимізаторів можна також змінювати безпосередньо у процесі навчання:

```
        def adjust_optim(optimizer, epoch):
            if epoch == 1000:
                optimizer.param_groups[0]['betas'] = (0.3,
optimizer.param_groups[0]['betas'][1])
            if epoch > 1000:
                optimizer.param_groups[0]['lr'] *= 0.9999
```

Крім нейронної мережі необхідно створити середовище для її навчання.

Gym – інструментарій для розробки та порівняння алгоритмів навчання з підкріпленням з відкритим вихідним кодом. Робота з ним полегшується тим, що він дозволяє структурувати середовище за допомогою всього кількох рядків коду та сумісний з будь-якою бібліотекою обчислень, як то TensorFlow чи torch.

Бібліотека Gym – це колекція тестових завдань та середовищ, якими можна скористатися для навчання та розробки ефективніших моделей навчання з

підкріпленням. Представлені середовища мають загальний інтерфейс, що дозволяє писати узагальнені алгоритми. Gym надає широкий спектр середовищ симуляції.

При розробці інтелектуальної машини важливу роль відіграють Reinforcement Learning та середовище навчання машини. Середовище розробки, що застосовується для машинного навчання, не менш важливе, ніж застосовувані у вирішенні завдання прогнозного моделювання методи машинного навчання.

У задачі навчання з підкріпленням середовище становить основні та фундаментальні елементи, тому важливо розуміти базове середовище, з яким агент RL буде взаємодіяти. Розуміння середовища допомагає розробити правильний дизайн і методику навчання агента, що викладається.

Середовище – це світ агента, в якому він живе; агент взаємодіє із середовищем, виконуючи якісь дії, але, виконуючи ці дії, він не має права впливати на правила чи динаміку середовища, подібно до того, як люди – агенти в земному середовищі та обмежені її законами.

Середовище також дає агенту винагороду: скалярне значення, що повертається, яке діє як зворотний зв'язок, інформуючи агента про те, чи була його дія гарною або поганою.

У рамках навчання з підкріпленням багато парадигм досягають виграшної стратегії, тобто змушують агента виконувати бажану дію кількома способами. У складних ситуаціях обчислити точну виграшну стратегію або функцію винагороди складно, особливо коли агенти починають вчитися на основі взаємодії, а не раніше набутого досвіду.

Типи навчальних середовищ:

– Детерміноване середовище, де наступний стан середовища завжди визначається поточним станом та діями агента.

– Стохастичне середовище навчання з підкріпленням, де наступний стан завжди визначається поточним станом чи діями агента.

– Одноагентне середовище, де існує і взаємодіє із середовищем лише

один агент.

– Багатоагентне середовище, де є більше одного агента, що взаємодіє з середовищем.

– Дискретне середовище, простір дій якого має дискретний характер.

– Безперервне середовище, простір дій якого безперервний за своєю природою.

– Епізодичне середовище, де дії агента обмежені лише конкретним епізодом і не пов’язані з попередніми діями.

– Послідовне середовище, де дії агента пов’язані з попередніми діями.

### 3.2 Розробка структурної схеми

Структурна схема розробленого програмного забезпечення наведена на рис. 3.4.

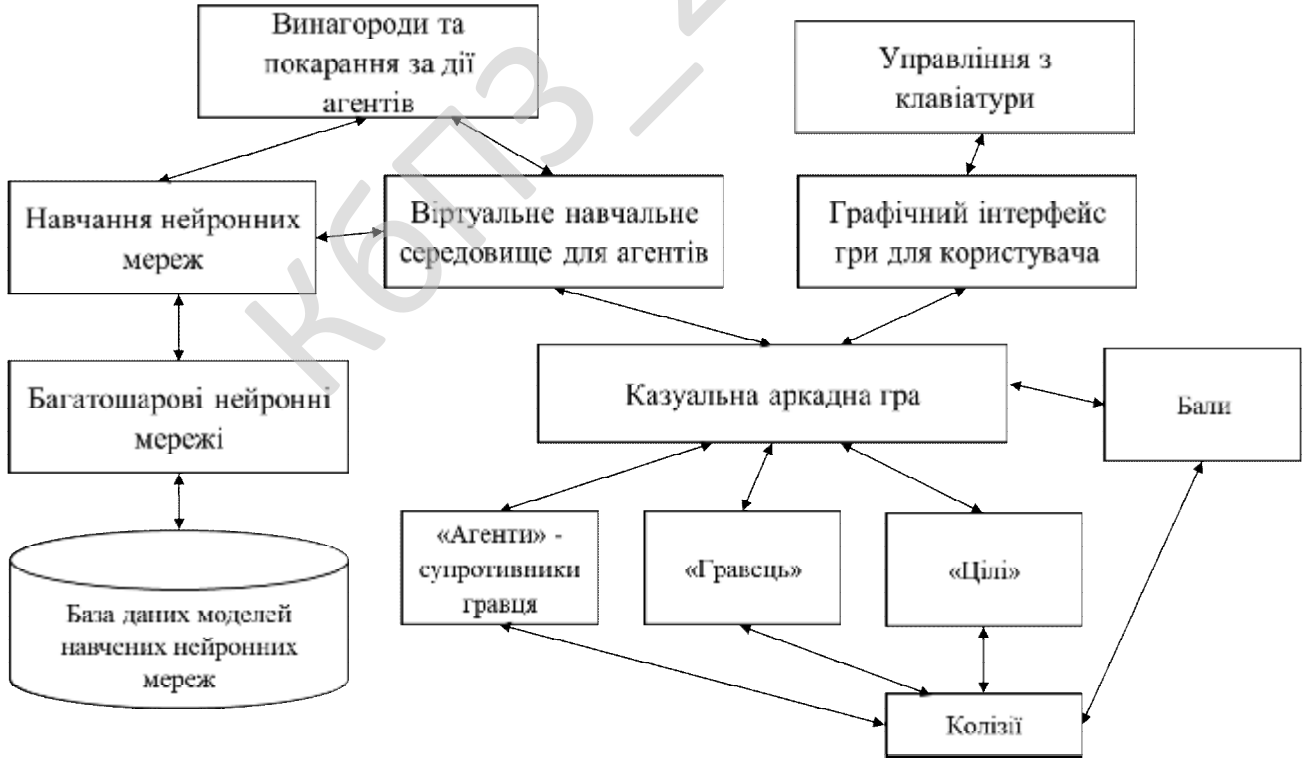


Рисунок 3.4 – Структурна схема системи

Як видно з рисунку, розроблена система складається з наступних елеменітв:

– Казуальна аркадна гра.

– Графічний інтерфейс гри для користувача.

– Управління з клавіатури.

– Навчання нейронних мереж.

– Багатошарові нейронні мережі.

– База даних моделей навчених нейронних мереж.

– «Агенти» – супротивники гравця.

– «Гравець».

– «Цілі».

– Колізії.

– Бали.

– Винагороди та покарання за дії агентів.

Система реалізована на мові високого рівня Python у середовищі розробки Anaconda та з застосуванням бібліотек pygame, gym, torch. Бібліотека pygame використана для створення казуальної аркадної комп'ютерної гри та інтерфейсу користувача. Бібліотека gym використана для створення віртуального навчального середовища для інтелектуальних агентів – супротивників гравця в розробленій казуальній аркадній грі. Бібліотека torch використана для реалізації нейронних мереж, які здійснюють управління агентами у комп'ютерній грі.



### 3.3 Розробка функціональної схеми

Функціональна схема розробленої системи представлена на рис. 3.5.

Як видно з рисунку, система складається з наступних модулів:

– Модуль реалізації нейронних мереж.

– Модуль комп'ютерної гри.

– Модуль обробки даних.


Рисунок 3.5 – Функціональна схема системи

Модуль реалізації нейронних мереж забезпечує насупний функціонал:

– Створення архітектури нейронної мережі.

– Ініціалізація нейронної мережі.

– Одержання нейронною мережею вхідних даних.

– Прийняття рішення нейронною мережею по управлінню «агентом» у комп'ютерній грі.

– Одержання нейронною мережею винагород та покарань за обрані дії.

– Навчання нейронної мережі на основі нагород та покарань.

Модуль комп'ютерної гри забезпечує насупний функціонал:

– Ініціалізація елементів гри.

– Відображення ігрового поля.

– Відображення «гравця».

– Відображення «агентів».

– Відображення «цілей».

– Управління «гравцем» з клавіатури.

– Управління «агентами» на основі нейронних мереж.

– Відслідковування досягнення цілей та нарахування балів.

– Відображення балів.

Модуль обробки даних забезпечує насупний функціонал:

– Збереження таблиці рейтингів.

– Збереження вагових коефіцієнтів нейронних мереж.

– Побудова та відображення графіків винагород та покарань нейронних мереж за прийняття рішень в ході гри.

– Побудова та відображення графіків величини помилок нейронних мереж в ході гри.

– База даних таблиць рейтингів.

– База даних моделей навчених нейронних мереж.

### 3.4 Розробка діаграми процесів

На рисунку 3.6 зображена діаграма процесів, що описує наявні у системі процеси та їх взаємодію.

У розробленій системі присутні наступні процеси:

- Головне вікно програми.
- Аркадна казуальна гра.
- Відображення «гравця».
- Управління «гравцем» з клавіатури.
- Відображення «агентів».
- Управління «агентами» на основі нейронних мереж.
- Відображення «цілей».
- Віртуальне середовище для навчання агентів.
- Нарахування і відображення балів за досягнення цілей.
- Нейронні мережі.

- Створення нейронної мережі.
- Навчання нейронної мережі.
- Збереження вагових коефіцієнтів нейронних мереж.
- Графіки роботи нейронних мереж.
- Графіки винагород та покарань.
- Графіки величин помилок.
- Таблиця рейтингів.
- Збереження таблиці рейтингів.

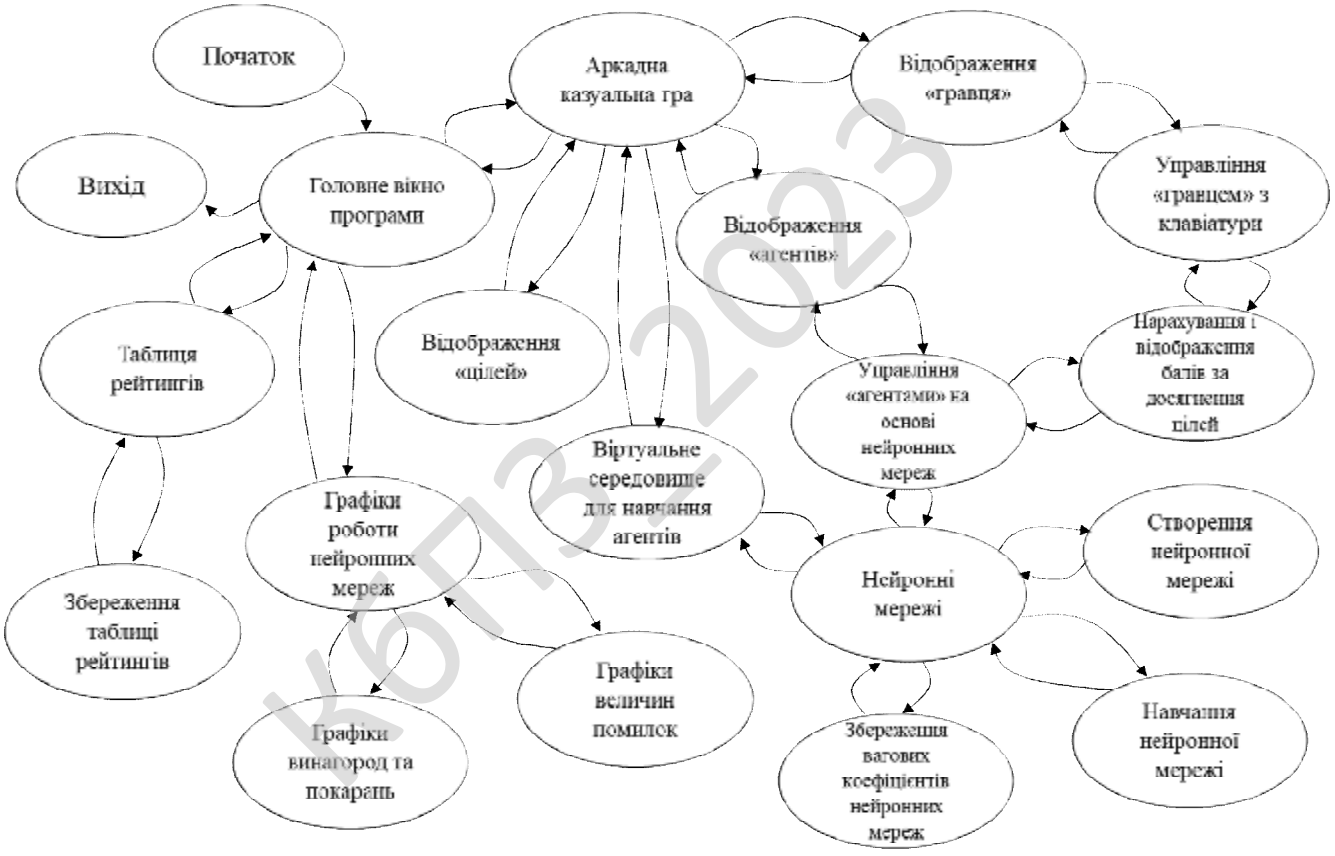


Рисунок 3.6 – Діаграма процесів системи

Розроблена система включає декілька рівнів взаємодії, інтерактивний користувацький інтерфейс, аналітику та моніторинг роботи нейронних мереж та інструменти для заберігання даних.

# 4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

## 4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведена блок-схема основної програми.

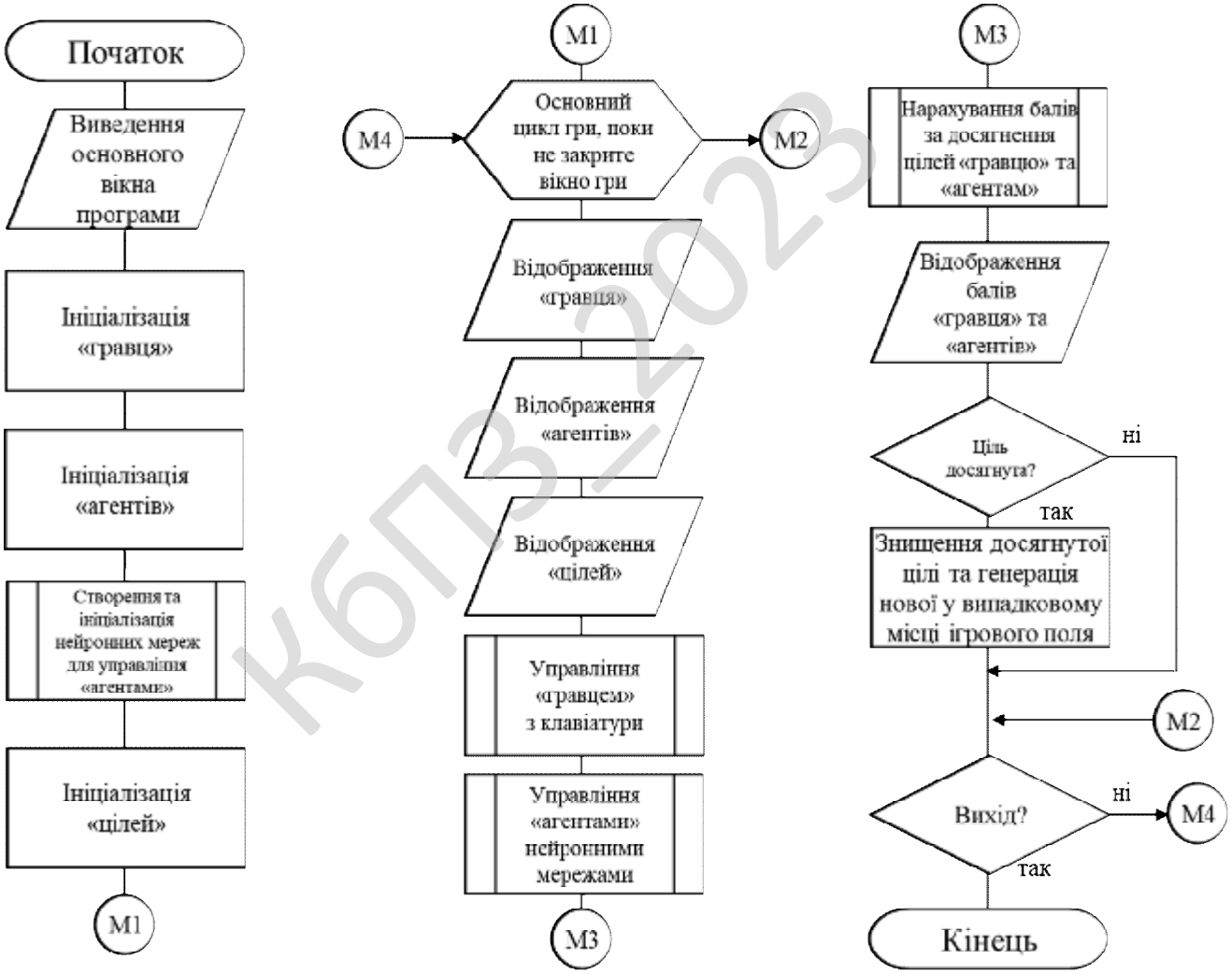


Рисунок 4.1 – Блок-схема основної програми

Агоритм роботи основної програми містить наступні кроки:

**Крок 1.** Виведення основного вікна програми.

**Крок 2.** Ініціалізація «гравця».

**Крок 3.** Ініціалізація «агентів».

**Крок 4.** Запуск підпрограми створення та ініціалізації нейронних мереж для управління «агентами».

**Крок 5.** Ініціалізація «цілей».

**Крок 6.** Основний цикл гри, який відбувається поки користувач не закриє вікно. В цьому циклі безперервно виконуються всі наступні кроки.

**Крок 7.** Відображення «гравця».

**Крок 8.** Відображення «агентів».

**Крок 9.** Відображення «цілей».

**Крок 10.** Управління «гравцем» з клавіатури.

**Крок 11.** Управління «агентами» нейронними мережами.

**Крок 12.** Запуск підпрограми відслідковування зіткнень з «цілями» та нарахування балів за досягнення цілей «гравцю» та «агентам».

**Крок 13.** Відображення балів «гравця» та «агентів».

**Крок 14.** Перевірка чи ціль досягнута кимось. Якщо ціль досягнута – виконання кроку 15, в іншому випадку – кроку 16.

**Крок 15.** Знищення досягнутої цілі та генерація нової у випадковому місці ігрового поля.

**Крок 16.** Перевірка чи натиснув користувач вихід з гри. Якщо так – зупиняємо гру та виводимо таблицю рейтингів та графіки контролю роботи нейронних мереж. Якщо ні – переходимо на наступну ітерацію циклу з кроку 6.

На рисунку 4.2 наведена блок-схема алгоритму управління ігровим об’єктом на основі нейронної мережі.

Ця блок-схема відображає спосіб взаємодії нейронної мережі та правила нарахування балів за вірні та невірні дії нейронної мережі.

Нейронна мережа на кожному кроці основного циклу гри отримує поточні координати «агента», яким управляє та координати найближчої цілі. Вона повинна визначити, у якому напряпку повинен здійснити рух агент: вниз, вверх, вліво, вправо, вниз-вліво, вниз-вправо, вгору-вліво, вгору-вправо.

Як видно зі схеми, бали нейронній мережі нараховуються наступним чином:

– За наближеня до найближчої цілі +1 бал.

– За віддалення від найближчої цілі –1 бал.

– За досягнення цілі +2 бали.

– За вихід за межі ігрового поля –2 бали.

Кожного разу після вибору та здійснення дії і нарахування балів відбувається запуск підпрограми навчання нейронної мережі. Таким чином чим довше триває гра, тим краще починає грати нейронна мережа.

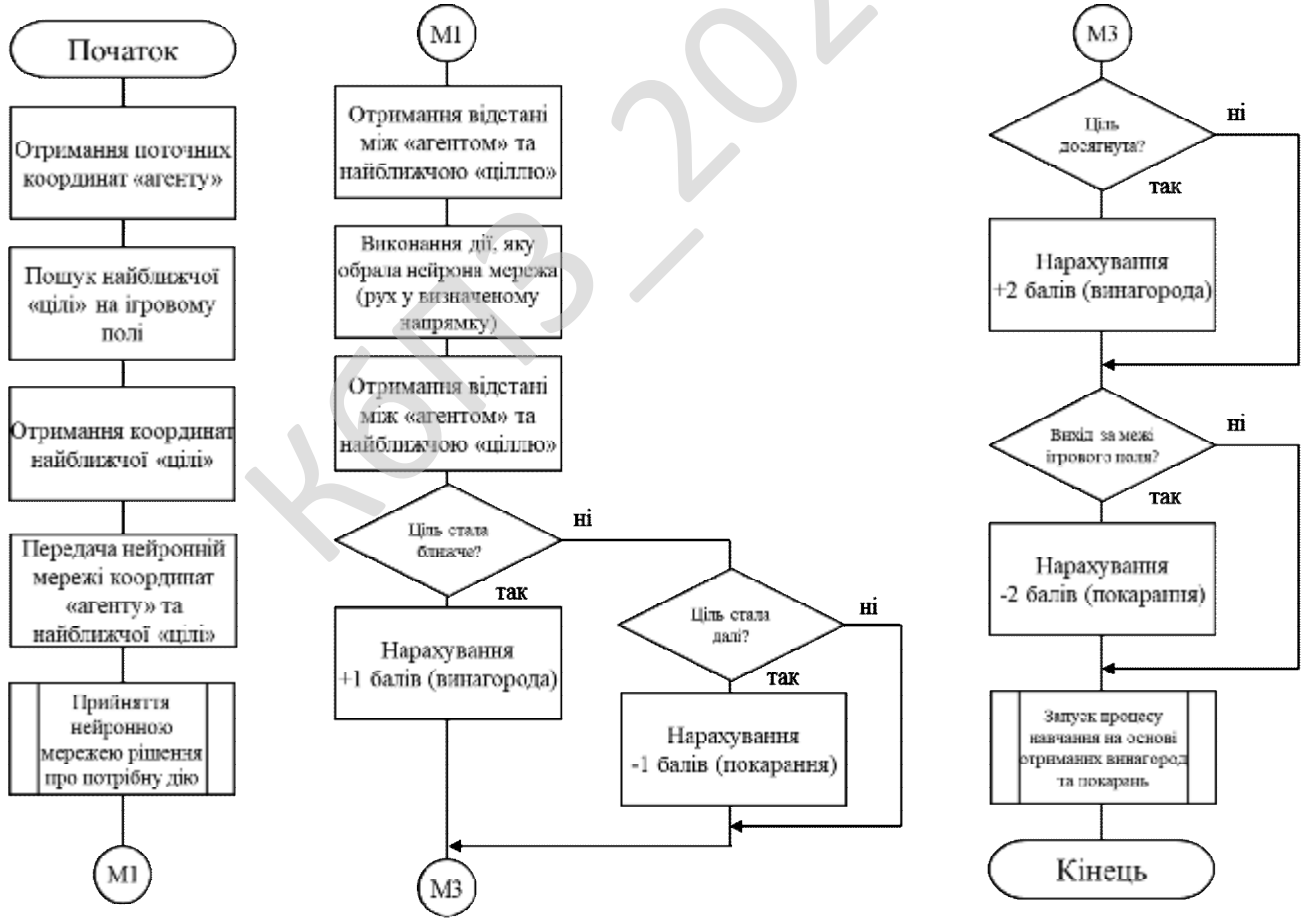


Рисунок 4.2 – Блок-схема алгоритму управління ігровим об’єктом на основі нейронної мережі

Клас для роботи з нейронною мережею реалізовано наступним чином:

```
# Клас для роботи нейронної мережі, яка управляє агентами - супротивниками у грі
class NeuralNetwork(nn.Module):

    def __init__(self, observation_space, action_space):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(observation_space, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, action_space)

    def forward(self, x):
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x

    def select_action(state, network):
        with torch.no_grad():
            state = torch.from_numpy(state).float().unsqueeze(0)  # Додаємо додатковий вимір для пакету
            action_probs = network(state)
            action = action_probs.max(1)[1].view(1, 1)
            print("Action chosen:", action)  # Для тестування
        return action

    def compute_loss(state, next_state, action, reward, done, neural_network_index):

        # Перевірка, чи всі необхідні дані присутні
        if state is None or next_state is None or action is None or reward is None:
            return None


        # Перетворення в тензори
        state = torch.from_numpy(state).float()
        next_state = torch.from_numpy(next_state).float()
        reward = torch.tensor(reward).float()
        action = torch.tensor(action).long()
```

```
        # Отримання прогнозованих значень від нейронної мережі
        predicted_values = neural_networks[neural_network_index](state)

        # Вибір значення для зробленої дії
        action = action.view(1) # Зміна форми action до [batch_size, 1]
        predicted_value = predicted_values.gather(-1, action).squeeze(0)

        # Розрахунок цільового значення
        target_value = reward
        if not done:
            target_value = reward + 0.99 *
neural_networks[neural_network_index](next_state).max(0)[0]

        # Обчислення втрат
        loss = F.mse_loss(predicted_value, target_value.detach())

        return loss
```

Клас для реалізації віртуального навчального середовища для нейронної мережі реалізовано наступним чином:

```
# клас для створення ігрового середовища для нейронної мережі, у якому вона
буде діяти, оримувати винагороди та покарання і навчатися

class SimpleGameEnv(gym.Env):

    def __init__(self):
        super(SimpleGameEnv, self).__init__()

        # Створюємо простір можливих дій нейронної мережі, вона може
пересувати ігрового агента в різні сторони
        # Дії: 0 - вліво, 1 - вправо, 2 - вгору, 3 - вниз
        #      4 - вгору-вліво, 5 - вгору-вправо, 6 - вниз-вліво, 7 - вниз-
вправо
        self.action_space = spaces.Discrete(8)


        # Спостереження: позиція агента та цілі
```

```
        self.observation_space = spaces.Box(low=0, high=max(screen_width,
screen_height), shape=(4,), dtype=np.float32)

        # Ініціалізація стану
        self.state = None
        self.next_state = None

        self.agents = [Agent(random.randint(100, 700), random.randint(100,
500), (255, 0, 255), f"Агент {i}") for i in range(3)]

        # Ініціалізація цілей – кульок, які треба збирати
        self.targets = [Target() for _ in range(10)]

        # Ініціалізація гравця, яким буде управляти людина
        self.player = Player(100, 100, (0, 0, 255), "Гравець")

        self.reset()

    # метод обчислення дистанції між агентом та ціллю

    def calculate_distance(self, agent, target):
        return np.linalg.norm([agent.x - target.x, agent.y - target.y])




    # метод для виконання агентом одного кроку та нарахування йому балів та
винагород за цей крок

    def step(self, action, agent_index, nearest_target, rnd = False):

        agent = self.agents[agent_index]

        d1 = self.calculate_distance(agent, nearest_target)

        if rnd:
            step_size = random.randint(5, 100)
        else:
            step_size = 1

        # Оновлення позиції агента
```

```
if action == 0: # вгору
    agent.x = agent.x - step_size
elif action == 1: # вниз
    agent.x = agent.x + step_size
elif action == 2: # вліво
    agent.y = agent.y - step_size
elif action == 3: # вправо
    agent.y = agent.y + step_size

# Додавання діагональних напрямків

elif action == 4:  # вгору-вліво
    agent.x -= step_size
    agent.y -= step_size
elif action == 5:  # вгору-вправо
    agent.x += step_size
    agent.y -= step_size
elif action == 6:  # вниз-вліво
    agent.x -= step_size
    agent.y += step_size
elif action == 7:  # вниз-вправо
    agent.x += step_size
    agent.y += step_size



done = False
reward = -0.01

d2 = self.calculate_distance(agent, nearest_target)

if d2 < d1:
    reward += 2

elif d2 == d1:
    reward -= 0

elif d2 > d1:
    reward -= 2
```

```
        if agent.x < 0 or agent.x > screen_width or agent.y < 0 or agent.y >
screen_height:
            reward -= 1.0  # Мінусувати бали за вихід за межі екрану
            agent.x, agent.y = random.randint(100, 700), random.randint(100,
500)

        for target in self.targets[:]:
            if self.check_collision(agent, target):
                if target == nearest_target:
                    agent.score += target.radius
                    reward += target.radius
                else:
                    agent.score += 3
                    reward += 3

                # Перевірка, чи ціль ще є в списку перед її видаленням
                if target in self.targets:
                    self.targets.remove(target)
                    self.targets.append(self.create_new_target())

                reward = 1.0
                done = True

                break

        # Пошук найближчої за відстанню цілі
        nearest_target = self.find_nearest_target(agent, env.targets)

        # Оновлення поточного стану середовища для агента
        self.state = np.array([agent.x, agent.y, nearest_target.x,
nearest_target.y])

        return self.state, reward, done, {}


    # скидання стану ігрового середовища
    def reset(self):

        # Випадкове розміщення кожного агента
        for agent in self.agents:
            agent.x = random.randint(100, 700)
            agent.y = random.randint(100, 500)
```

```
        # Випадкове розміщення кожної цілі
        for target in self.targets:
            target.x = random.randint(100, 700)
            target.y = random.randint(100, 500)
            target.color = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))


        self.state = None
        return np.array(self.state)

    # рендер ігрового середовища
    def render(self, mode='human'):
        screen.fill((255, 255, 255))
        for agent in self.agents:
            agent.draw()
        for target in self.targets:
            target.draw()

        self.player.draw()

        pygame.display.flip()

        # Відображення балів
        font = pygame.font.SysFont(None, 24)
        score_text = font.render(f"Гравець: {self.player.score}", True, (0, 0,
0))
        screen.blit(score_text, (screen_width - 150, 10))
        for i, agent in enumerate(self.agents):
            agent_score_text = font.render(f"{agent.name}: {agent.score}",
True, (0, 0, 0))
            screen.blit(agent_score_text, (screen_width - 150, 40 + i * 30))


        pygame.display.flip()



    # метод визначення колізій між агентом (або гравцем) і ціллю
    def check_collision(self, entity, target):
```

```
        # Розрахунок центру агента (гравця), якщо агент - квадрат
        entity_center_x = entity.x + entity.size / 2
        entity_center_y = entity.y + entity.size / 2

        # Відстань між центрами агента (гравця) і цілі
        distance = np.linalg.norm([entity_center_x - target.x, entity_center_y
- target.y])

        # Перевірка зіткнення: чи відстань менша або дорівнює сумі радіусів
        return distance <= (entity.size / 2 + target.radius)

    # метод створення нової цілі
    def create_new_target(self):
        return Target()

    # метод знаходження найближчої цілі у просторі
    def find_nearest_target(self, agent, targets):
        nearest_target = None
        min_distance = float('inf')
        for target in targets:
            distance = np.linalg.norm([agent.x - target.x, agent.y -
target.y])
            if distance < min_distance:
                min_distance = distance
                nearest_target = target
        print(agent.name, nearest_target.x, nearest_target.y)
        return nearest_target

    # метод закриття вікна гри
    def close(self):
        # Закриття вікна гри
        pygame.quit()
```

Далі розглянемо яким чином реалізовані ігрові елементи.

Клас для реалізації «гравця» має наступний вигляд:

```
# клас «гравець»
class Player:
```

```
    # конструктор класу
    def __init__(self, x, y, color, name):

        self.x = x
        self.y = y
        self.size = 20
        self.color = color
        self.name = name
        self.score = 0

    # метод зміни розміру гравця при збільшенні кількості набраних балів
    def update_size(self, n):

        # Оновлення розміру на основі балів
        self.size = n

    # метод малювання гравця на екрані
    def draw(self):

        pygame.draw.rect(screen, self.color, (self.x, self.y, self.size,
self.size))
        font = pygame.font.SysFont(None, 24)
        name_text = font.render(self.name, True, (0, 0, 0))
        screen.blit(name_text, (self.x, self.y - 20))


    # метод керування гравцем з клавіатури
    def handle_player_movement(player):

        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            player.x -= 5
        if keys[pygame.K_RIGHT]:
            player.x += 5
        if keys[pygame.K_UP]:
            player.y -= 5
        if keys[pygame.K_DOWN]:
            player.y += 5
```

Клас для реалізації «агентів» має наступний вигляд:

```
# клас «агент»
class Agent:

    # конструктор класу
    def __init__(self, x, y, color, name):

        self.x = x
        self.y = y
        self.size = 20
        self.color = color
        self.name = name
        self.score = 0

    # метод малювання агента на екрані
    def draw(self):

        pygame.draw.rect(screen, self.color, (self.x, self.y, self.size,
self.size))
        font = pygame.font.SysFont(None, 24)
        name_text = font.render(self.name, True, (0, 0, 0))
        screen.blit(name_text, (self.x, self.y - 20))

    # метод зміни розміру агента при збільшенні кількості набраних балів
    def update_size(self, n):

        # Оновлення розміру на основі балів
        self.size = n
```

Клас для реалізації «цілей» має наступний вигляд:

```
# клас «ціль»
class Target:

    # конструктор класу
    def __init__(self):

        k = 100
        self.x = random.randint(0+k, screen_width-k)
        self.y = random.randint(0+k, screen_height-k)
        self.radius = random.randint(5, 30)
```

```
            self.color = (random.randint(0, 255), random.randint(0, 255),
        random.randint(0, 255))

            # метод малювання цілі на екрані
            def draw(self):

                pygame.draw.circle(screen,     self.color,     (self.x,     self.y),
    self.radius)
```

### 4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення пропонується використовувати криптографічний алгоритм RSA. За допомогою нього можна здійснювати шифрування файлів бази даних програмного забезпечення, а також здійснювати захищену авторизацію.

Розглянемо приклад шифрування файлів бази даних для захищеного їх зберігання та обмеження до них доступу.

Шифрування файлів за допомогою алгоритму RSA на Python вимагає декількох кроків, включаючи генерацію ключів RSA, шифрування файлу за допомогою публічного ключа та дешифрування за допомогою приватного ключа. Ось прикладний код, який демонструє ці кроки:

**1. Встановлення бібліотеки для шифрування.** Перш за все, потрібно встановити бібліотеку cryptography, яка надає інструменти для роботи з RSA:

```
pip install cryptography
```

**2. Генерація ключів RSA:**

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization

# Генерація приватного ключа
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
```

```
)

# Збереження приватного ключа
with open("private_key.pem", "wb") as f:
    f.write(
        private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        )
    )

# Генерація публічного ключа
public_key = private_key.public_key()

# Збереження публічного ключа
with open("public_key.pem", "wb") as f:
    f.write(
        public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
    )
```

### 3. Шифрування файлу:

```
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes

def encrypt_file(file_path, public_key_path):
    # Відкриття публічного ключа
    with open(public_key_path, "rb") as key_file:
        public_key = serialization.load_pem_public_key(
            key_file.read(),
            backend=default_backend()
        )

    # Читання даних файлу
    with open(file_path, "rb") as f:
        data = f.read()

    # Шифрування даних
```

```
    encrypted = public_key.encrypt(
        data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )


    # Збереження зашифрованих даних
    with open(file_path + ".encrypted", "wb") as f:
        f.write(encrypted)

encrypt_file("path/to/your/file", "public_key.pem")
```

**4. Дешифрування файлу:**

```
def decrypt_file(encrypted_file_path, private_key_path):
    # Відкриття приватного ключа
    with open(private_key_path, "rb") as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=None,
            backend=default_backend()
        )

    # Читання зашифрованих даних
    with open(encrypted_file_path, "rb") as f:
        encrypted_data = f.read()

    # Дешифрування даних
    original_data = private_key.decrypt(
        encrypted_data,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )

    # Збереження розшифрованих даних
```

```
        with open(encrypted_file_path + ".decrypted", "wb") as f:
            f.write(original_data)
    decrypt_file("path/to/file.encrypted", "private_key.pem")
```

Ці фрагменти коду демонструють базовий процес створення ключів RSA, шифрування та дешифрування файлів у Python. Важливо пам’ятати, що безпека шифрування залежить від безпечного зберігання та обробки ключів шифрування.

# 5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для запуску розробленого программного забезпечення треба попередньо встановити бібліотеки для мови програмування Python, а саме:

– **pygame** – використовується для створення відеоігор та мультимедійних програм на Python. Pygame надає функціонал для роботи з графікою, звуком, клавіатурою, мишею та іншими вхідними пристроями. Це дуже популярна бібліотека для простих та середніх за складністю проектів у галузі розваг та освіти.

– **gym –** інструментарій для розробки та порівняння алгоритмів навчання з підкріпленням. Розроблена компанією OpenAI. Надає широкий вибір середовищ (або "envs" у термінології Gym), у яких можна тренувати і тестувати інтелектуальних агентів.

– **torch** – використовується для машинного навчання, надає широкі можливості для створення нейронних мереж, глибокого навчання та наукових обчислень. PyTorch відома своєю гнучкістю, швидкістю та легкістю використання, особливо при прототипуванні та дослідженнях в області штучного інтелекту. Дозволяє легко виконувати чисельні операції на GPU, що значно прискорює обчислення при навчанні нейронних мереж.

Приклади роботи розробленого програмного забезпечення показаний на рис. 5.1-5.7.

Для тестування роботи запропонованого методу управління ігровими об’єктами у комп’ютерній грі була розроблена казуальна аркадна гра (рис. 5.1). Правила гри наступні: гравці повинні зібрати якомога більше кульок, за кожну зібрану кульку нараховуються бали, виграє той, хто набрав найбільше балів по завершенню гри. Кульки збирають один «гравець» та три «агенти». Людина управляє «гравцем», нейронні мережі управляють «агентами» у грі. Якщо хтось зібрав кульку, то у випадковому місці екрану з’являється нова, таким чином гру

можна продовжувати безкінечно. При цьому начання нейронних мереж продовжується весь час, тому чим довше продовжується гра – тим краще починають грати агенти.




Рисунок 5.1 – Розроблена казуальна аркадна комп'ютерна гра, у якій агентами-супротивниками керує багатошарова нейронна мережа

По завершенню гри на екран виводиться таблиця рейтигів, також вона записується у файл з назвою у форматі:

"ratings" + "%Y-%m-%d %H_%M_%S"+ ".txt"

де "%Y-%m-%d %H" – поточна дата, %H_%M_%S"+ " – поточний час.

Після завершення гри, виведення та збереження таблиці рейтигів, у розробленому програмному забезпеченні виводяться графіки, що ілюструють

процес навчання нейронних мереж під час гри. А саме виводяться графіки отримання винагород та покарань нейронних мереж під час гри за правильні та неправильні дії (рис. 5.2-5.4) та графіки величини помилок нейронних мереж прийняття рішень під час гри (рис. 5.5-5.7).

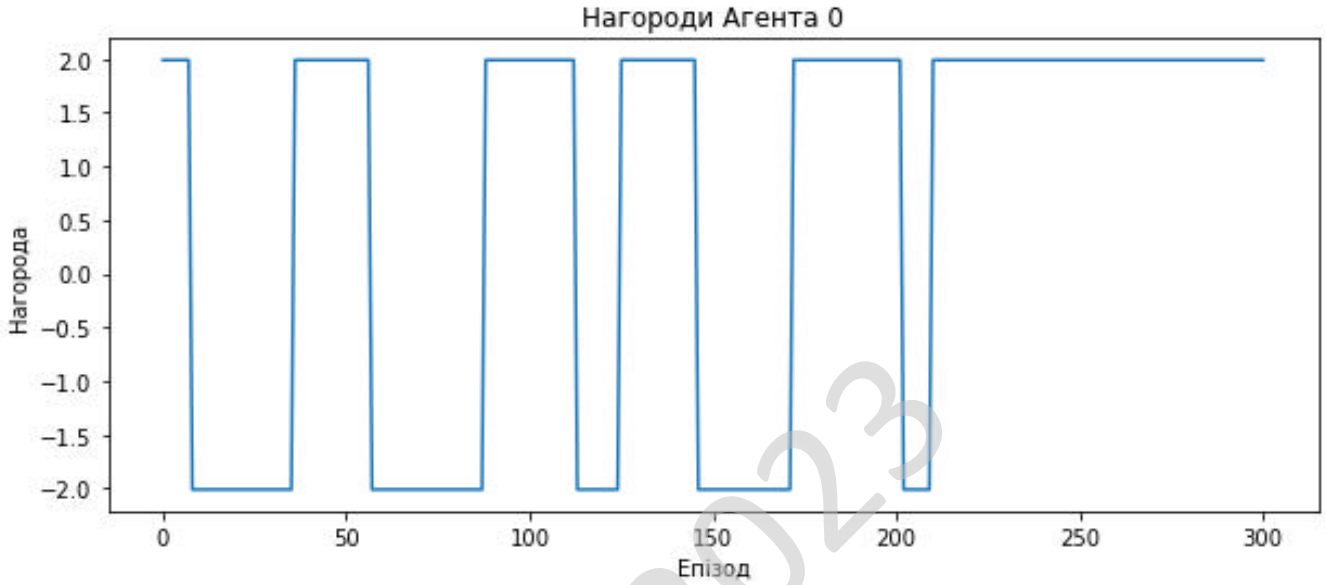


Рисунок 5.2 – Приклад №1: Графік отримання винагород та покарань нейронною мережею, що управляла «Агентом 0», за правильні та неправильні дії

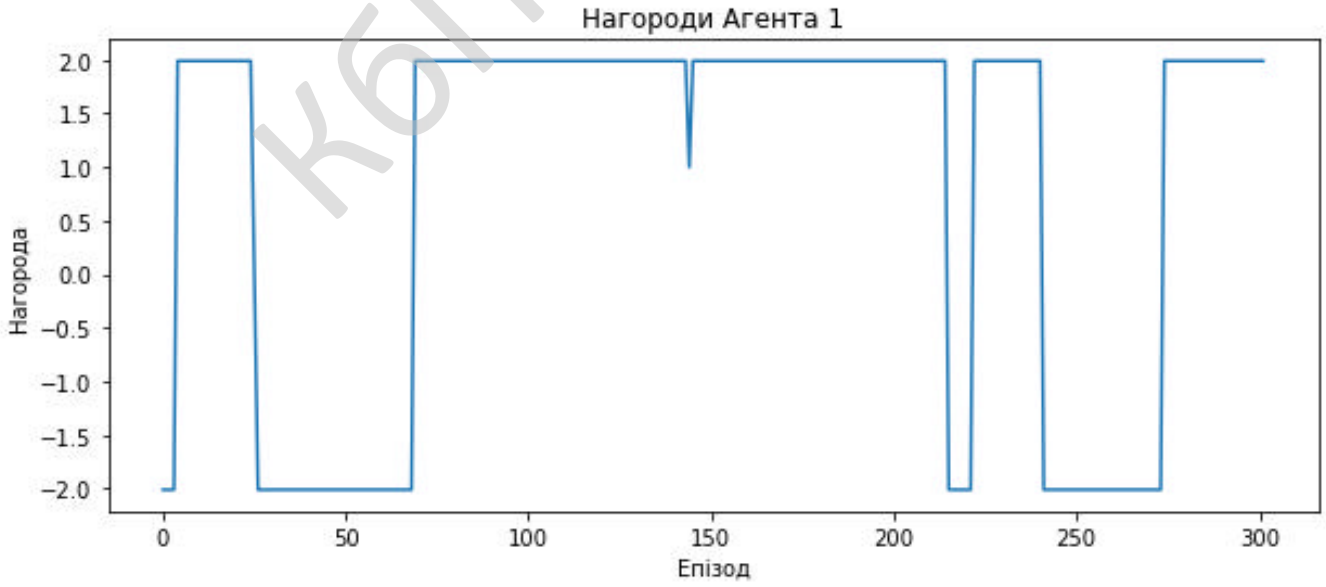


Рисунок 5.3 – Приклад №2: Графік отримання винагород та покарань нейронною мережею, що управляла «Агентом 1», за правильні та неправильні дії

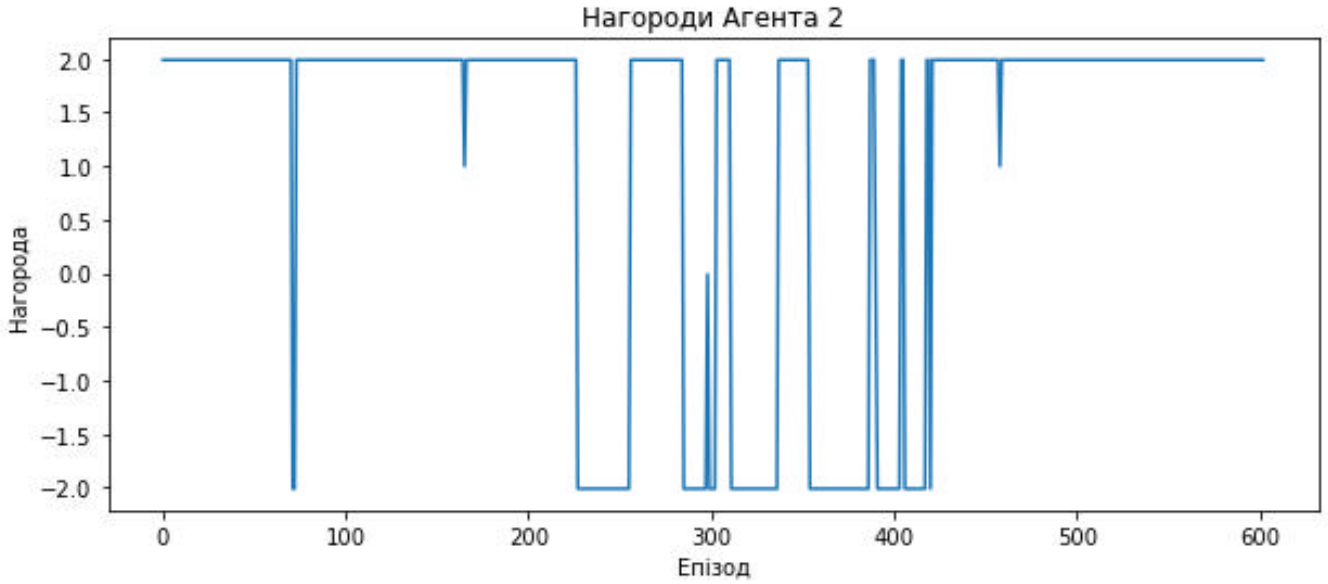


Рисунок 5.4 – Приклад №3: графік отримання винагород та покарань нейронною мережею, що управляла «Агентом 2», за правильні та неправильні дії

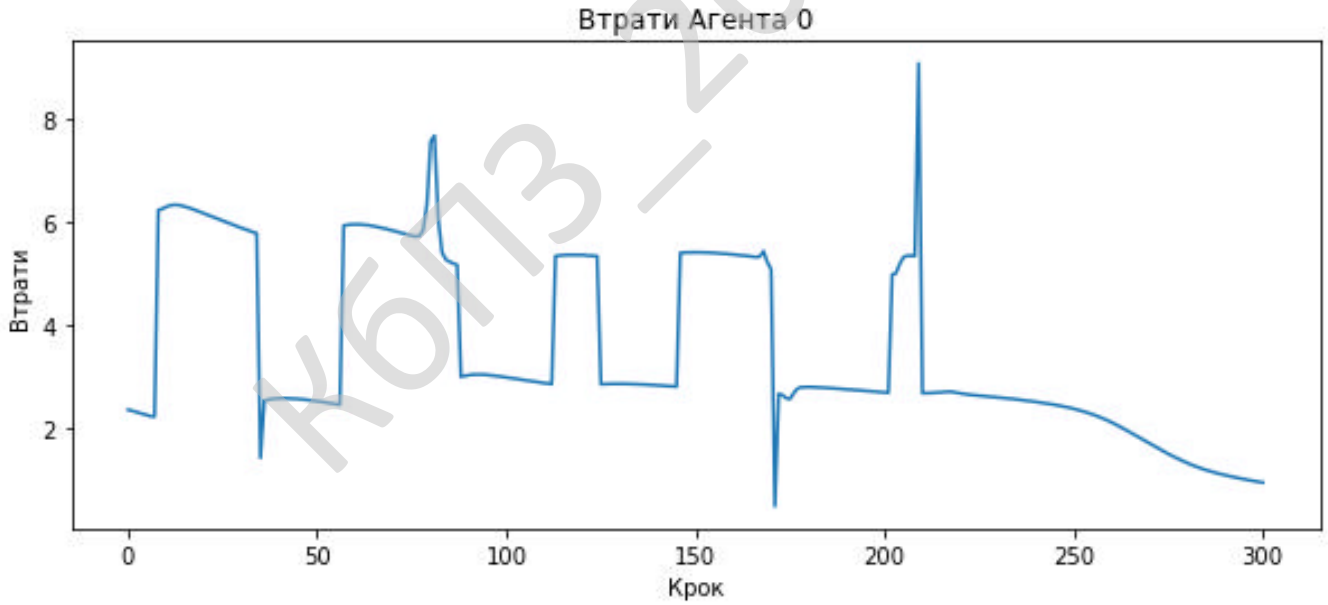


Рисунок 5.5 – Приклад №4: Графік величини помилок при прийнятті рішень нейронною мережею, що управляла «Агентом 0» під час гри

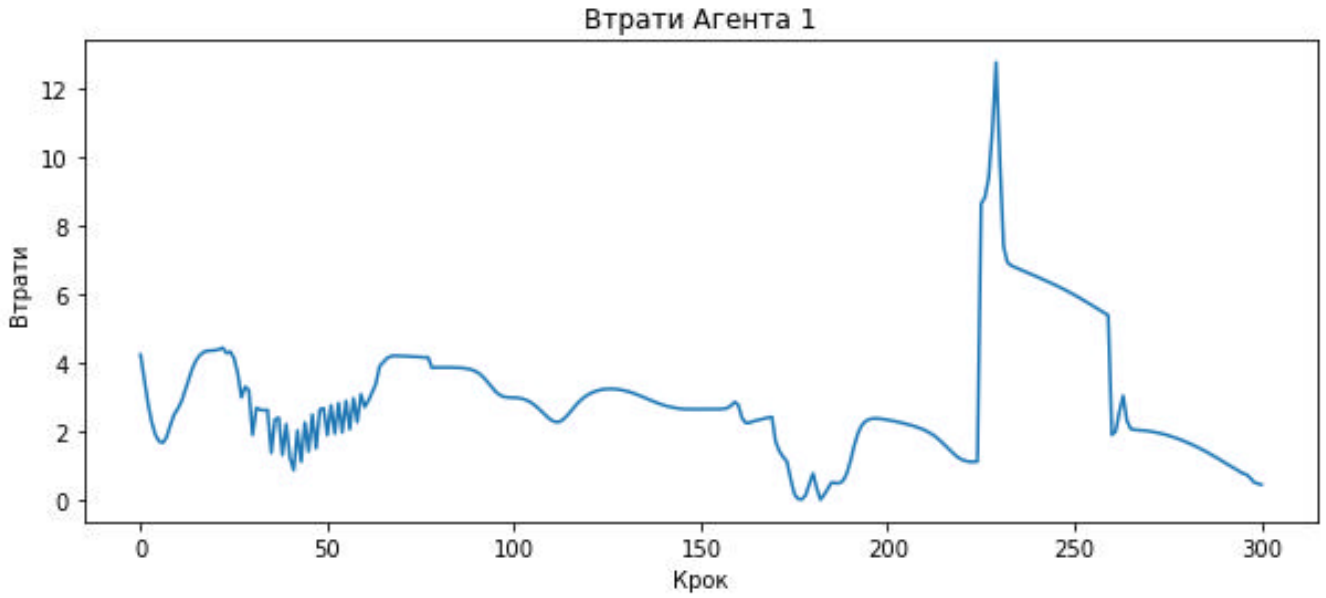


Рисунок 5.6 – Приклад №5: Графік величини помилок при прийнятті рішень нейронною мережею, що управляла «Агентом 1» під час гри

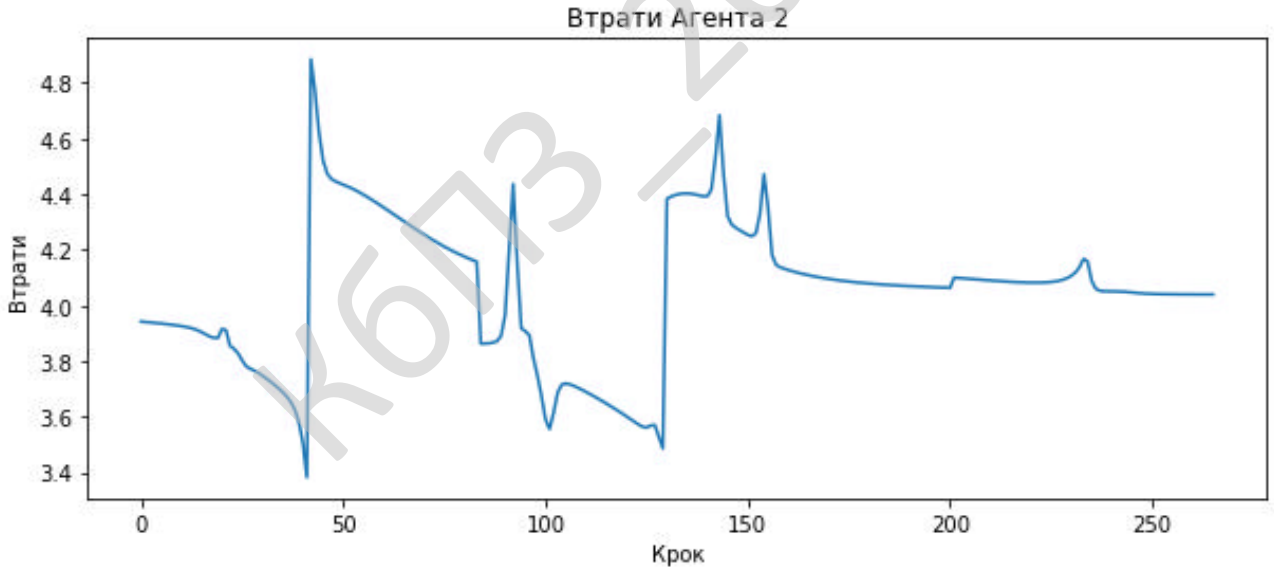


Рисунок 5.7 – Приклад №6: Графік величини помилок при прийнятті рішень нейронною мережею, що управляла «Агентом 2» під час гри

# 6 НАУКОВА НОВИЗНА

У магістерській роботі розроблено програмне забезпечення, яке призначено для інтелектуального управління агентами-супротивниками у комп'ютерній грі на основі багатошарових нейронних мереж.

*Об'єктом дослідження* є процес управління об'єктами у віртуальному ігровому середовищі.

*Предметом дослідження* є методи управління ігровими об'єктами за допомогою багатошарових нейронних мереж.

*Методи дослідження* базуються на теорії штучного інтелекту, теорії математичної статистики, методах розробки програмного забезпечення, методах об'єктно-орієнтованого програмування та методах розробки комп'ютерних ігор.

**Наукова новизна отриманих результатів.** У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

1. Запропоновано метод управління ігровими агентами, що є супротивниками гравця у казуальній аркадній грі, за допомогою багатошарових нейронних мереж та безперервного навчання у віртуальному ігровому середовищі.

2. Розроблено вітчизняний продукт для інтелектуального управління ігровими об'єктами за допомогою нейронних мереж, який має більш широкі можливості, на відміну від існуючих аналогів та може бути використаний при розробці комп'ютерних ігор.

**Практична цінність отриманих результатів** полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтелектуального управління агентами-супротивниками у комп'ютерній грі.

# 7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

## 7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведено дослідження та виконана програмна реалізація нейронних мереж для управління ігровими об’єктами.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп’ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

| Показники | | Позначення | Характеристика або величина |
|---|---|---|---|
| 1 | | 2 | 3 |
| 1. | Кількість розроблених програм період, шт. | N | 1 |
| 2. | Кількість екземплярів програм, шт. | Ne | 150 |
| 3. | Запланований термін розробки, днів | Fpq | 60 (3 місяці) |
| 4. | Група задачі підсистеми управління (1-6) | – | 1 |
| 5. | Ступінь новизни задачі (А, Б, В, Г) | – | Б |
| 6. | Складність алгоритму (1, 2, 3) | – | 2 |

Продовження таблиці 7.1

| 1 | 2 | 3 |
|---|---|---|
| 7. Кількість макетів вхідної інформації | – | 3 |
| 8. Кількість форм вихідної інформації. | – | 4 |
| 9. Мова програмування (1-6) | – | 1 |
| 10. Попередній досвід (1-6) | – | 3 |
| 11. Гнучкість проекту ПП (1-6) | – | 3 |
| 12. Детальність проекту ПП (1-6) | – | 2 |
| 13. Рівень спрацьованості колективу (1-6) | – | 2 |
| 14. Ступінь вимірності процесів (1-6) | – | 3 |
| 15. Необхідна надійність програмного забезпечення (1-6) | – | 2 |
| 16. Розмір бази даних (порівняно з розміром програми) (1-6) | – | 2 |
| 17. Складність кінцевого програмного продукту (1-6) | – | 2 |
| 18. Необхідний рівень забезпечення повторного використання (1-6) | – | 2 |
| 19. Документованість відповідно до планованого життєвого циклу (1-6) | – | 2 |
| 20. Вимоги до швидкодії ПП (1-6) | – | 2 |
| 21. Обмеження на розміри основного сховища даних (1-6) | – | 2 |
| 22. Різноманітність використовуваних обчислювальних платформ (1-6) | – | 2 |
| 23. Професійний рівень аналітиків (1-6) | – | 2 |
| 24. Професійний рівень програмістів (1-6) | – | 2 |
| 25. Постійність складу команди розробників (1-6) | – | 2 |
| 26. Досвід розробки додатків (1-6) | – | 2 |
| 27. Досвід роботи з обчислювальною платформою (1-6) | – | 2 |

Продовження таблиці 7.1

| 1 | 2 | 3 |
|---|---|---|
| 28. Досвід роботи з мовою і інструментами середовища розробки (1-6) | – | 2 |
| 29. Досвід роботи з програмними інструментами розробки (1-6) | – | 3 |
| 30. Розробка ПЗ для декількох серверів одночасно (1-6) | – | 2 |
| 31. Вимоги до дотримання встановленого графіка робіт (1-6) | – | 2 |
| 32. Вартість ПЗ у розробника (НМА), грн. | – | 15000 |
| 33. Норматив додаткової зарплати, % : | $Н_д$ | 10 |
| 34. Норматив відрахувань у соціальні фонди, % | Нс | 22 |
| 35. Норматив загальногосподарських витрат, % | Нг | 15 |
| 36. Норматив витрат на освоєння нових мов програмування, % | Нп | 15 |
| 37. Рівень рентабельності програмної продукції, % | Ре | 60 |
| 38. Ставка податку на додану вартість, % | Ндв | 20 |



### 7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B, \qquad (7.1)$$

де: $A$ – коефіцієнт Боема, $A$ = 2,45;

Size – загальний об’єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \quad (7.2)$$

де: *Wi* – сумарне значення п’яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}02\ 7.$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \quad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 6{,}78 \cdot (0{,}88 \cdot 0{,}93 \cdot 0{,}88 \cdot 0{,}91 \cdot 0{,}95 \cdot 1 \cdot 1 \cdot 0{,}87 \cdot 1{,}22 \cdot 1{,}16 \cdot 1{,}1 \cdot 1{,}1 \cdot$$
$$\cdot 1{,}12 \cdot 1{,}1 \cdot 1 \cdot 1{,}1 \cdot 1{,}1) = 9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об’ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \quad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4);

*S* – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 3{,}23 \cdot 9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)} \cdot 86 = 168 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

| Стадії розробки | Трудомісткість за типовими нормами та розрахунками | |
|---|---|---|
| | Величина, люд/дні | Підстава |
| Технічне завдання | 9 | Д5 |
| Ескізний проект | 10 | Д6 |
| Технічний проект | 9 | Д7 |
| Робочий проект | 168 | Ф 7.1- 7.4 |
| Впровадження | 13 | Д13 |
| Всього | 209 | – |



### 7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \qquad (7.5)$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{209 \cdot 1}{60 - 5} = 3,8 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

| Найменування обладнання | Профілактичне обслуговування | | | |
|---|---|---|---|---|
| | Кількість хв. на один. обл. | Кількість облад-нання | Затрати часу в хв. | Затрати часу в год. |
| Системний блок ПК | 90 | 7 | 630 | 10,5 |
| Монітор | 60 | 7 | 420 | 7 |
| Клавіатура | 30 | 7 | 210 | 3,5 |
| Маніпулятор «мишка» | 30 | 7 | 210 | 3,5 |
| Принтер матричний | 60 | 0 | 0 | 0,0 |
| Принтер лазерний | 120 | 2 | 240 | 4 |
| Принтер струминний | 60 | 1 | 60 | 1 |
| Сканер | 20 | 1 | 20 | 0,33 |
| Концентратор– маршрутизатор | 30 | 2 | 60 | 1 |
| Кабельні господарства ЛВС на 1 м. п. | 2,5 | 300 | 750 | 12,5 |
| Копіювальний апарат | 140 | 1 | 140 | 2,33 |
| Усього за рік: | | | $З_ч$ | 45,66 |

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{міс}}{1,2}, \quad (7.6)$$

$$Ф_{др}^{с} = \frac{45,66 \cdot 3}{1,2} = 114,15 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф^{с}_{др}}{F_{др} \cdot T_{зм}}, \quad (7.7)$$

$$Ч_{ел} = 114{,}15/(60 \cdot 8) = 0{,}2 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків. Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

| Посада | Вид роботи | Час | К-ть штатних одиниць |
|---|---|---|---|
| Адміністратор загальної мережі, аналітик | Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi | 0,5 | 0,25 |
| | Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS) | 0,5 | |
| | Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ | 0,5 | |
| | Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет | 0,5 | |
| Всього | | 2 | |

Продовження таблиці 7.4

| Посада | Вид роботи | Час | К-ть штатних одиниць |
|---|---|---|---|
| Продакт-менеджер | Презентації нової продукції, пошук каналів збуту | 1 | 0,25 |
| | Підтримка постійних клієнтів | 0,5 | |
| | Оформлення договорів, ведення тендерів | 0,25 | |
| | Контроль взаєморозрахунків з постачальниками | 0,25 | |
| Всього | | 2 | |
| Дизайнер WEB | Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію | 0,5 | 0,6 |
| | Створення графічних і стилістичних елементів сайту | 2,0 | |
| | Оформлення банерів і промо-сторінок | 0,3 | |
| | Розміщення графіки і контенту на Інтернет сторінках | 2,0 | |
| Всього | | 4,8 | |
| Інженер верстальник | Розробка та верстка макетів рекламної продукції та технічної документації | 1,0 | 0,4 |
| | Верстка друкованих видань | 1,0 | |
| | Додрукова підготовка макетів | 0,6 | |
| | Розміщення графіки і контенту на Інтернет сторінках | 0,6 | |
| Всього | | 3,2 | |

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

| Посада | Кількість ставок | Середньомісячний оклад, грн. | Всього за період розробки, грн. |
|---|---|---|---|
| Керівник (ІТ-менеджер) | 1 | 15540 | 46620 |
| Продакт-менеджер | 0,25 | 15000 | 11250 |
| Інженер-програміст | 3,8 | 15000 | 171000 |
| Інженер-електронщик | 0,2 | 15000 | 9000 |
| Адміністратор мережі | 0,25 | 15000 | 11250 |
| Дизайнер WEB | 0,6 | 15000 | 27000 |
| Інженер-верстальник | 0,4 | 15000 | 18000 |
| Бухгалтер-економіст | 0,5 | 15000 | 22500 |
| Всього за період розробки | $R_{сп} = 7$ | - | $Ф_{роб} = 316620$ |

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{316620}{7 \cdot 60} = 754 \; грн.$$

## 7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць;

$S_y$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пекторaль» (м. Кропивницький, вул. Глинки 16) ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 200…1600 $у.о./м^2$.

Враховуючи, що курс складає 1 у.о. = 37 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м$^2$.

На кожне робоче місце у середньому потрібно 8 $м^2$.

З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$I_{нв} = R^1_{сп} \cdot Ц_м, \qquad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 8 \cdot 3500 = 28000 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.6.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Supercomp https://supercomp.kiev.ua/ за 09.11.23.

Таблиця 7.6 – Специфікація

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| Персональний комп’ютер | | 10747 |
| Системний блок | | 7347 |
| Процесор | Intel Core™ i3 10105 1200, 4/8 потоків, 3.7 GHz, 4.4 GHz, TDP - 65 Вт, 14nm | - |
| Системна плата | ASUS PRIME H510M-K сокет - 1200, DDR4, 3200 MHz, LAN - 1 Гбіт/с, D-Sub (VGA), HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX | - |
| Відеокарта | Вбудована Intel UHD Graphics 630 | - |
| Жорсткий диск | SSD M.2 2280 240GB Apacer (AP240GAS2280P4-1) 240 GB, 3D TLC, M.2, PCI Express 3.0 x4 | - |
| Оперативна пам’ять | DDR4 8GB 2666 MHz Kingston | - |
| Блок живлення | Gamemax 500W (GM-500B) ATX 12V v2.3, 500 Вт, 20+4 pin, CPU - 4+4pin, GPU - 1x6 pin, SATA - 3, Peripheral - 2, +12V1 - 20A, 1x120 мм, 150 x 140 x 86 мм | - |
| Корпус | Vinga CS210B Класичний, Miditower, ATX, Micro - ATX, Mini - ITX, Слотів розширення - 7, Додатково - Front 2x12 см, Back 1x12 см, Side 2x12 см, GPU - до 370 мм, CPU - 166 мм, PSU - без обмежень, 413 x 198 x 422 мм | - |

Продовження таблиці 7.6

| Найменування комплектуючої або обладнання | Тип | Оптова ціна |
|---|---|---|
| Кардрідер внутрішній | USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black | - |
| інше | Клавіатура, мишка | Подару-нок |
| Монітор | 22" TFT, ASUS VW223D ( 5ms, 300/3000: 1, 170/160, D-SUB, Wide) | 3400 |
| Принтер лазерний | Canon i-SENSYS LBP6030W | 2700 |
| Принтер струминний | Epson Stylus Photo P50 (C11CA45341) + USB cable | 5500 |
| Копіювальний апарат | Canon i-SENSYS MF217W with Wi-Fi | 5965 |

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

| Найменування обчислювальної техніки | Кількість, шт. | Ціна за одиницю, грн. | Витрати на транспортування, монтаж та випробовування. | Загальна вартість, грн. |
|---|---|---|---|---|
| Персональні комп’ютери | 8 | 10747 | 8597,6 | 94573,6 |
| Принтер лаз. | 2 | 2700 | 540 | 5940 |
| Принтер струм. | 1 | 5500 | 550 | 6050 |
| Копіюв. апарат | 1 | 5965 | 596,5 | 6561,5 |
| Всього | – | – | – | 113125,1 |

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

| Групи та види основних фондів | Балансова вартість, грн. | Амортизація | |
|---|---|---|---|
| | | Норма, % | Відрахування, грн. |
| 1 | 2 | 3 | 4 |
| Група 3 | | | |
| 1. Будівлі | 1280000 | - | - |
| 2. Передавальні пристрої | 128000 | - | - |
| Всього по групі | 1408000 | 5 | 70400 |
| Група 4 | | | |
| 3. Обчислювальна техніка | 113125 | - | - |
| Всього по групі | 113125 | 40 | 45250 |
| Нематеріальні активи | | | |
| 4. Нематеріальні активи | 15000 | 10 | 1500 |
| Група 5, 6 | | | |
| 5. Вимірювальні пристрої | 5190 | 25 | 1297,5 |
| 6. Транспортні засоби | 0 | 20 | 0,0 |
| 7. Господарський інвентар | 28000 | 25 | 7000 |
| Всього по групі | 33190 | - | 8297,5 |
| Разом | $К_р$ = 1569315 | | $А_р$ = 125447,5 |

Примітка: вартість транспортного засобу приймаємо рівним нулю.

## 7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 754 \cdot 209/150 = 1050 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 1050 \cdot 10 \cdot 0{,}01 = 105 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $Н_с = 22\%$ від суми основної та додаткової зарплати:

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де: $Н_с$ – відрахування на соціальні потреби, %.

$$С_{оц} = 0{,}01 \cdot 22(1050+105) = 254 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $Н_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де: $Н_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 1050 \cdot 15 \cdot 0{,}01 = 158 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам'ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картриджів, тонеру, грн.;

$N_e$ – кількість екземплярів програм, шт.

Згідно виданих норм приймаємо 0,6 пачки паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п = 200$ грн., визначаємо вартість паперу за період розробки $N_м = 3$ міс:

$$З_{М1} = Ц_п \cdot N_м. \quad (7.16)$$

$$З_{М1} = 200 \cdot 0{,}6 \cdot 3 = 360 \text{ грн.}$$

Згідно виданих норм до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків в кількості 10 примірників:

$$З_{М2} = \sum Ц_д\,, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 32,5 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 32,5 грн./шт.

$$З_{М2} = 32{,}5 \cdot 10 = 325 \text{ грн.}$$

Згідно виданих норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_з., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570 = 1702 \text{ грн.}$$

$$З_М = (360 + 325 + 1702)/150 = 16 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п = 15\%$) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0{,}01, \quad (7.19)$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 1050 \cdot 15 \cdot 0{,}01 = 158 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_e = 150$ прим.):

$$A_м = \frac{A_р \cdot N_{міс}}{N_е \cdot 12}, \qquad (7.20)$$

де: $A_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$A_м = 125448 \cdot 3/(15 \cdot 12) = 209 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

| Найменування статей витрат | Позначення | Величина, грн. |
|---|---|---|
| 1 | 2 | 3 |
| 1. Основна зарплата виконавців | $З_о$ | 1050 |
| 2. Додаткова зарплата виконавців | $З_д$ | 105 |
| 3. Відрахування на соціальні потреби | $С_{оц}$ | 254 |
| 4. Загальногосподарські витрати | $Г_{осп}$ | 158 |
| 5. Витрати на матеріали | $З_М$ | 16 |
| 6. Освоєння нових операційних систем, мов програмування | $О_п$ | 158 |
| 7. Амортизація основних фондів | $А_м$ | 209 |
| 8. Повна собівартість програмного забезпечення | $С_п$ | 1950 |
| 9. Плановий прибуток | $П_р$ | 1170 |
| 10.Ціна підприємства $Ц_п = С_п + П_р$ | $Ц_п$ | 3120 |
| 11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$ | $ПДВ$ | 624 |
| 12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$ | $Ц$ | 3744 |

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$C_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \quad (7.21)$$

$$C_п = 1050+105+254+158+16+158+209 = 1950 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності ($P_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 60%.

$$П_р = 0{,}01 \cdot P_п \cdot C_п, \quad (7.22)$$

де: $P_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 60 \cdot 1950 = 1170 \text{ грн.}$$

## 7.6 Визначення об'єму капітальних вкладень у споживача програмної продукції

Об'єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Вартість програмного рішення взятого для порівняння складе 15000 грн. Результати розрахунків зводимо у таблицю 7.9.

Таблиця 7.10 – Розрахунок об'єму капітальних вкладень у споживача програмної продукції

| Найменування капітальних вкладень | Сума за варіантами, грн. | |
|---|---|---|
| | Базовий | Новий |
| Вартість програмної продукції | 4000 | 3744 |
| Всього капітальних витрат | 4000 | 3744 |

### 7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

| Найменування статей витрат | Позначення | Сума витрат за варіантами, грн. | |
|---|---|---|---|
| | | Базовий | Новий |
| 1. Витрати на обслуговування системи | $З_р$ | 63360 | 47520 |
| 2. Витрати на електроенергію | $З_{ел}$ | 0 | 0 |
| 3. Витрати на амортизацію | $З_{ам}$ | 1000 | 936 |
| Всього витрат за рік | $І$ | 64360 | 48456 |

Витрати на обслуговування системи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування сервера за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість годин для обслуговування системи зменшилось з 400 годин на рік до 300 годин на рік, тому витрати складуть:

$$З_{р\,баз} = 400 \cdot 120 \cdot 1{,}1 \cdot 1{,}22 = 63360\ грн,$$

до:

$$З_{р\,нов} = 300 \cdot 120 \cdot 1{,}1 \cdot 1{,}22 = 47520\ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

Визначити різницю споживання електроенергії при впровадженні систем не має можливості, тому витрати на електроенергію в розрахунку приймаємо рівними нулю.

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

| Групи основних фондів | Норма амортизації % | Балансова вартість, грн., за варіантами | | Сума відрахувань, грн., за варіантами | |
|---|---|---|---|---|---|
| | | Базовий | Новий | Базовий | Новий |
| Програмна продукція | 25 | 4000 | 3744 | 1000 | 936 |
| Всього відрахувань | - | 4000 | 3744 | 1000 | 936 |

## 7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \sum E_р K_р, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.

$$E_в = (3120 - 1950) \cdot 150 - (0,05 \cdot 1408000 + 0,4 \cdot 113125 + 0,25 \cdot 33190 + 0,1 \cdot 15000) \frac{3}{12} = 144138 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_p^*}{(Ц_п - С_п) \cdot N_е}, \qquad (7.26)$$

де: $K_p^*$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1569315}{(3120 - 1950) \cdot 150 \cdot 12 / 3} = 2,2 \text{ років}.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

| Найменування показників | Одиниця виміру | Величина |
|---|---|---|
| 1. Кількість екземплярів програми | Прим. | 150 |
| 2. Повна собівартість розробленої програми | Грн. | 1950 |
| 3. Ціна розробленої програми | Грн. | 3120 |
| 4. Плановий прибуток від реалізації розробленої програми | Грн. | 1170 |
| 5. Рентабельність програмної продукції | % | 60 |
| 6. Об□єм додаткових капітальних вкладень у виробника програмної продукції | Грн. | 1569315 |
| 7. Загальний прибуток від реалізації програмної продукції | Грн. | 175500 |
| 8. Величина економічного ефекту при виготовлені програмної продукції | Грн. | 144138 |
| 9. Період окупності додаткових капітальних вкладень у виробника програмної продукції | Років | 2,2 |
| 10.Об□єм додаткових капітальних вкладень у споживача програмної продукції | Грн. | 3744 |
| 11.Величина економічного ефекту у користувача програмної продукції | Грн. | 15968 |
| 12.Період окупності додаткових капітальних вкладень у користувача програмної продукції | Років | 0,1 |

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б + E_н K_б) - (I_н + E_н K_н), \qquad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (64360+0{,}25\cdot4000)-(48456+0{,}25\cdot3744) = 15968 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{\Delta K}{E_{сп}}, \qquad (7.28)$$

$$T_{сп} = \frac{4000 - 3744}{15968} < 0{,}1 \; року .$$

Як бачимо з розрахунків запропонований варіант е більш економічно доцільним ніж варіант вибраний для порівняння, який на даний момент є лідером продаж на ринку.

### 7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

# 8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

## 8.1 Вступ

В кожній ІТ компанії є трудові відносини з працівниками. Згідно закону України “Про охорону праці” кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

- Інструкцій з охорони праці по кожній професії і загальні;
- Положення про охорону праці;
- Накази з охорони праці;
- Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних

машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби польців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робитників галузі інформаційних технологій (ІТ) спеціалісти відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98, та «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

- параметри повітряного середовища в приміщенні;
- вентиляція приміщення;
- освітлення приміщення;
- параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп'ютера на організм людини визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.



### 8.2 Шкідливі і небезпечні фактори при роботі з комп'ютером

Електронно-обчислювальна машин (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом. Так як робота програміста характеризується істотним зоровим навантаженням, то вимагає належного освітлення. У приміщенні, в якому працюють люди (у т.ч. програмісти) необхідно

створити належний мікроклімат, параметри якого регламентуються, Державними санітарними правилами і нормами, зокрема ДСанПіН 3.3.2.007-98.

При роботі з використанням ЕОМ відзначають наступні небезпечні та шкідливі фактори:

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.
- ризик виникнення пожежі;
- негативний вплив на органи зору людини;
- ризики ураження електричним струмом;
- недостатня, або надмірна освітленість робочого місця;
- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання;
- несприятливі мікрокліматичні умови;
- нервово-емоційна напруженість праці;
- інтелектуальні навантаження;
- монотонність праці;
- невідповідність ергономічних показників робочого місця діючим вимогам;
- шум;
- статичні навантаження на кістково-м'язовий апарат.



## 8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

| Найменування | Значення, м |
|---|---|
| Ширина | 5 |
| Довжина | 6,2 |
| Висота | 3,4 |

Таблиця 8.2 - Площа та обсяг приміщення, на одного працюючого*

| Геометрична характеристика | Одиниця виміру | Нормативне значення* | Фактичне значення |
|---|---|---|---|
| Площа, S | $м^2$ | не менше 6.0 | 6,2 |
| Об'єм, V | $м^3$ | не менше 20.0 | 21,8 |

* Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

У зазначеному приміщенні працюють двоє людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин»). Таким чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря,

виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

| **Пора року** | **Оптимальні для Іа** | | | **Фактичні** | | |
|---|---|---|---|---|---|---|
| | **Температура, °С** | **Вологість,%** | **Швидкість повітря, м/с** | **Температура,°С** | **Вологість%** | **Швидкість повітря, м/с** |
| Холодна | 22-24 | 40-60 | 0,1 | 22-23 | 40-58 | 0,11 |
| Тепла | 23-25 | 50-70 | 0,1 | 24-25 | 50-65 | 0,2 |

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер *Epson L132 (C11CE58403)*, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання

програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми україни “Природне і штучне освітлення” – ДБН В.2.5-28:2018, у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп’ютером, згідно ДБН В.2.5-28:2018, можна віднести до роботи з малою точністю (найменший розмір об’єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об’єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об’єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк., Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп’ютера повинні бути приблизно однаковими.



### 8.4 Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

- розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;
- мікроклімат відповідає нормативному значенню;
- акустичні умови роботи не перевищують нормативних значеннь;

Таким чином можна припустити, що основною причиною можнивого

зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов'язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга).

Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В.

Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

### 8.5. Розрахункова частина

Проводемо розрахунок штучного освітлення за методом коефіцієнта використання світлового потоку для приміщення ширина якого складає 2,6 м, довжина – 2,6 м, висота – 3 м.

У зазначеному приміщенні працює 1 особа.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = (E \cdot S \cdot K \cdot Z) / n,$$

де $F$ – світловий потік, що розраховується, Лм; $E$ – нормована мінімальна освітленість, Лк; $E$ = 300 Лк; $S$ – площа освітлюваного приміщення (у нашому випадку $S$=2,6 х 2,6 = 6,76 м$^2$); $K$ – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку $K$ = 1,5); $Z$ – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку $Z$ = 1,1); $n$ – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$I = S / (h \cdot (A + B)),$$

де $S$ – площа приміщення, $S$ = 6,76 м$^2$; $h$ – розрахункова висота підвісу, $h$ = 3 м (співпадає з висотою стелі, т.я. лампи освітлення закріплюються на стелі); $A$ – ширина приміщення, $A$ = 2,6 м; $B$ – довжина приміщення, $B$ = 2,6 м.

Підставимо всі значення у формулу та визначимо індексу приміщення: і=0,43.

Знаючи індекс приміщення, за знаходимо $n$ = 0,23 (з табличних даних коефіцієнтів використання світлового потоку ($n$) світильників з відповідним типом лампам). Підставимо всі значення у формулу, визначимо світловий потік: *F=14548 Лм.*

Для розрахунку дудемо використовувати світлодіодні панелі *LED панель 42Вт 6000К SUNLED 000000127*, світловий потік яких $F_л$ = *3990 Лм.*

Число ламп визначається по формулі:

$$N = F / F_л$$

де $F$ – світловий потік, $F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо індексу приміщення:

$$N= 14548/ 3990=3,6 \text{ шт.}$$

Приймаємо необхідну кількість ламп 4 шт.

### 8.6 Висновки

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва вцілому.

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров'я програміста. Розроблено заходи з охорони праці.

# 9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для реалізації нейронних мереж, що здійснюють управління ігровими об’єктами у казуальній аркадній комп’ютерній грі.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження і розробки методів інтелектуального управління ігровими об’єктами.

Рішення поставленого завдання полягало у вирішенні наступних задач:

– Дослідження існуючих підходів та методів для управління ігровими об’єктами.

– Розробка методів та алгоритмів системи управління ігровими об’єктами на основі нейронних мереж.

– Програмна реалізація комп’ютерної гри та системи управління ігровими об’єктами на основі нейронних мереж.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання інтелектуального управління об’єктами у комп’ютерній грі.

Проведено аналіз предметної галузі в ході якого були виявлені об’єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудовані алгоритми і вибране середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python у середовищі розробки Anaconda та з застосуванням бібліотек pygame, gym, torch. Бібліотека pygame використана для створення казуальної аркадної комп'ютерної гри та інтерфейсу користувача. Бібліотека gym використана для створення віртуального навчального середовища для інтелектуальних агентів – супротивників гравця в розробленій казуальній аркадній грі. Бібліотека torch використана для реалізації нейронних мереж, які здійснюють управління агентами у комп'ютерній грі.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для захисту програмного забезпечення від несанкціонованого використання та поширення був обраний метод криптографічного захисту на основі алгоритму RSA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 15968 грн. З урахуванням вартості розробки програми та обладнання, строк окупності становить 0,1 роки.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Aggarwal C.C. Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edition. – Springer, 2018. – 520 p.

2. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 p.13. Кормен Т., Лейзерсон Ч., Риверст Р., Штайн К. Алгоритмы: построение и анализ, 3-е издание – М.: Диалектика, 2019. – 1328 p.

3. Ari N., Ustazhanov M. Matplotlib in python. 2014 11th International Conference on Electronics, Computer and Computation (ICECCO). IEEE, 2014.

4. Chung J., Gulcehre C., Cho K., Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. – Available from: https://arxiv.org/abs/1412.3555. – [Accessed February 2021].

5. Embarak O. Data analysis and visualization using python. Berkeley, CA, USA: Apress, 2018. DOI: https://doi.org/10.1007/978-1-4842-4109-7

6. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns. Elements of reusable object-oriented software. Addison Wesley. 1994. 395 c.

7. Gym Documentation. – URL: https://gymnasium.farama.org/

8. Hermans M., Schrauwen B. Training and analysing deep recurrent neural networks // in Advances in neural information processing systems. – 2013. – Vol 2: p. 190-198.

9. Howard J., Gugger S. Deep Learning for Coders with fastai and PyTorch. O'Reilly Media, 2020.

10. Hunter J. D. Matplotlib: A 2D graphics environment. Computing in science & engineering 9.03. 2007. P. 90-95.

11. Knuth D. E. The Art of Computer Programming: Fundamental Algorithms. 3rd Ed. Addison-Wesley, 1997.

12. Knuth D. E. The Art of Computer Programming: Seminumerical Algorithms. 3rd Ed. Addison-Wesley, 1998.

13. Knuth D. E. The Art of Computer Programming: Sorting and Searching. 2nd Ed. Addison-Wesley, 1998.

14. Lambert K. A. Fundamentals of Python: First Programs, 2nd Edition. – Cengage, 2019.

15. Lutz M. Learning Python, 5th Edition Fifth Edition. - O'Reilly Media, 2016. - 1643 p.

16. Lutz M. Python: Pocket Reference Fourth Edition. - O'Reilly Media, 2016. - 210 p.

17. Matplotlib. – URL: http://matplotlib.org

18. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd Edition. – O'Reilly Media, 2018.

19. Michael I. Jordan, Cristopher M. Bishop. Neural Networks. Massachusets Institute of Technology. AI Memo No. 1562. Anonymous ftp://publications.ai.mit.edu.

20. Nagpal A., Gabrani G. Python for data analytics, scientific and technical applications. 2019 Amity international conference on artificial intelligence (AICAI). IEEE, 2019.

21. Nelli F. Python data analytics: Data analysis and science using PANDAs, Matplotlib and the Python Programming Language. Apress, 2015.

22. Pygame Front Page. Quick start. – URL: https://www.pygame.org/docs/

23. PyTorch documentation. – URL: https://pytorch.org/docs/stable/index.html

24. Rogel-Salazar J. Data science and analytics with Python. CRC Press, 2018.

25. Stevens E., Antiga L., Viehmann T. Deep learning with PyTorch. Manning Publications, 2020.

26. The Python Tutorial. –https://docs.python.org/3/tutorial/index.html

27. Tosi S. Matplotlib for Python developers. Packt Publishing Ltd, 2009.

28. Wentworth P., Elkner J., Downey A., Meyers C. How to Think Like a Computer Scientist: Learning with Python 3. – Green Tea Press, 2018.

29. Wood D. Data Structures, Algorithms, and Performance. Addison-

Wesley, 1993.

30. Wood D. Data Structures, Algorithms, and Performance. Addison-Wesley, 1993.

31. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - ІваноФранківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с.

32. Архітектура персонального комп'ютера, Тема 3 - Загальні принципи архітектури комп'ютерів, 3.1 Принципи побудови комп'ютера. Архитектура Фон Неймана, 3.3 Архітектура і структура ПК

33. Бреславець В. С. Технології розробки комп'ютерних ігор : [довідник модуля] / В. С. Бреславець. – Харків : Друкарня Мадрид, 2018. – 162 с. – URI: https://repository.kpi.kharkov.ua/handle/KhPI-Press/37404

34. Гудфеллоу Я., Бенджіо І., Курвілль А. Глибоке навчання 2017р. Глава 3. Теорія ймовірності і теорія інформації 61с.

35. Гудфеллоу Я., Бенджіо І., Курвілль А. Глибоке навчання 2017р. Глава 5. Основи машинного навчання 96с.

36. Гудфеллоу Я., Бенджіо І., Курвілль А. Глибоке навчання 2017р. Глава 2. Лінійна алгебра 44с.

37. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://e-construction.gov.ua/laws_detail/3074958732556240833?doc_type=2

38. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

39. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

40. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

41. Кормен Т. Х., Лейзерсон Ч. І., Ривест Рональд Л., Штайн Кліффорд, Алгоритми. Побудова і аналіз. 2020. 1328 с.

42. Мелешко Є. В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних // Навчальний посібник для студенів технічних спеціальностей денної та заочної форми навчання. – Кропивницький: Видавець – Лисенко В.Ф., 2019. – 156 с. – URL: http://dspace.kntu.kr.ua/jspui/handle/123456789/8944

43. Моделі життєвого циклу, принципи і методології розробки програмного забезпечення [Електронний ресурс]. – https://evergreens.com.ua/articles/ software-development-metodologies.html

44. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

45. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

46. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград: КІСМ, 1997. - 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

47. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

48. Пчелянський Д., Воінова С. (2019). Штучний інтелект: перспективи та тенденції розвитку. Automation of Technological and Business Processes, 11(3),

59-64. https://doi.org/10.15673/atbp.v11i3.1500

49. Ракович В. А. (2017) Аналіз засобів розробки ігор для навчання майбутніх інженерів-програмістів. Ukrainian Journal of Educational Studies and Information Technology, 5 (2). с. 32-36.

50. Смірнов О.А., Коваленко О.В., Мелешко Є.В., Константинова Л.В., Кожанова А.С. Інженерія програмного забезпечення // Навчальний посібник для студентів вищих навчальних закладів напрямів підготовки 6.050102 «Комп'ютерна інженерія». Гриф "Навчальний посібник" надано у відповідності з листом Міністерства освіти і науки, молоді та спорту України від 18.03.13 року №1/11-5584 – Кіровоград: КНТУ 2013. – 335 с.

51. Сміт Б. «Beginning JSON» / Бен Сміт – К. : «Apress», 2015. – 324 с.

52. Соколова Н., Гнатушенко В., Міщенко М., Атаманчук О. (2022). Моделювання поведінки неігрових персонажів на основі штучного інтелекту. Прикладні питання математичного моделювання, Вип. 5(1). – С. 87-94.

53. Субботін С. О. Нейронні мережі : теорія та практика : навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.

54. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навчальний посібник. - Запоріжжя: ЗНТУ, 2008. - 341 с.

55. Центр післядипломної освіти та підвищення кваліфікації. - Режим доступу до ресурсу: https://cpo.stu.cn.ua

Додаток А
(обов’язковий)

**Технічне завдання**

**Зміст**

## 1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи моделювання комп’ютерних мереж та процесів передачі даних.

## 2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №__-__ від __.__.20__ року).

## 3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація системи моделювання комп’ютерних мереж та процесів передачі даних.

## 4 Джерела розробки

Джерелом цієї магістерської роботи є стосовна до теми література і існуючі аналоги.

## 5 Технічні вимоги

### 5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці розробників програмного забезпечення;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

### 5.2 Показники призначення

Система повинна забезпечувати:

– моделювання комп’ютерних мереж та процесів передачі даних;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

### 5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

### 5.4 Вимоги до архітектури

Застосунок, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

### 5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

### 5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

### 5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

### 5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

#### 5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

#### 5.8.2 Мова програмування

Мова програмування високого рівня Python у середовищі розробки програмного забезпечення Anaconda.

### 5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

### 5.8.4 Вихідні дані

Робоча програма.

## 6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

## 7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 1 вересня 2023 року.

## 8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи повинні бути розглянуті умови праці програмістів під час розробки програмного забезпечення.

## 9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуші.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 96 аркушів.

## 10 Етапи розробки

10.1 Збір і обробка інформації по темі магістерської роботи. Постановка задачі на виконання магістерської роботи (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень магістерської роботи.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

## 11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист 10.12.2023 р.

11.2 Подання магістерської роботи на захист __.__.12.2023 р.

Додаток Б

(обов’язковий)

**Міністерство освіти і науки України**

**Центральноукраїнський національний технічний університет**

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи

за другим (магістерським) рівнем вищої освіти

_____ Є.В. Мелешко



***Дослідження та програмна реалізація нейронних мереж для управління ігровими об’єктами***

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 14

Літера: РП

Кропивницький – 2023 року

# Основна програма

## Файл NeuroGame.py основної програми

```
# -*- coding: utf-8 -*-
"""
Created on 2023

@author: Kononchuk M.S.
"""

import pygame

import random
import numpy as np

import gym
from gym import spaces
import matplotlib.pyplot as plt

import gc

import torch
import torch.nn as nn
import torch.nn.functional as F

import os

os.environ['KMP_DUPLICATE_LIB_OK']='True'


# Клас для роботи нейронної мережі, яка управляє агентами - супротивниками у грі
class NeuralNetwork(nn.Module):

    def __init__(self, observation_space, action_space):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(observation_space, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, action_space)

    def forward(self, x):
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x

    def select_action(state, network):
        with torch.no_grad():
            state = torch.from_numpy(state).float().unsqueeze(0)  # Додаємо
додатковий вимір для пакету
            action_probs = network(state)
            action = action_probs.max(1)[1].view(1, 1)
            print("Action chosen:", action)  # Для тестування
        return action

    def compute_loss(state, next_state, action, reward, done,
neural_network_index):

        # Перевірка, чи всі необхідні дані присутні
        if state is None or next_state is None or action is None or reward is
None:
            return None
```

```
        # Перетворення в тензори
        state = torch.from_numpy(state).float()
        next_state = torch.from_numpy(next_state).float()
        reward = torch.tensor(reward).float()
        action = torch.tensor(action).long()

        # Отримання прогнозованих значень від нейронної мережі
        predicted_values = neural_networks[neural_network_index](state)

        # Вибір значення для зробленої дії
        action = action.view(1) # Зміна форми action до [batch_size, 1]
        predicted_value = predicted_values.gather(-1, action).squeeze(0)

        # Розрахунок цільового значення
        target_value = reward
        if not done:
            target_value = reward + 0.99 *
neural_networks[neural_network_index](next_state).max(0)[0]

        # Обчислення втрат
        loss = F.mse_loss(predicted_value, target_value.detach())

        return loss


# клас для створення ігрового середовища для нейронної мережі, у якому вона буде
діяти, оримувати винагороди та покарання і навчатися

class SimpleGameEnv(gym.Env):

    def __init__(self):
        super(SimpleGameEnv, self).__init__()

        # Створюємо простір можливих дій нейронної мережі, вона може пересувати
ігрового агента в різні сторони
        # Дії: 0 - вліво, 1 - вправо, 2 - вгору, 3 - вниз
        #      4 - вгору-вліво, 5 - вгору-вправо, 6 - вниз-вліво, 7 - вниз-
вправо
        self.action_space = spaces.Discrete(8)


        # Спостереження: позиція агента та цілі
        self.observation_space = spaces.Box(low=0, high=max(screen_width,
screen_height), shape=(4,), dtype=np.float32)

        # Ініціалізація стану
        self.state = None
        self.next_state = None

        self.agents = [Agent(random.randint(100, 700), random.randint(100, 500),
(255, 0, 255), f"Агент {i}") for i in range(3)]

        # Ініціалізація цілей – кульок, які треба збирати
        self.targets = [Target() for _ in range(10)]

        # Ініціалізація гравця, яким буде управляти людина
        self.player = Player(100, 100, (0, 0, 255), "Гравець")

        self.reset()

    # метод обчислення дистанції між агентом та ціллю

    def calculate_distance(self, agent, target):
        return np.linalg.norm([agent.x - target.x, agent.y - target.y])
```

```
    # метод для виконання агентом одного кроку та нарахування йому балів та
винагород за цей крок

    def step(self, action, agent_index, nearest_target, rnd = False):

        agent = self.agents[agent_index]

        d1 = self.calculate_distance(agent, nearest_target)

        if rnd:
            step_size = random.randint(5, 100)
        else:
            step_size = 1

        # Оновлення позиції агента

        if action == 0: # вгору
            agent.x = agent.x - step_size
        elif action == 1: # вниз
            agent.x = agent.x + step_size
        elif action == 2: # вліво
            agent.y = agent.y - step_size
        elif action == 3: # вправо
            agent.y = agent.y + step_size

        # Додавання діагональних напрямків

        elif action == 4:  # вгору-вліво
            agent.x -= step_size
            agent.y -= step_size
        elif action == 5:  # вгору-вправо
            agent.x += step_size
            agent.y -= step_size
        elif action == 6:  # вниз-вліво
            agent.x -= step_size
            agent.y += step_size
        elif action == 7:  # вниз-вправо
            agent.x += step_size
            agent.y += step_size



        done = False
        reward = -0.01

        d2 = self.calculate_distance(agent, nearest_target)

        if d2 < d1:
            reward += 2

        elif d2 == d1:
            reward -= 0

        elif d2 > d1:
            reward -= 2

        if agent.x < 0 or agent.x > screen_width or agent.y < 0 or agent.y >
screen_height:
            reward -= 1.0  # Мінусувати бали за вихід за межі екрану
            agent.x, agent.y = random.randint(100, 700), random.randint(100,
500)

        for target in self.targets[:]:
            if self.check_collision(agent, target):
                if target == nearest_target:
                    agent.score += target.radius
                    reward += target.radius
                else:
                    agent.score += 3
```

```
                    reward += 3

                # Перевірка, чи ціль ще є в списку перед її видаленням
                if target in self.targets:
                    self.targets.remove(target)
                    self.targets.append(self.create_new_target())

                reward = 1.0
                done = True

                break

        # Пошук найближчої за відстанню цілі
        nearest_target = self.find_nearest_target(agent, env.targets)

        # Оновлення поточного стану середовища для агента
        self.state = np.array([agent.x, agent.y, nearest_target.x,
nearest_target.y])

        return self.state, reward, done, {}


    # скидання стану ігрового середовища
    def reset(self):

        # Випадкове розміщення кожного агента
        for agent in self.agents:
            agent.x = random.randint(100, 700)
            agent.y = random.randint(100, 500)


       # Випадкове розміщення кожної цілі
        for target in self.targets:
            target.x = random.randint(100, 700)
            target.y = random.randint(100, 500)
            target.color = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))


        self.state = None
        return np.array(self.state)

    # рендер ігрового середовища
    def render(self, mode='human'):
        screen.fill((255, 255, 255))
        for agent in self.agents:
            agent.draw()
        for target in self.targets:
            target.draw()

        self.player.draw()

        pygame.display.flip()

        # Відображення балів
        font = pygame.font.SysFont(None, 24)
        score_text = font.render(f"Гравець: {self.player.score}", True, (0, 0,
0))
        screen.blit(score_text, (screen_width - 150, 10))
        for i, agent in enumerate(self.agents):
            agent_score_text = font.render(f"{agent.name}: {agent.score}", True,
(0, 0, 0))
            screen.blit(agent_score_text, (screen_width - 150, 40 + i * 30))


        pygame.display.flip()
```

```
    # метод визначення колізій між агентом (або гравцем) і ціллю
    def check_collision(self, entity, target):

        # Розрахунок центру агента (гравця), якщо агент - квадрат
        entity_center_x = entity.x + entity.size / 2
        entity_center_y = entity.y + entity.size / 2

        # Відстань між центрами агента (гравця) і цілі
        distance = np.linalg.norm([entity_center_x - target.x, entity_center_y -
target.y])

        # Перевірка зіткнення: чи відстань менша або дорівнює сумі радіусів
        return distance <= (entity.size / 2 + target.radius)

    # метод створення нової цілі
    def create_new_target(self):
        return Target()

    # метод знаходження найближчої цілі у просторі
    def find_nearest_target(self, agent, targets):
        nearest_target = None
        min_distance = float('inf')
        for target in targets:
            distance = np.linalg.norm([agent.x - target.x, agent.y - target.y])
            if distance < min_distance:
                min_distance = distance
                nearest_target = target
        print(agent.name, nearest_target.x, nearest_target.y)
        return nearest_target

    # метод закриття вікна гри
    def close(self):
        # Закриття вікна гри
        pygame.quit()


# Класи для об'єктів гри

# клас «гравець»
class Player:

    # конструктор класу
    def __init__(self, x, y, color, name):

        self.x = x
        self.y = y
        self.size = 20
        self.color = color
        self.name = name
        self.score = 0

    # метод зміни розміру гравця при збільшенні кількості набраних балів
    def update_size(self, n):

        # Оновлення розміру на основі балів
        self.size = n

    # метод малювання гравця на екрані
    def draw(self):

        pygame.draw.rect(screen, self.color, (self.x, self.y, self.size,
self.size))
        font = pygame.font.SysFont(None, 24)
        name_text = font.render(self.name, True, (0, 0, 0))
        screen.blit(name_text, (self.x, self.y - 20))
```

```
    # метод керування гравцем з клавіатури
    def handle_player_movement(player):

        keys = pygame.key.get_pressed()
        if keys[pygame.K_LEFT]:
            player.x -= 5
        if keys[pygame.K_RIGHT]:
            player.x += 5
        if keys[pygame.K_UP]:
            player.y -= 5
        if keys[pygame.K_DOWN]:
            player.y += 5

# клас «агент»
class Agent:

    # конструктор класу
    def __init__(self, x, y, color, name):

        self.x = x
        self.y = y
        self.size = 20
        self.color = color
        self.name = name
        self.score = 0

    # метод малювання агента на екрані
    def draw(self):

        pygame.draw.rect(screen, self.color, (self.x, self.y, self.size,
self.size))
        font = pygame.font.SysFont(None, 24)
        name_text = font.render(self.name, True, (0, 0, 0))
        screen.blit(name_text, (self.x, self.y - 20))

    # метод зміни розміру агента при збільшенні кількості набраних балів
    def update_size(self, n):

        # Оновлення розміру на основі балів
        self.size = n


# клас «ціль»
class Target:

    # конструктор класу
    def __init__(self):

        k = 100
        self.x = random.randint(0+k, screen_width-k)
        self.y = random.randint(0+k, screen_height-k)
        self.radius = random.randint(5, 30)
        self.color = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))

    # метод малювання цілі на екрані
    def draw(self):

        pygame.draw.circle(screen, self.color, (self.x, self.y), self.radius)


# Ініціалізація Pygame
pygame.init()

# Створення вікна
screen_width, screen_height = 800, 600
screen = pygame.display.set_mode((screen_width, screen_height))
```

```
# Задання назви вікну
pygame.display.set_caption("Гра 'Збери кульки' з нейронною мережею")


env = SimpleGameEnv()

# Розмір простору спостережень нейронної мережі - 4, а простір її дій - 4
observation_space = 4
action_space = 8


# Створення трьох окремих нейронних мереж для кожного агента
neural_networks = [NeuralNetwork(observation_space, action_space) for _ in
range(3)]


# Завантаження вагів у нові нейронні мережі

p = False # True - якщо завантажити ваги нейронних мереж з файлів (якщо вони
були попередньо збережені з минулих запусків гри), False - не завантажувати

if p == True:
    for i, network in enumerate(neural_networks):
        network.load_state_dict(torch.load(f'agent_{i}_model.pth'))


# Ініціалізація оптимізатора
optimizer = [torch.optim.Adam(neural_networks[i].parameters(), lr=0.001) for i
in range(3)]

# Ініціалізація змінних для збору даних
rewards = [[] for _ in range(3)]  # Якщо у грі 3 агенти
rwrd = 0
losses = [[] for _ in range(3)]


running = True



# Головний цикл гри
while running:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((255, 255, 255))

    # Управління гравцем
    Player.handle_player_movement(env.player)


    for agent_index, agent in enumerate(env.agents):
        nearest_target = env.find_nearest_target(agent, env.targets)
        if nearest_target:
            state = np.array([agent.x, agent.y, nearest_target.x,
nearest_target.y])

            # Вибір дії за допомогою нейронної мережі
            action = NeuralNetwork.select_action(state,
neural_networks[agent_index])

            # Виконання дії
            next_state, rwrd, done, _ = env.step(action, agent_index,
nearest_target)
            rewards[agent_index].append(rwrd)
```

```
            # Обчислення втрат
            loss = NeuralNetwork.compute_loss(state, next_state, action, rwrd,
done, agent_index)

            if loss is not None:
                losses[agent_index].append(loss.item())

            # Оновлення ваг мережі
            optimizer[agent_index].zero_grad()
            loss.backward()
            optimizer[agent_index].step()

            # Оновлення стану
            state = next_state

            # Оновлення позиції агента
            if done:

                for agent in env.agents:

                    agent.score += nearest_target.radius

                    if agent.score > 100:
                        agent.update_size(30)

                    elif agent.score > 500:
                        agent.update_size(40)

                    elif agent.score > 1000:
                        agent.update_size(50)

                    elif agent.score > 1500:
                        agent.update_size(60)

                    elif agent.score > 2000:
                        agent.update_size(70)


                if nearest_target in env.targets:
                    env.targets.remove(nearest_target)
                    env.targets.append(env.create_new_target())


                # Оновлюємо найближчу ціль для агента
                nearest_target = env.find_nearest_target(agent, env.targets)

                if nearest_target:
                    state = np.array([agent.x, agent.y, nearest_target.x,
nearest_target.y])
                    action = NeuralNetwork.select_action(state,
neural_networks[agent_index])
                    next_state,  rwrd, done, _ = env.step(action, agent_index,
nearest_target)
                    rewards[agent_index].append(rwrd)

                    # Обчислення втрат
                    loss = NeuralNetwork.compute_loss(state, next_state, action,
rwrd, done, agent_index)
                    losses[agent_index].append(loss.item())

                    # Оновлення ваг мережі
                    optimizer[agent_index].zero_grad()
                    loss.backward()
                    optimizer[agent_index].step()

                    # Оновлення стану
                    state = next_state
```

```
    # Оновлення стану для нової найближчої цілі
    nearest_target = env.find_nearest_target(agent, env.targets)
    state = np.array([agent.x, agent.y, nearest_target.x, nearest_target.y])
    action = NeuralNetwork.select_action(state, neural_networks[agent_index])
    next_state, rwrd, done, _ = env.step(action, agent_index, nearest_target)
    rewards[agent_index].append(rwrd)

    # Обчислення втрат
    loss = NeuralNetwork.compute_loss(state, next_state, action, rwrd, done,
agent_index)
    losses[agent_index].append(loss.item())

    # Оновлення ваг мережі
    optimizer[agent_index].zero_grad()
    loss.backward()
    optimizer[agent_index].step()

    # Оновлення стану
    state = next_state

    # Перевірка зіткнення гравця з цілями

    for target in env.targets[:]:

        if env.check_collision(env.player, target):
            env.player.score += target.radius

            if nearest_target in env.targets:
                env.targets.remove(target)
                env.targets.append(env.create_new_target())  # Додавання нової
цілі

        if env.player.x < 0 or env.player.x > screen_width or env.player.y < 0
or env.player.y > screen_height:
            env.player.score -= 1.0  # Мінусувати бали за вихід за межі екрану



    if env.player.score > 100:
       env.player.update_size(30)

    elif env.player.score > 500:
        env.player.update_size(40)

    elif env.player.score > 1000:
        env.player.update_size(50)

    elif env.player.score > 1500:
        env.player.update_size(60)

    elif env.player.score > 2000:
        env.player.update_size(70)

    # Перевірка зіткнення агентів з цілями
    for agent in env.agents:
        for target in env.targets[:]:

            if env.check_collision(agent, target):
                agent.score += target.radius

                if nearest_target in env.targets:
                    env.targets.remove(target)
                    env.targets.append(env.create_new_target())

                    nearest_target = env.find_nearest_target(agent, env.targets)
                    if nearest_target:
                        state = np.array([agent.x, agent.y, nearest_target.x,
nearest_target.y])
```

```
                        action = NeuralNetwork.select_action(state,
neural_networks[agent_index])
                        next_state, rwrd, done, _ = env.step(action,
agent_index, nearest_target)
                        rewards[agent_index].append(rwrd)


                        # Обчислення втрат
                        loss = NeuralNetwork.compute_loss(state, next_state,
action, rwrd,  done, agent_index)
                        losses[agent_index].append(loss.item())


                        # Оновлення ваг мережі
                        optimizer[agent_index].zero_grad()
                        loss.backward()
                        optimizer[agent_index].step()


                        # Оновлення стану
                        state = next_state


            if agent.x < 0 or agent.x > screen_width or agent.y < 0 or agent.y >
screen_height:
                agent.score -= 1.0  # Мінусувати бали за вихід за межі екрану



    # Відображення об'єктів гри
    screen.fill((255, 255, 255))
    env.player.draw()

    for agent in env.agents:
        agent.draw()

    for target in env.targets:
        target.draw()


    # Відображення балів
    font = pygame.font.SysFont(None, 24)
    score_text = font.render(f"Гравець: {env.player.score}", True, (0, 0, 0))
    screen.blit(score_text, (screen_width - 150, 10))

    for i, agent in enumerate(env.agents):
        agent_score_text = font.render(f"{agent.name}: {agent.score}", True, (0,
0, 0))
        screen.blit(agent_score_text, (screen_width - 150, 40 + i * 30))


    env.render()  # Оновлення візуалізації гри
    pygame.display.flip()


# функція для створення таблиці рейтингів

def show_scores(player, agents):

    # Збирання даних про гравця та агентів
    scores = [('Гравець', player.score)] + [(agent.name, agent.score) for agent
in agents]


    # Сортування за балами
    scores.sort(key=lambda x: x[1], reverse=True)
```

```
    # Виведення таблиці рейтигрів
    print("\nРейтинги:")

    print("-" * 30)

    print("{:<10} {:<10}".format('Ім\'я', 'Бали'))

    for name, score in scores:
        print("{:<10} {:<10}".format(name, score))



# виведення таблиці рейтингів по заверщенню гри
show_scores(env.player, env.agents)



# підключення бібліотеки для роботи з часом
import datetime


# функція для береження рейтигів у файл

def save_scores_to_file(player, agents):


    # Отримання поточної дати та часу
    now = datetime.datetime.now()

    date_time = now.strftime("%Y-%m-%d %H_%M_%S")

    filename = "ratings" + date_time + ".txt"


    # Збирання даних про гравця та агентів
    scores = [('Гравець', player.score)] + [(agent.name, agent.score) for agent
in agents]


    # Сортування за балами
    scores.sort(key=lambda x: x[1], reverse=True)


    # Відкриття файлу для запису
    with open(filename, "w", encoding="utf-8") as file:

        file.write(f"Дата та час гри: {date_time}\n")
        file.write("-" * 30 + "\n")
        file.write("{:<10} {:<10}\n".format('Ім\'я', 'Бали'))

        for name, score in scores:
            file.write("{:<10} {:<10}\n".format(name, score))

        file.close()



# Збереження рейтигів у файл
save_scores_to_file(env.player, env.agents)




pygame.display.quit()

# Закриття Pygame
pygame.quit()
```

```
torch.cuda.empty_cache()
gc.collect()



plt.ion()



# Побудова графіків винагород

plt.figure(figsize=(10, 4))
plt.plot(rewards[0])

plt.title(f"Нагороди Агента 0")
plt.xlabel("Епізод")
plt.ylabel("Нагорода")

plt.show()


plt.figure(figsize=(10, 4))
plt.plot(rewards[1])

plt.title(f"Нагороди Агента 1")
plt.xlabel("Епізод")
plt.ylabel("Нагорода")

plt.show()


plt.figure(figsize=(10, 4))
plt.plot(rewards[2])

plt.title(f"Нагороди Агента 2")
plt.xlabel("Епізод")
plt.ylabel("Нагорода")

plt.show()



# Побудова графіків втрат

plt.figure(figsize=(10, 4))
plt.plot(losses[0])

plt.title(f"Втрати Агента 0")
plt.xlabel("Крок")
plt.ylabel("Втрати")

plt.show()


plt.figure(figsize=(10, 4))
plt.plot(losses[1])

plt.title(f"Втрати Агента 1")
plt.xlabel("Крок")
plt.ylabel("Втрати")

plt.show()


plt.figure(figsize=(10, 4))
plt.plot(losses[2])

plt.title(f"Втрати Агента 2")
```

```
plt.xlabel("Крок")
plt.ylabel("Втрати")

plt.show()



print(«Вагові коефіцієнти нейронних мереж збережено у файл. Їх можна буде використати для інших запусків гри»)


# Збереження вагових коефіцієнтів кожної нейронної мережі
for i, network in enumerate(neural_networks):
    torch.save(network.state_dict(), f'agent_{i}_model.pth')
```