МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університетімені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доц.
Є. В. Сіденко
«__»_____2023 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

Система інтелектуального керування персонажем комп'ютерної гри у жанрі "Аркада"

Спеціальність 124 «Системний аналіз»

124 – МКР – 607.21710104

Виконав студент 6-го курсу, групи 607
_____*М. В. Боляк*
_____«_____»2023 р.

Керівник: *д.т.н., професор*
_____*О. П. Гожий*
«_____»2023 р.

Миколаїв – 2023

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **124 «Системний аналіз»**
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____ Є. В. Сіденко
«____» _____ 2022р.

ЗАВДАННЯ
на магістерську кваліфікаційну роботу
Боляку Микиті В'ячеславовичу
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Система інтелектуального керування персонажем комп'ютерної гри у жанрі "Аркада"».
Керівник роботи Гожий Олександр Петрович, д-р техн. наук, професор.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «03» листопада 2022 р. № 197
2. Строк подання студентом роботи 17 лютого 2023 р.
3. Вхідні (початкові) дані до роботи: огляд існуючих ігор та технологій ШІ, розробка дизайну та концепції гри, вибір алгоритмів машинного навчання, розробка користувацького інтерфейсу та бази ігрових даних для навчання алгоритмів. Очікуваний результат: робочий прототип аркадної гри з використанням бота на основі штучного інтелекту. Прототип повинен продемонструвати потенціал штучного інтелекту в керуванні персонажем гри.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

- Вивчення та огляд існуючих аркадних ігор та технологій штучного інтелекту, пов'язаних з цією сферою.

− Розробка дизайну та концепції гри, яка буде використана як основа для інтелектуальної системи управління персонажем.

− Вибір та реалізація алгоритмів машинного навчання, які будуть використовуватися для управління персонажем та прийняття рішень у реальному часі.

− Інтеграція алгоритмів машинного навчання та ігрового дизайну для створення робочого прототипу інтелектуальної системи управління персонажем.

− Оцінка та тестування прототипу для визначення його продуктивності та ефективності у забезпеченні цікавого та захоплюючого ігрового досвіду.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: забезпечення здоров'я та безпеки, шляхом дотримання правил безпеки, належного навчання та обладнання, ергономічного дизайну, моніторингу стану здоров'я, регулярних оцінок безпеки та документування заходів з безпеки.

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці		
Методична частина	д-р техн. наук, професор Гожий О.П.	

Керівник роботи __д-р техн. наук, проф. Гожий О. П._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Боляк М.В._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__06__» _____листопада____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання магістерської кваліфікаційної роботи

Тема: Система інтелектуального керування персонажем комп'ютерної гри у жанрі Аркада

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2022	20.10.2022	Виконано
2	Отримання завдання на виконання МКР	21.10.2022	10.11.2022	Виконано
3	Складання календарного плану на період виконання МКР	11.11.2022	15.11.2022	
4	Огляд літератури за темою дослідження	16.11.2022	04.12.2022	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	05.12.2022	25.12.2022	Виконано
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	26.12.2022	12.01.2023	Виконано
7	Опис фахової частини МКР, зокрема дослідження публікацій щодо використання ШІ для керування персонажами гри, огляд існуючих архітектур штучних нейронних мереж для вирішення поставленої задачі, реалізація обраних технологій з аналізом отриманих результатів	13.01.2023	25.01.2023	Виконано
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2023	02.02.2023	Виконано
9	Попередній захист МКР на засіданні комісії кафедри	03.02.2023	03.02.2023	Виконано
10	Корегування роботи за результатами попереднього захисту	04.02.2023	06.02.2023	Виконано
11	Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту	07.02.2023	09.02.2023	Виконано
12	Подання МКР рецензенту	09.02.2023	10.02.2023	Виконано
13	Рецензування МКР	11.02.2023	12.02.2023	Виконано
14	Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	16.02.2023	17.02.2023	Виконано
15	Захист МКР перед екзаменаційною комісією (ЕК)	24.02.2023	24.02.2023	

Розробив студент _БолякМ.В._____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи __ д-р техн. наук професор Гожий О.П._____ _____
(*наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«12» листопада 2022 р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 607 ЧНУ ім. Петра Могили
Боляка Микити В'ячеславовича
на тему: **"СИСТЕМА ІНТЕЛЕКТУАЛЬНОГО КЕРУВАННЯ ПЕРСОНАЖЕМ КОМП'ЮТЕРНОЇ ГРИ У ЖАНРІ АРКАДА"**

Актуальність дослідження полягає в застосуванні алгоритмів машинного навчання, сприяючи розвитку досліджень і технологій ШІ. Розробка інтелектуальної системи управління персонажем має потенціал для подолання деяких обмежень традиційних аркадних ігор, таких як необхідність постійної взаємодії з гравцем та обмежений час реакції людини.

Об'єктом дослідження є процес розробки інтелектуальної системи управління персонажем для аркадних ігор.

Предметом дослідження є методи штучного інтелекту та алгоритми аркадних ігор.

Метою роботи є вдосконалення та підвищення швидкодії системи управління персонажем для аркадних ігор.

Результат роботи є високоінтелектуальною системою управління персонажем для аркадних ігор. Система використовує передові алгоритми машинного навчання, щоб дозволити головному герою приймати рішення і виконувати дії в режимі реального часу на основі поточного стану гри. Алгоритми навчання з підкріпленням і глибокого Q-навчання дозволять системі вчитися на власному досвіді та вдосконалюватися з часом

Дана робота складається з п'яти розділів. Кожен розділ відповідно присвячений: аналізу предметної області, математичним моделям і методам, використаним у магістерській роботі, моделюванню і проектуванню системи управління ботом гри та тестуванню, охороні праці, методичній частині магістерської роботи. Загальний обсяг роботи – 98 сторінок. Магістерська кваліфікаційна робота містить 60 рисунків, 9 таблиць і посилання на 45 літературних джерел.

Ключові слова: навчання з підкріпленням, deep-Q learning, PyTorch, бот, ШІ.

ABSTRACT

to the master's qualification work by the student of the group 607 of Petro Mohyla Black Sea National University

Boliak Mykyta

“ Intelligent character control system for an arcade computer game ”

The relevance of the study lies in the application of machine learning algorithms, contributing to the development of AI research and technology. The development of an intelligent character management system has the potential to overcome some of the limitations of traditional arcade games, such as the need for constant interaction with the player and limited human reaction time.

The object of research is the process of developing an intelligent character management system for arcade games.

The subject of research is artificial intelligence methods and algorithms for arcade games.

The aim of the work is improving and enhancing performance of a character management system for arcade games.

The result is a highly intelligent character management system for arcade games. The system uses advanced machine learning algorithms to allow the main character to make decisions and perform actions in real time based on the current state of the game. Reinforcement learning and deep Q-learning algorithms will allow the system to learn from its own experience and improve over time

This thesis consists of five chapters. Each chapter is devoted to: analysis of the subject area, mathematical models and methods used in the master's thesis, modeling and design of the game bot control system and testing, labor protection, and the methodological part of the master's thesis. The total volume of the work is 98 pages. The master's qualification work contains 60 figures, 9 tables and references to 45 literary sources.

Keywords: reinforcement learning, deep-Q learning, PyTorch, bot, AI.

ЗМІСТ

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«Система інтелектуального керування персонажем комп'ютерної гри у жанрі "Аркада"»

Спеціальність 124 «Системний аналіз»

124 – МКР – 607.21710104

Виконав студент 6-го курсу, групи 607
М.В. Боляк
_____
(підпис, ініціали та прізвище)
«___»_____2023 р.

Керівник: д.т.н., професор О.П. Гожий
(наук. ступінь, вчене звання)

_____
(підпис, ініціали та прізвище)
«___»_____2023 р.

Миколаїв – 2023

ВСТУП

Аркадні ігри були популярним джерелом розваг протягом десятиліть, надаючи гравцям цікавий і захоплюючий досвід.

Метою цього дипломного проекту є розробка інтелектуальної системи управління персонажем для аркадних ігор, яка може забезпечити надзвичайно цікавий та захоплюючий досвід для гравця, навіть без прямої взаємодії. Головним героєм гри буде керувати ШІ, приймаючи рішення в реальному часі та реагуючи на різні виклики та перешкоди. Гравець спостерігатиме за грою, спостерігаючи за тим, як персонаж, керований ШІ, переміщується в ігровому світі та досягає своїх цілей. Кінцевим результатом стане новий тип аркадної гри, який поєднує в собі азарт традиційних аркадних ігор з передовими можливостями ШІ, забезпечуючи абсолютно новий і захоплюючий ігровий досвід.

Для досягнення цієї мети в рамках проекту будуть проведені масштабні дослідження і розробки в галузі штучного інтелекту, геймдизайну та взаємодії людини з комп'ютером. Будуть вивчені та впроваджені різні алгоритми машинного навчання, включаючи навчання з підкріпленням, та нейронні мережі. Ці алгоритми будуть використані для навчання системи штучного інтелекту приймати інтелектуальні рішення та реагувати на вподобання гравця.

В цілому, завданням цього дипломного проекту є демонстрація потенціалу ШІ в революції в світі аркадних ігор і надання унікального і захоплюючого ігрового досвіду для гравців. Успішне завершення цього проекту стане відправною точкою для подальших досліджень і розробок у галузі аркадних ігор зі штучним інтелектом, а також продемонструє захоплюючі можливості поєднання передових технологій з традиційними формами розваг.

Крім того, цей проект також сприятиме розвитку досліджень у галузі ШІ, вивчаючи нові способи використання алгоритмів машинного навчання для покращення ігрового досвіду. Він також кине виклик традиційній концепції взаємодії гравців в аркадних іграх і дослідить нові форми непрямої взаємодії. Інтелектуальна система управління персонажем буде розроблена таким чином,

щоб працювати автономно, приймаючи рішення і виконуючи дії, оптимізовані для задоволення гравця. Такий підхід забезпечить свіжий та інноваційний погляд на аркадні ігри і продемонструє потенціал ШІ у створенні нових форм розваг.

Отже, розробка інтелектуальної системи управління персонажем для аркадних ігор - це складний і захоплюючий проект, який об'єднає різні галузі досліджень, такі як штучний інтелект, ігровий дизайн і взаємодія людини з комп'ютером. Проект забезпечить унікальний та захопливий ігровий досвід для гравців, а також сприятиме розвитку технологій, демонструючи потенціал штучного інтелекту в революційних змінах у світі ігор. Кінцевим результатом стане інноваційна аркадна гра.

Крім того, проект матиме практичне застосування не лише в ігровій індустрії. Інтелектуальна система управління персонажами, розроблена в цьому проекті, може бути адаптована і застосована в інших сферах, де прийняття рішень в режимі реального часу і автономна поведінка є важливими, наприклад, в робототехніці і автономних системах. Досліджуючи нові способи використання ШІ для управління персонажами і прийняття рішень, цей проект матиме ширший вплив на сферу штучного інтелекту і технологій в цілому.

Проект стане важливим кроком вперед в інтеграції ШІ та аркадних ігор, і він має потенціал надихнути майбутні розробки в цій галузі. Інноваційний підхід і використання передових технологій зроблять цей проект цінним внеском в академічну спільноту та ігрову індустрію. Успішне завершення цього проекту продемонструє можливості ШІ у покращенні ігрового досвіду.

Апробація результатів магістерської роботи. Основні результати магістерських досліджень доповідались та обговорювались на Науково-практичній конференції «Могилянські читання – 2023», підсекція «Машинне навчання та штучний інтелект» (2023 р., ЧНУ, ім. Петра Могили, Миколаїв).

Мета магістерської кваліфікаційної роботи – є вдосконалення та підвищення швидкодії системи управління персонажем для аркадних ігор.

Об’єкт дослідження – є процес розробки інтелектуальної системи управління персонажем для аркадних ігор.

Предмет дослідження – є методи штучного інтелекту та алгоритми аркадних ігор.

Завдання:

- вивчення та огляд існуючих аркадних ігор та технологій штучного інтелекту, пов'язаних з цією сферою;
- розробка дизайну та концепції гри, яка буде використана як основа для інтелектуальної системи управління персонажем;
- вибір та реалізація алгоритмів машинного навчання, які будуть використовуватися для управління персонажем та прийняття рішень у реальному часі;
- інтеграція алгоритмів машинного навчання та ігрового дизайну для створення робочого прототипу інтелектуальної системи управління персонажем;
- оцінка та тестування прототипу для визначення його продуктивності та ефективності у забезпеченні цікавого та захоплюючого ігрового досвіду;
- подальша робота та рекомендації щодо вдосконалення та подальших досліджень.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ

1.1 Геймінг

Геймінг - це гра в електронну відеогру, яка часто здійснюється на спеціальній ігровій приставці, комп'ютері або смартфоні. Людей, які часто грають у відеоігри, називають геймерами.

Ігри є надзвичайно популярним видом діяльності у всьому світі. Кількість людей, які грають в ігри, оцінюється в понад 3 мільярди. За деякими оцінками, вартість світового ігрового ринку у 2022 році становитиме від $180 млрд до $220 млрд. Це приблизно вдвічі більше, ніж ринок світової кіноіндустрії. Більшість ігрового ринку - це люди, які грають в ігри на своїх смартфонах.

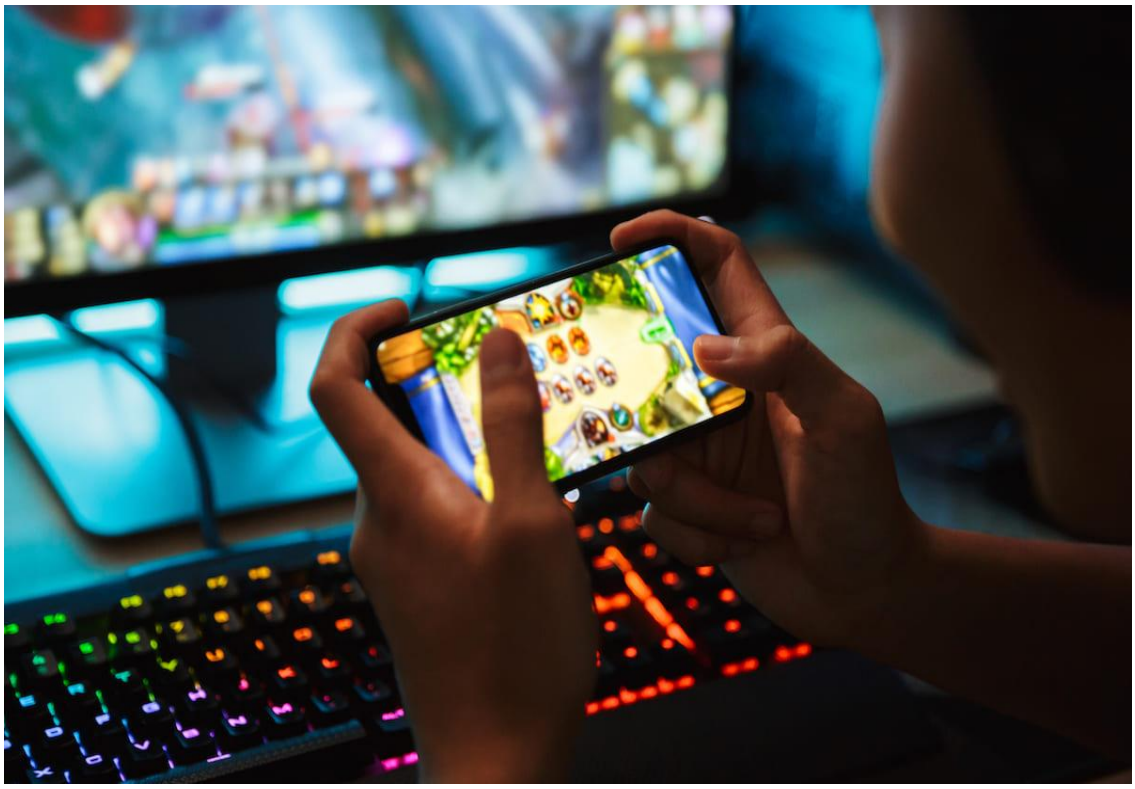

Рисунок 1.1 – Геймінг на будь-яких пристроях

Слово "геймінг" виникло в 15 столітті для позначення азартних ігор в кості або карткові ігри. Термін "геймінг" використовувався для позначення азартних

ігор до кінця 1900-х років, коли він застосовувався до розважальних настільних ігор та рольових ігор з ручкою та папером. З появою відеоігор у 1970-х роках термін "геймінг" став більше асоціюватися з його сучасним використанням [1].

Геймерів часто поділяють на кілька груп, залежно від того, як часто і серйозно вони займаються іграми.

Казуальний геймер - це той, хто нерегулярно грає в ігри або грає лише в одну гру низької інтенсивності. Прикладом випадкового гравця може бути той, хто грає в ігри зі словами або головоломки на своєму смартфоні, перебуваючи в громадському транспорті або вдома під час перегляду телевізора. Казуальні ігри становлять найбільшу частину ігрового ринку завдяки своїй широкій привабливості та простоті доступу, що забезпечується смартфонами. Казуальний геймер часто не ідентифікує себе як геймер.

Рисунок 1.2 – Казуальні геймери

Затятий геймер - це той, хто регулярно грає у відеоігри. Вони зазвичай володіють спеціальними пристроями для відеоігор, такими як консолі або ігрові комп'ютери. Затяті гравці часто грають у багато ігор, але зосереджуються на одній

грі або типі ігор, які їм особливо подобаються. Запеклий гравець зазвичай самоідентифікує себе як геймер.

Рисунок 1.3 – Затятий геймер

Професійний геймер - це той, хто заробляє гроші або утримує себе, граючи в ігри. Деякі з них є спортсменами з кіберспорту, які грають у відеоігри на професійному рівні на турнірах. Вони можуть бути частиною команди або клану і брати участь у трансляціях змагань. Вони можуть заробляти гроші з призових фондів турнірів і від корпоративного спонсорства. Зазвичай вони зосереджуються лише на одній грі.

Рисунок 1.4 – Українська кіберспортивна команда "NAVI"

Стример заробляє гроші, публікуючи відео, на яких він грає в ігри. Ці відео часто робляться в прямому ефірі і викладаються на Twitch або YouTube. Вони заробляють гроші за рахунок пожертвувань глядачів, спонсорства та доходів від реклами.

Рисунок 1.5 – Український стример Віталій "Arthas" Цаль

Хоча це традиційно прийняті категорії геймерів, не існує остаточного способу класифікувати всіх геймерів, і ці категорії мають перекіс у бік традиційної ігрової культури. Наприклад, пенсіонерка, яка багато годин на день грає у фермерську гру або судоку на своєму iPad, буде розглядатися багатьма

людьми як випадковий геймер, хоча вона, ймовірно, грає у відеоігри більше годин на місяць, ніж традиційний хардкор-геймер.

Ігри можуть сприяти когнітивному розвитку. Вони можуть містити головоломки та виклики, які гравці повинні розгадати та подолати. Деякі ігри вимагають від гравця багатозадачності та розстановки пріоритетів для досягнення найкращих результатів. Гравці можуть навчитися швидко приймати правильні рішення. Деякі ігри можуть імітувати реальні наукові, історичні та економічні системи. Деякі дослідження показали, що люди, які грають в комп'ютерні ігри, краще справляються із завданнями, що вимагають короткочасного відтворення в пам'яті.

Комп'ютерні ігри можуть сприяти розвитку просторового мислення. Гравці можуть навчитися формувати ментальні карти і орієнтуватися в 3D-просторі. Це може призвести до кращої навігації в реальному світі. Також було показано, що хірурги, які грають у відеоігри перед проведенням лапароскопічних або роботизованих операцій, мають кращі результати, ніж ті, хто цього не робить[2].

1.2 AI та ігрові боти

Сприйняття, мислення і дія - це загальні терміни, які визначають систему на основі ШІ. Ця система повинна бути здатна сприймати задане середовище, обробляти спостережувані та латентні структури в середовищі та приймати рішення про розумну (оптимальну) дію з урахуванням середовища та поставленої мети.

Як і скрізь, AI можна використовувати в іграх по-різному. Щоб визначити сфери, в яких можна використовувати ШІ, розділимо їх на дві частини, які назвемо "розважальна " і "бізнес", де перша пояснює використання ШІ в іграх, а друга - випадки використання в маркетингу та монетизації.

Ігри можуть мати різні таксономії, такі як ігри з нульовою сумою (проти), де одна сторона програє, а інша виграє, або багатокористувацько-кооперативні ігри, де два або більше гравців злагоджено працюють як члени команди, як

правило, проти одного або більше опонентів. Існують також інші таксономії для таких типів ігор, як екшн, стратегія, головоломки, аркади тощо. Ще одна таксономія може бути сформована за обсягом інформації, доступної для гравця: повна інформація та часткова інформація налаштувань. До перших відносяться ігри, стан яких можна спостерігати повністю (наприклад, шахи), а до других - ігри, в яких гравець може отримати лише часткове уявлення про стан гри (наприклад, покер, початкові етапи Red Alert 2 та Age Of Empires 2) [3].

Боти створюються з наборів алгоритмів, які допомагають їм у виконанні поставлених завдань. Існує безліч різних типів ботів, розроблених по-різному для виконання найрізноманітніших завдань, і це не є винятком для ботів в іграх. Різні типи ігор можуть вимагати різних типів дизайну ботів. Однак можна розділити ботів на дві основні категорії: вони можуть бути або на основі сценаріїв, або на основі штучного інтелекту. Боти на основі сценаріїв, як правило, розробляються за певним сценарієм, і вони реагують на основі правил для цього конкретного сценарію. Колись вони широко використовувалися в MMORPG, щоб легко підвищувати рівень (якщо здоров'я падає, використовуйте зілля, або визначте найближчого ворога і використовуйте список атак). Вони дійсно корисні як навчальні посібники та помічники для гравців в цілому. Але для широкого спектру ігрових процесів, особливо для людиноподібних, вони зовсім не ефективні. Боти на основі штучного інтелекту вирішують цю проблему, навчаючи програму методом проб і помилок і заохочуючи її, якщо вона робить хорошу роботу, або караючи, якщо вона помиляється [4].

Боти для відеоігор - це екземпляри штучного інтелекту (ШІ), які грають у гру замість гравця. Ігрові боти присутні майже в кожній багатокористувацькій грі, від Call of Duty до MMORPG.

Ігрові боти виконують різні завдання, починаючи від імітації дій гравця і закінчуючи NPC, які виконують сільськогосподарські роботи. Штучний інтелект пройшов довгий шлях з часів Pong.

Існують різні типи ігрових ботів:

- статичні ігрові боти потребують значної допомоги, щоб діяти належним

чином. Ці ігрові боти мають відносно невелику здатність рухатися за власним бажанням. Зазвичай вони існують для виконання лінійних функцій, таких як виконання вказівок або торгівля з гравцем у магазині;

- динамічні ігрові боти можуть брати участь у вивченні макетів рівнів, вивченні нових стратегій тощо. Поведінку таких ботів найбільше можна зустріти в такій популярній грі, як Counter-Strike;
- існують навіть ігрові боти, які займаються як статичною, так і динамічною поведінкою. Наприклад, Counter-Strike використовує як статичні точки проходження, так і динамічні дії та поведінку в онлайн-іграх, щоб діяти більш реалістично. Існують боти, які беруть участь як у виконанні повторюваних завдань, пов'язаних з дрібними ігровими деталями, так і виконують квестові завдання [5].

Між ігровим ботом та NPC може виникнути невелика плутанина. NPC, неігрові персонажі, ніколи не грають у гру. Боти, з іншого боку, моделюються на основі гравця для створення реального ігрового процесу. NPC, наприклад, в MMORPG, ніколи не мають на меті імітувати поведінку гравця в онлайн-грі.

NPC більше виступають в ролі декорації. Вони існують для того, щоб наповнити ігровий світ, але не взаємодіють з його елементами відкрито [6].

Ігрові боти - не єдині боти, з яких можна зустріти. В додатках або на веб-сайтах можна почати спілкуватися в чаті з ботом. Це бот, запрограмований на виконання певних завдань за допомогою штучного інтелекту. Багато ботів такого виду використовують навчене спілкування на основі проаналізованих даних та інформації за допомогою таких речей, як машинне навчання.

Оскільки ШІ активно використовується в іграх і стає все більш популярним, можна з упевненістю сказати, що в майбутньому використання ШІ в іграх буде збільшуватися [7].

Можна виділити декілька напрямків використання ШІ в іграх у майбутньому:

- використання ШІ в аркадних іграх, які вимагають безперервного проектування рівнів. Наприклад, ШІ може бути використаний для розуміння

стилю гравця і розташування будівельних блоків рівня таким чином, що він стає більш складним для гравця, одночасно усуваючи ручну працю при створенні рівня;

− цікавим варіантом використання ботів може бути розробка та впровадження ботів на основі ШІ, які адаптуються до ігрового процесу гравця. Бот на основі ШІ, який коригує свою стратегію з урахуванням стилю гри користувача, може кинути користувачеві більший виклик, ніж звичайний ШІ. Або NPC в грі можуть розробляти захист з урахуванням улюблених стратегій користувача;

− федеративне навчання: Це концепція спільного навчання мобільних телефонів за спільною моделлю. Без обміну даними між пристроями федеративне навчання має на меті побудувати спільну загальну модель машинного навчання розподілених даних, використовуючи обмін математичними оновленнями для моделі, а не явний обмін даними. У світі, де зберігається конфіденційність, слідування таким тенденціям стає обов'язковим[8].

1.3 Аналіз аналогів.

Нижче наведено найвідоміші приклади використання AI в історії ігор з кінця 90-х років.

1) DeepBlue, 1997. Розроблена компанією IBM шахова програма DeepBlue, навчена комп'ютерними фахівцями та експертами в цій галузі, кинула виклик гросмейстеру Гаррі Каспарову. У 1996 році Каспаров виграв матч, що складався з 6 партій. Після майже року перенавчання і доопрацювання, у 1997 році DeepBlue виграла другий матч з 6 партій. За умовами турніру це був перший випадок, коли чемпіон світу програв матч комп'ютеру [9];

Рисунок 1.6 – Суперкомп'ютер, розроблений компанією IBM

2) Monte Carlo Tree Search (MCTS) for Go, 2006. Програма під назвою CrazyStone показала чудові результати у грі в го на дошці 9×9, використовуючи нову техніку під назвою Monte Carlo Tree Search. Потім нова програма під назвою MoGo досягла рівня майстра на дошці 9×9;

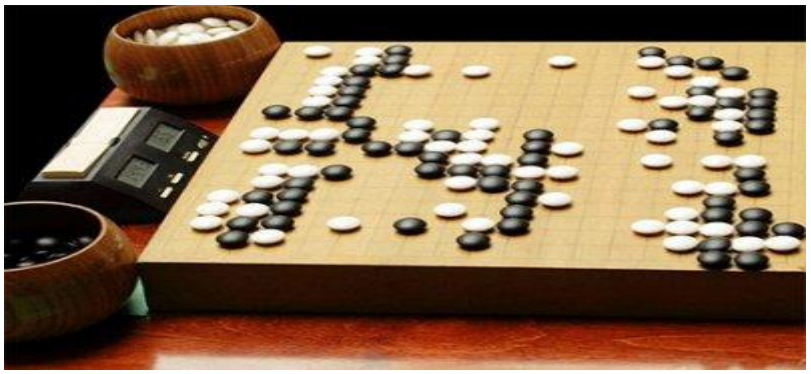

Рисунок 1.7 – Гра в го

3) deep-Q-Learning на іграх Atari Games, 2013. У 2012 році сталося щось революційне, переможцем конкурсу з класифікації зображень під назвою ImageNet стала глибока нейронна мережа (AlexNet), яку раніше вважали дуже складною для навчання. З тих пір феномен глибокого навчання торкнувся

практично всіх сфер машинного навчання (від завдань розпізнавання зображень до обробки мови і природної мови). Не оминула ця революція і ігрову індустрію. У 2013 році на конференції NeurIPS (на той час NIPS) компанія DeepMind представила алгоритм під назвою Deep-Q-Learning, який вирішує проблему людиноподібної гри Atari з новою, революційною ідеєю. Для того, щоб агент міг обирати відповідну дію з певного стану, вони навчили згорткову нейронну мережу на основі ігрових кадрів. Дозволивши їй оптимізувати свій ігровий процес протягом усього часу, цей агент навчання з підкріпленням показав продуктивність, яку раніше не міг показати жоден інший алгоритм. З тих пір геймплей Atari покращився, а новий алгоритм DeepMind, який отримав назву Agent57, перевершив стандартний людський бенчмарк на всіх 57 іграх Atari в 2020 році [10];

Рисунок 1.8 – Ігрова система Atari

4) AlphaGo, 2016. AlphaGo - це перша комп'ютерна програма, яка перемогла професійного гравця в Го, перша, яка перемогла чемпіона світу з Го, і, можливо, найсильніший гравець в Го в історії. AlphaGo використовує спостереження за

прикладами ігор, а потім самостійно грає тисячі життєвих циклів. Вона поєднує ідею дерева пошуку в Монте-Карло з нейронними мережами, які передбачають наступний хід і ймовірність перемоги. Експерти з гри в го були вражені деякими ходами, які AlphaGo зробила під час матчу з Лі Седолом;

Рисунок 1.9 – Лого AlphaGo

5) AlphaZero (on chess, Go and shogi), 2017. AlphaZero була представлена наприкінці 2017 року. Це єдина система, яка з нуля навчилася опановувати шахи, сьогі (японські шахи) та го, обігравши в кожному з них програму-чемпіона світу;

Рисунок 1.10 – Лого AlphaZero

6) OpenAI Five, 2017-2019. 13 квітня 2019 року OpenAI Five став першою AI-системою, яка перемогла чемпіонів світу з кіберспорту. Гра Dota 2 ставить нові виклики перед системами ШІ, такі як довгі часові горизонти, недосконала інформація та складні, безперервні простори дій зі станами - всі ці виклики набувають все більшого значення для більш потужних систем ШІ. OpenAI Five використовує існуючі методи навчання з підкріпленням, масштабовані для навчання на партіях приблизно в 2 мільйони кадрів кожні 2 секунди. Була розроблена розподілена система навчання та інструменти для безперервного навчання, що дозволило навчити OpenAI Five за 10 місяців. Перемігши чемпіона світу з Dota 2 (Team OG), OpenAI Five продемонстрував, що самонавчання з підкріпленням може досягти надлюдських показників у виконанні складних завдань[11].

Рисунок 1.11 – Матч OpenAi Five проти кіберспортивної команди OG

Висновки до розділу 1

Ігрові боти і штучний інтелект стають все більш вражаючими. Завдяки таким речам, як машина і глибоке навчання, ігрові боти будуть майже не відрізнятися від реальних гравців. Вони дозволяють гравцям, яким не вистачає товаришів по грі, все ще відчувати гру. Швидше за все, в майбутньому вам навіть не знадобляться товариші по команді.

Траєкторія глибокого навчання в галузі ШІ вражає, як це демонструють самокеровані автомобілі Tesla і не тільки.

До цього ігри використовували різні евристичні алгоритми на основі штучного інтелекту, які використовують моделі ймовірностей та/або структури дерев рішень для отримання рішень для ігор.

2 МОДЕЛІ, МЕТОДИ, АЛГОРИТМИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗРОБКИ ГРИ ТА AI БОТА

2.1 Нейронні мережі

Нейронні мережі - це тип алгоритму машинного навчання, змодельований за структурою та функціями людського мозку. Вони складаються з взаємопов'язаних вузлів, які називаються штучними нейронами, що обробляють і передають інформацію. Кожен нейрон отримує вхідні дані від інших нейронів, виконує прості обчислення на основі цих даних і виробляє вихідні дані, які можуть бути передані іншим нейронам. Використовуючи багато шарів взаємопов'язаних нейронів, нейронні мережі здатні вивчати складні взаємозв'язки між входами і виходами [12].

Процес навчання в нейронних мережах передбачає коригування сили зв'язків між нейронами, відомих як ваги, на основі продуктивності мережі на навчальному наборі даних. Цей процес відбувається за допомогою алгоритму оптимізації, наприклад, стохастичного градієнтного спуску, який налаштовує ваги в напрямку, що мінімізує різницю між прогнозами мережі та фактичними результатами.

Нейронні мережі можна використовувати для широкого кола завдань, включаючи класифікацію зображень, обробку природної мови та ігор. Вони особливо корисні для проблем, які включають великі обсяги даних і складні взаємозв'язки між входами і виходами, де традиційні методи можуть не давати точних прогнозів.

Отже, нейронні мережі - це потужний інструмент машинного навчання, здатний вивчати складні взаємозв'язки між входами і виходами. Вони складаються з взаємопов'язаних штучних нейронів, які обробляють і передають інформацію, а сила зв'язків між цими нейронами коригується в процесі навчання,

щоб мінімізувати різницю між передбаченнями мережі та фактичними результатами.

Архітектура нейронної мережі може широко варіюватися залежно від завдання, яке вона покликана виконувати. Деякі поширені типи архітектур нейронних мереж включають мережі прямого поширення, рекурентні мережі та згорткові нейронні мережі [13].

Мережі прямого поширення - це найпростіший тип нейронних мереж, в яких інформація рухається лише в одному напрямку, від входу до виходу, без зворотного зв'язку. Ці мережі часто використовуються для таких завдань, як класифікація зображень і регресія.

Рекурентні мережі, з іншого боку, дозволяють інформації повертатися назад і оброблятися кілька разів, що робить їх добре придатними для завдань, які включають послідовності даних, таких як моделювання мови і розпізнавання мови.

Згорткові нейронні мережі (ConvNets або CNN) - це тип мереж прямого поширення, які спеціально розроблені для обробки сіткоподібних даних, таких як зображення. ConvNets застосовують набір фільтрів до вхідних даних, що допомагає їм вивчати локальні закономірності та особливості даних. Вони зазвичай використовуються для класифікації зображень, виявлення об'єктів і сегментації зображень.

Існує також багато варіантів і модифікацій цих основних типів нейронних мереж, включаючи автокодери, генеративні змагальні мережі (GAN) і трансформаторні мережі.

На додаток до архітектури, вибір функцій активації, функцій втрат і оптимізаторів також відіграє значну роль у продуктивності нейронної мережі. Функції активації визначають вихід нейрона на основі його входів, функції втрат вимірюють різницю між прогнозами мережі та фактичними виходами, а оптимізатори налаштовують ваги мережі на основі функції втрат [14].

Для навчання нейронної мережі в мережу подається велика кількість маркованих навчальних даних, а ваги ітеративно коригуються за допомогою

алгоритму оптимізації, поки продуктивність мережі на навчальних даних не досягне задовільного рівня. Потім мережу можна протестувати на окремому наборі даних, щоб оцінити її продуктивність і внести необхідні корективи.

Загалом, нейронні мережі є універсальним і потужним інструментом машинного навчання, здатним вирішувати широкий спектр завдань, навчаючись на даних. Однак вони також можуть бути складними у розробці та навчанні, і слід бути обережними, щоб уникнути надмірного пристосування до навчальних даних.

Ще одним важливим аспектом нейронних мереж є поняття перенавчання. Перенавчання відбувається, коли нейронна мережа занадто добре навчена на навчальних даних і не здатна добре узагальнювати нові, невідомі дані. Це може призвести до поганої роботи на тестових даних, навіть якщо мережа має низьку похибку на навчальних даних. Щоб уникнути надмірного пристосування, можна використовувати такі методи, як регуляризація, рання зупинка та відсів [15].

Регуляризація додає штрафний член до функції втрат, що перешкоджає мережі вивчати надто складні представлення даних. Рання зупинка передбачає моніторинг продуктивності мережі на перевірочному наборі і зупинку процесу навчання, коли продуктивність на перевірочному наборі починає знижуватися, що вказує на те, що мережа надмірно пристосована. Відсіювання - це метод, коли випадкова підмножина нейронів у мережі тимчасово відсіюється під час кожної ітерації навчання, змушуючи мережу вивчати більш надійні представлення даних.

Нарешті, варто зазначити, що успіх нейронних мереж значною мірою залежить від наявності достатньої кількості якісних навчальних даних. Нейронні мережі є дуже гнучкими і можуть моделювати складні взаємозв'язки між входами і виходами, але вони також можуть бути схильні до засвоєння шаблонів у навчальних даних, які погано узагальнюються на нові дані. Тому необхідно ретельно підходити до розробки архітектури мережі, вибору функцій активації, функцій втрат і оптимізаторів, а також до якості навчальних даних [16].

Розуміння основ нейронних мереж, включаючи їхню архітектуру, процес навчання та потенціал перенавчання, має вирішальне значення для розробки ефективних та дієвих моделей.

2.2 Навчання з підкріпленням

Навчання з підкріпленням - це тип машинного навчання, де агент вчиться приймати рішення, виконуючи дії в середовищі, щоб максимізувати сигнал винагороди. Агент взаємодіє з навколишнім середовищем, виконуючи дії та отримуючи зворотній зв'язок у вигляді винагород або покарань. З часом агент навчається зіставляти стани з діями, які приносять найбільшу очікувану винагороду, що називається оптимальною політикою. Навчання з підкріпленням застосовується в різних сферах, таких як ігри, робототехніка, рекомендаційні системи та автономні транспортні засоби. Воно також використовується для вирішення складних проблем, таких як послідовне прийняття рішень і компроміси між розвідкою та експлуатацією.

У навчанні з підкріпленням процес прийняття рішень моделюється як марковський процес прийняття рішень (MDP), який складається зі станів, дій та винагород. На кожному часовому кроці агент спостерігає за поточним станом і вибирає дію для переходу в наступний стан. Потім середовище подає сигнал про винагороду на основі нового стану та виконаної дії. Мета полягає в тому, щоб знайти правила, які зіставляють стани з діями, що максимізують кумулятивну винагороду з часом, також відому як віддача.

Алгоритми навчання з підкріпленням можна розділити на два основні підходи: засновані на цінностях і засновані на правилах. Методи, засновані на цінностях, такі як Q-навчання, оцінюють цінність виконання певної дії в певному стані, а правила виводяться з цих цінностей. Методи, що базуються на правилах, такі як Policy Gradient, безпосередньо вивчають правила, зіставляючи стани з діями [17].

Навчання з підкріпленням є складною областю машинного навчання через труднощі пошуку оптимальних моделей у складних середовищах, балансування між пошуком та експлуатацією, а також навчання на основі рідкісних або запізнілих сигналів винагороди. Однак нещодавні досягнення в галузі глибокого навчання з підкріпленням, яке поєднує навчання з підкріпленням і глибокі

нейронні мережі, призвели до вражаючих результатів у різних сферах.

Загалом, навчання з підкріпленням є багатообіцяючою галуззю, яка може вплинути на багато галузей, дозволяючи агентам приймати розумні рішення та вирішувати складні проблеми.

Важливо зазначити, що алгоритми навчання з підкріпленням вимагають великої кількості даних, оскільки агенту потрібно дослідити середовище і зібрати достатньо інформації, щоб навчитися правильній політиці. Це може бути особливо складно для реальних додатків, де збір даних може бути дорогим або навіть неможливим [18].

Щоб подолати цю проблему, деякі дослідники розробили алгоритми навчання з підкріпленням, які можуть навчатися на синтетичних даних, таких як симуляції, або на демонстраціях, наданих людиною. Крім того, до навчання з підкріпленням було застосовано трансферне навчання, підгалузь машинного навчання, щоб дозволити агентам переносити знання, отримані в одному завданні, на інше, пов'язане з ним, зменшуючи кількість даних, необхідних для навчання.

Ще однією проблемою в навчанні з підкріпленням є так звана "проблема присвоєння кредиту", яка стосується труднощів пов'язування винагороди з конкретною дією, здійсненою в минулому. Цю проблему можна вирішити, використовуючи алгоритми, які можуть поширювати винагороду в минуле, наприклад, методи акторської критики, або використовуючи методи, які можуть вчитися на основі зворотного зв'язку з людиною, наприклад, зворотне навчання з підкріпленням [19].

Отже, навчання з підкріпленням є потужним інструментом для прийняття рішень і розв'язання проблем, але воно також створює багато проблем, які потребують вирішення. Тим не менш, останні досягнення в цій галузі показують, що воно має великий потенціал для майбутнього застосування в широкому спектрі областей.

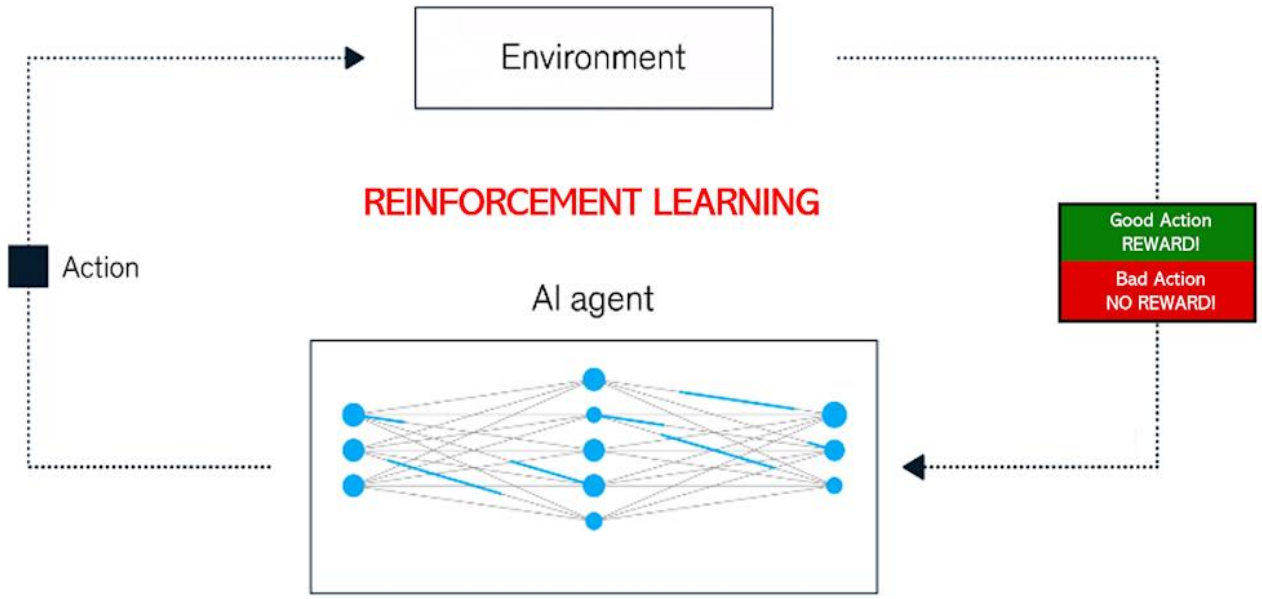

Рисунок 2.1 – Цикл навчання з підкріпленням

2.3 Deep Q-Learning

Deep Q-Learning (DQN) - це тип алгоритму навчання з підкріпленням, який поєднує глибокі нейронні мережі з Q-Learning, класичним методом навчання з підкріпленням. Мета DQN - навчити агента приймати рішення в навколишньому середовищі, максимізуючи сигнал винагороди. Агент навчається шляхом спроб і помилок, оновлюючи своє розуміння середовища на основі результатів своїх дій.

Навчання з підкріпленням - це область машинного навчання, яка фокусується на навчанні агента приймати рішення в середовищі, отримуючи винагороду або покарання за свої дії. У традиційному Q-Learning агент використовує таблицю для зберігання передбачуваної винагороди для кожної пари стан-дія. Однак ця таблиця може швидко стати громіздкою для складних середовищ з великою кількістю станів і дій [20].

DQN вирішує цю проблему, використовуючи глибоку нейронну мережу для апроксимації функції Q-значення замість таблиці. Нейронна мережа приймає стан середовища на вході і виводить оцінену винагороду за кожну можливу дію. Це дозволяє агенту узагальнити своє розуміння середовища і працювати зі складними просторами станів.

Ключовою інновацією DQN є використання відтворення досвіду, коли агент зберігає свій досвід у буфері пам'яті і випадковим чином вибирає його для навчання нейронної мережі. Це упорядковує дані, полегшуючи навчання мережі та зменшуючи ризик надмірної адаптації.

Іншим важливим аспектом DQN є використання мереж з фіксованою метою, де окрема мережа використовується для оцінки Q-значень для цільової політики. Це зменшує кореляцію між метою та прогнозом, що полегшує навчання мережі.

DQN був успішно застосований у різних складних середовищах, включаючи ігри Atari, керування роботами та складні настільні ігри, такі як Го. Поєднання глибокого навчання та навчання з підкріпленням зробило його популярним підходом для вирішення проблем у таких галузях, як робототехніка, управління та розробка ігор [21].

Рівняння Беллмана - це фундаментальне рівняння в навчанні з підкріпленням, яке пов'язує цінність стану з очікуваною цінністю майбутніх станів, до яких він може призвести. Рівняння виражає цінність стану як суму безпосередньої винагороди за виконання певної дії в цьому стані та очікуваної майбутньої цінності перебування в наступному стані.

У глибокому Q-навчанні рівняння Беллмана є основою для оновлення функції цінності Q, яка представляє очікувану винагороду за виконання певної дії в даному стані. Функція цінності Q оцінюється за допомогою глибокої нейронної мережі, яка навчена апроксимувати значення, передбачені рівнянням Беллмана.

Зокрема, глибока нейронна мережа приймає стан навколишнього середовища як вхідні дані і виводить оцінені Q-значення для кожної можливої дії. Під час навчання мережа оновлюється за допомогою рівняння Беллмана, яке порівнює прогнозовані значення якості з цільовими значеннями. Цільові значення якості отримуються за допомогою окремої мережі, відомої як цільова мережа, яка оновлюється рідше, щоб зменшити кореляцію між прогнозом і ціллю.

Рівняння Беллмана та функція Q-значення слугують основою для процесу навчання в DQN. Агент використовує функцію Q-значення для прийняття рішень в навколишньому середовищі і оновлює своє розуміння середовища,

використовуючи рівняння Беллмана для коригування своїх оцінок Q-значень. З часом агент вчиться приймати кращі рішення, які призводять до більшої винагороди, покращуючи свою загальну продуктивність [22].

Загальний вигляд рівняння Беллмана:

$$Q(S\mathrm{t},A\mathrm{t}) = (1-\alpha)\,Q(S\mathrm{t},A\mathrm{t}) + \alpha * (R\mathrm{t} + \lambda * max_{\mathrm{a}}\, Q(S_{\mathrm{t+1}},\alpha)) \qquad (2.1)$$

де Q - ключове поняття у навчанні з підкріпленням. Воно являє собою очікувану винагороду за виконання певної дії в певному стані.

S - стан або спостереження,

A - Дія, яку виконує агент,

R - Винагорода від виконання дії,

t - часовий крок,

α - швидкість навчання,

ƛ - коефіцієнт дисконтування, який призводить до того, що винагорода втрачає свою цінність з часом, тому більш негайні винагороди оцінюються вище

Отже, DQN - це потужний алгоритм навчання з підкріпленням, який поєднує глибокі нейронні мережі з Q-Learning для обробки складних просторів станів і підвищення стабільності навчання. Його використання мереж з відтворенням досвіду та фіксованою метою довело свою ефективність у багатьох додатках, і він залишається популярним підходом для вирішення проблем у різних галузях.

2.4 Мова програмування Python

Python - це високорівнева мова програмування загального призначення, яка стала популярною в різних галузях, включаючи науку про дані та машинне навчання. Однією з ключових причин її популярності є простота та легкість у використанні. Синтаксис Python простий, що робить її легкою для початківців, а також надає багато бібліотек і модулів для досвідчених користувачів.

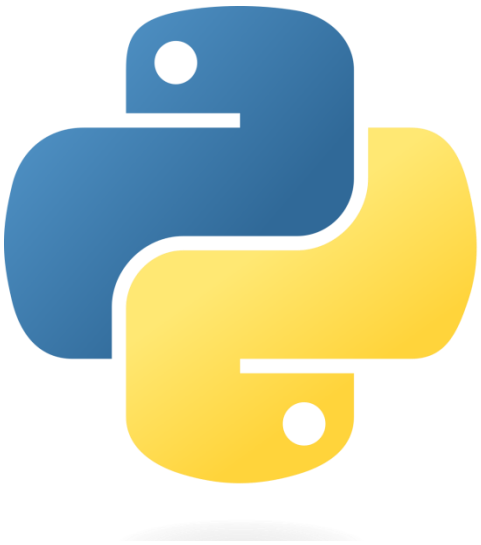

Рисунок 2.2 – Логотип мови програмування Python

Одне з найпопулярніших застосувань Python в науці про дані - це використання її для побудови та навчання нейронних мереж. Нейронні мережі - це тип алгоритму машинного навчання, який моделюється за структурою та функціями людського мозку. Вони складаються з шарів взаємопов'язаних вузлів, які обробляють і передають інформацію, що робить їх добре придатними для таких завдань, як розпізнавання зображень і мови, переклад мови і навіть гра в такі ігри, як шахи [23].

Python надає кілька бібліотек для побудови та навчання нейронних мереж, зокрема TensorFlow, Keras та PyTorch. TensorFlow - це бібліотека з відкритим вихідним кодом, розроблена Google, яка широко використовується в індустрії для великомасштабних проектів машинного навчання. Keras - це високорівневий API, який побудований на основі TensorFlow і відомий своєю простотою використання та гнучкістю. PyTorch - новіша бібліотека, яка набуває популярності завдяки своєму динамічному обчислювальному графіку, що полегшує налагодження та експерименти з різними мережевими архітектурами [24].

Окрім використання для побудови та навчання нейронних мереж, Python також популярний для попередньої обробки та візуалізації даних. Для попередньої обробки даних можна використовувати такі бібліотеки, як pandas та

numpy, для очищення, маніпулювання та перетворення даних. Для візуалізації можна використовувати такі бібліотеки, як matplotlib та seaborn, щоб створювати діаграми, графіки та графіки, які допомагають зрозуміти взаємозв'язки та закономірності в даних.

Ще одна причина, чому Python є чудовим вибором для машинного навчання - це його велика та активна спільнота. Завдяки великій кількості розробників, дослідників та аналітиків даних, які використовують Python для машинного навчання, існує безліч ресурсів, включаючи навчальні посібники, документацію та форуми. Спільнота також регулярно випускає оновлення та нові бібліотеки, що допомагає підтримувати мову в курсі останніх розробок у цій галузі [25].

Окрім машинного навчання, Python також використовується у багатьох інших сферах, таких як веб-розробка, наукові обчислення та написання сценаріїв. Така універсальність робить її чудовим вибором для розробників, які хочуть працювати над різноманітними проектами, оскільки вони можуть використовувати одну й ту саму мову для кількох завдань.

Ще однією перевагою Python є його здатність інтегруватися з іншими мовами та технологіями. Наприклад, Python можна використовувати в поєднанні з іншими мовами, такими як R та Julia, для побудови складних моделей, а також інтегрувати з базами даних та веб-додатками для створення повноцінних систем [26].

Варто також згадати, що Python став стандартом де-факто для машинного навчання в академічних та дослідницьких колах. Це пов'язано з його простотою використання, читабельністю, а також тим, що він безкоштовний і з відкритим вихідним кодом. Дослідники та студенти можуть використовувати Python для розробки та тестування своїх ідей, не турбуючись про витрати на ліцензування, а також можуть скористатися широким спектром бібліотек та інструментів, доступних в екосистемі Python.

Ще однією перевагою Python в академічному та дослідницькому контексті є можливість швидкого створення прототипів. Дослідники можуть швидко і

легко створювати і тестувати моделі, використовуючи високорівневі API, що надаються такими бібліотеками, як Keras і PyTorch. Це дозволяє їм ітерації та вдосконалення своїх ідей швидше, ніж якщо б вони використовували інші мови програмування [27].

Отже, Python - це потужна та універсальна мова програмування, яка стала основним інструментом у сфері машинного навчання. Простота використання, гнучкість та наявність потужних бібліотек роблять її чудовим вибором для науковців з даних, інженерів машинного навчання та дослідників. Незалежно від того, початківець ви чи досвідчений програміст, якщо ви хочете почати працювати з машинним навчанням, Python - чудове місце для старту.

2.5 PyGame

Рисунок 2.3 – Бібліотека мови програмування Python для створення ігор

Pygame - це бібліотека для розробки ігор на мові Python, яка надає набір модулів для створення ігор. Вона була вперше випущена в 2000 році і з тих пір стала однією з найпопулярніших бібліотек для розробки ігор на Python [28]. Pygame надає велику бібліотеку функцій для створення 2D ігор, включаючи функції для графіки, звуку та введення даних користувачем.

Pygame побудовано на основі бібліотеки SDL (Simple DirectMedia Layer), яка забезпечує низькорівневий доступ до аудіо, клавіатури, миші, джойстика та графічного обладнання. Це означає, що ігри Pygame можна запускати на різних операційних системах, включаючи Windows, macOS та Linux [29].

Вона надає повний набір функцій та можливостей, які допоможуть розробникам створювати ігри різних жанрів. Ось деякі з переваг використання PyGame:

- простота використання: PyGame розроблена так, щоб бути простою і легкою у використанні, що робить її чудовим вибором для початківців і любителів, які хочуть почати створювати ігри. Бібліотека надає інтуїтивно зрозумілий та добре задокументований API, що полегшує роботу з нею [30];
- крос-платформеність: PyGame є кросплатформенною, що означає, що ігри, розроблені з її допомогою, можуть працювати на різних операційних системах, включаючи Windows, macOS та Linux. Це робить його універсальним варіантом для розробки ігор і хорошим вибором для розробників, які хочуть охопити широку аудиторію;
- велика спільнота: PyGame має велику та активну спільноту розробників, які роблять свій внесок у бібліотеку, надаючи підтримку та ресурси, що допомагають розробникам створювати ігри. Спільнота також надає навчальні посібники, форуми та інші ресурси, які можуть допомогти розробникам вивчити всі тонкощі PyGame;
- велика бібліотека ресурсів: PyGame надає велику бібліотеку ресурсів, включаючи зображення, звукові ефекти та музику, які можна використовувати для створення ігор. Це полегшує розробникам створення ігор без необхідності витрачати багато часу на створення власних ресурсів;
- легка інтеграція з іншими інструментами: PyGame можна легко інтегрувати з іншими інструментами та бібліотеками, такими як PyOpenGL та Pyglet, що допоможе розробникам створювати більш складні ігри.

Pygame - популярний вибір для створення аркадних ігор завдяки простоті використання, великій бібліотеці ресурсів та підтримці ігор на основі спрайтів. Аркадні ігри - це тип 2D-ігор, які зазвичай відрізняються швидким розвитком подій і простою механікою геймплея [31].

Створюючи аркадні ігри за допомогою Pygame, розробники можуть використовувати широку колекцію функцій бібліотеки для створення графіки,

відтворення звуків і музики, а також обробки користувацького вводу. Це дозволяє легко створювати ігри з привабливою графікою та чуйним управлінням.

Підтримка Pygame ігор на основі спрайтів також є великою перевагою для розробки аркадних ігор. Спрайти - це графічні об'єкти, які можна переміщувати по екрану і використовувати для представлення персонажів, перешкод та інших елементів гри. Pygame надає ряд функцій для роботи зі спрайтами, що дозволяє легко створювати ігри з анімованими персонажами та динамічним ігровим процесом [32].

Крім того, Pygame надає ряд інструментів для створення ігор, які працюють з постійною частотою кадрів, що важливо для швидких аркадних ігор. Сюди входить підтримка управління ігровим циклом, оновлення ігрових елементів та управління анімацією.

Загалом, Pygame є чудовим вибором для створення аркадних ігор завдяки простоті використання, підтримці ігор на основі спрайтів та великій бібліотеці функцій для розробки ігор. Незалежно від навиків розробника ігор, Pygame допоможе з легкістю створювати захопливі та цікаві аркадні ігри.

Окрім вбудованих функцій, Pygame також добре інтегрується з іншими інструментами та бібліотеками, які можна використовувати для створення аркадних ігор. Наприклад, можна використовувати Pygame у поєднанні з іншими бібліотеками для створення більш складних ігор, таких як бібліотеки для моделювання фізики або штучного інтелекту [33].

Варто зазначити, що Pygame має активну спільноту розробників ігор, які постійно створюють нові ігри та поповнюють бібліотеку. Це означає, що для тих, хто зацікавлений у створенні аркадних ігор за допомогою Pygame, є безліч ресурсів та натхнення. Незалежно від того, чи ви тільки починаєте, чи прагнете розширити свої навички, спільнота Pygame - це чудове місце, куди можна звернутися за допомогою та підтримкою [34].

Отже, Pygame - це універсальна та потужна бібліотека для розробки ігор, яка надає широкий спектр можливостей та функцій, що допомагають розробникам створювати ігри різних жанрів. Простота використання, крос-

платформенна підтримка, велика спільнота та відкритий вихідний код роблять її популярним вибором для розробників ігор усіх рівнів.

2.6 Вибір бібліотеки глибокого навчання

Таблиця 2.1 – Порівняння різних бібліотек глибокого навчання

Особливість:	PyTorch:	TensorFlow:	Caffe:	Theano:
Динамічний обчислювальний граф	Так	Ні	Ні	Так
Попередньо зібрані модулі	Так	Так	Так	Ні
Алгоритми оптимізації	Так	Так	Так	Так
Інтеграція з іншими бібліотеками	Сильна	Сильна	Сильна	Добре
Попередньо навчені моделі	Так	Так	Так	Ні
Розподілене навчання	Так	Так.	Так	Так
Розгортання	Добре	Добре	Добре	Добре
Простий у використанні інтерфейс	Добре	Помірний	Помірний	Добре
Налагодження та візуалізація	Добре	Добре	Помірний	Добре
Продуктивність	Добре	Добре	Добре	Добре
Масштабованість	Добре	Добре	Добре	Добре
Активна спільнота	Добре	Добре	Добре	Добре
Робота зі складними моделями та нестандартними наборами даних	Добре	Добре	Добре	Добре

З точки зору динамічних обчислювальних графів, PyTorch унікальний тим, що дозволяє користувачам змінювати структуру мережі під час виконання, що полегшує реалізацію динамічних моделей, таких як RNN. Це неможливо в

TensorFlow.

Що стосується готових модулів, PyTorch і TensorFlow мають ряд готових модулів для побудови нейронних мереж, включаючи згорткові, рекурентні та лінійні шари. Caffe також має ряд готових модулів, але він менш зручний у використанні, ніж PyTorch і TensorFlow. Theano - це бібліотека нижчого рівня, яка не надає готових модулів.

Стосовно алгоритмів оптимізації, PyTorch, TensorFlow, Caffe та Theano підтримують популярні алгоритми оптимізації, такі як SGD, Adagrad та Adam.

В плані інтеграції з іншими бібліотеками, PyTorch та TensorFlow мають тісну інтеграцію з іншими популярними бібліотеками та фреймворками, такими як NumPy та Caffe2. Theano має хорошу інтеграцію з іншими бібліотеками, але не таку сильну, як PyTorch та TensorFlow.

У PyTorch і TensorFlow є ряд попередньо навчених моделей, які можна легко імпортувати і налаштовувати для конкретних випадків використання. Caffe також містить попередньо навчені моделі, але вони не настільки доступні, як попередньо навчені моделі в PyTorch і TensorFlow. Theano не містить попередньо навчених моделей.

Говорячи про розгортання, PyTorch, TensorFlow, Caffe та Theano мають інструменти та бібліотеки для розгортання моделей у виробництві. Однак інструменти розгортання PyTorch, такі як можливість конвертувати моделі в Caffe2 і ONNX, а також бібліотека PyTorch Mobile для розгортання моделей на мобільних пристроях, є особливо сильними.

Якщо говорити про простоту використання, то PyTorch надає знайомий інтерфейс Python для написання та запуску коду, що робить його простим у використанні для тих, хто вже знайомий з мовою програмування Python. TensorFlow також зручний у використанні, але він не такий інтуїтивно зрозумілий, як PyTorch, для тих, хто тільки починає займатися глибоким навчанням. Caffe і Theano - це бібліотеки нижчого рівня, які менш зручні у використанні, ніж PyTorch і TensorFlow.

З точки зору відладки та візуалізації, PyTorch, TensorFlow, Caffe та Theano

включають в себе інструменти для відладки та візуалізації [35, 36, 37, 38].

Таблиця 2.2 – Підсумок ключових особливостей PyTorch

Особливості	Опис
Динамічний обчислювальний графік	дозволяє створювати динамічні обчислювальні графіки, тобто користувач може змінювати графік на льоту, що забезпечує більшу гнучкість і полегшує налагодження.
Простий у використанні інтерфейс	має зручний інтерфейс, що полегшує початківцям початок роботи з глибоким навчанням.
Тісна інтеграція з іншими бібліотеками	добре інтегрується з іншими бібліотеками, що дозволяє легко впроваджувати більш просунуті методи.
Хороші можливості розгортання	пропонує хороші можливості для розгортання моделей у виробництві, включаючи розгортання як на CPU, так і на GPU.
Хороші інструменти налагодження та візуалізації	надає хороші інструменти для налагодження та візуалізації моделей, допомагаючи в процесі розробки та навчання.
Хороша продуктивність	має хорошу продуктивність, що дозволяє швидко навчати моделі та робити висновки.
Хороша масштабованість	масштабується, що дозволяє проводити навчання на декількох графічних процесорах і на декількох вузлах.
Доступні готові модулі та попередньо навчені моделі	має заздалегідь побудовані модулі та попередньо навчені моделі, що дозволяє прискорити розробку та експерименти.
Хороші алгоритми оптимізації	надає різноманітні алгоритми оптимізації, що полегшує пошук найкращого варіанту для конкретної задачі.
Сильна підтримка спільноти	має сильну спільноту, багато розробників роблять свій внесок у його розвиток та підтримку

PyTorch - це бібліотека машинного навчання з відкритим вихідним кодом для Python, заснована на бібліотеці Torch. Вона була розроблена лабораторією AI

Research компанії Facebook і випущена у 2017 році. Вона надає високорівневий інтерфейс для роботи з динамічними обчислювальними графами, які використовуються для побудови та навчання нейронних мереж [39].

PyTorch розроблений гнучко, що дозволяє користувачам визначати та запускати свої обчислення як на CPU, так і на GPU. Він надає широкий спектр функціональних можливостей, включаючи тензорні обчислення, глибокі нейронні мережі та алгоритми оптимізації.

Однією з ключових особливостей PyTorch є його динамічний обчислювальний граф, який дозволяє користувачам змінювати структуру мережі під час виконання. Це полегшує реалізацію динамічних моделей, таких як рекурентні нейронні мережі, а також забезпечує більшу гнучкість у налагодженні та дослідженні моделей [40].

Крім того, PyTorch надає кілька готових модулів для побудови нейронних мереж, таких як згорткові, рекурентні та лінійні шари. Він також включає підтримку популярних алгоритмів оптимізації, включаючи стохастичний градієнтний спуск (SGD), Adagrad та Adam.

Ще однією перевагою PyTorch є його тісна інтеграція з іншими популярними бібліотеками та фреймворками, такими як NumPy, Caffe2 та TensorFlow, що полегшує його використання у різноманітних проектах та дослідженнях.

PyTorch також надає велику кількість попередньо навчених моделей, так званих torchvision-моделей, які можна легко імпортувати і налаштовувати для конкретних випадків використання, заощаджуючи час і ресурси користувача. PyTorch також має велику спільноту, яка постійно зростає, з великою кількістю ресурсів та навчальних посібників, доступних для користувачів [41].

PyTorch також підтримує розподілене навчання, дозволяючи користувачам навчати моделі на декількох графічних процесорах або навіть на декількох машинах. Це особливо корисно для навчання великих моделей, оскільки дозволяє використовувати кілька графічних процесорів, щоб значно пришвидшити процес навчання.

З точки зору розгортання, PyTorch надає ряд інструментів і бібліотек для розгортання моделей у виробництві. Сюди входить підтримка конвертації моделей у Caffe2 та ONNX, а також бібліотека для розгортання моделей на мобільних пристроях, яка називається PyTorch Mobile [42].

Загалом, PyTorch - це потужна та гнучка бібліотека для побудови, навчання та розгортання моделей машинного навчання. Завдяки динамічному обчислювальному графіку, простим у використанні готовим модулям та потужній підтримці спільноти, вона є популярним вибором серед дослідників та практиків у галузі глибокого навчання.

Висновки до розділу 2

Отже, нейронні мережі є потужним інструментом машинного навчання, здатним вивчати складні взаємозв'язки між входами та виходами. Вони складаються з взаємопов'язаних штучних нейронів, які обробляють і передають інформацію, а сила зв'язків між цими нейронами регулюється в процесі навчання, щоб мінімізувати різницю між прогнозами мережі та фактичними результатами.

Відповідно, навчання з підкріпленням є потужним інструментом для прийняття рішень і розв'язання проблем, але воно також створює багато проблем, які потребують вирішення. Тим не менш, останні досягнення в цій галузі показують, що воно має великий потенціал для майбутніх застосувань у широкому спектрі областей.

DQN - це потужний алгоритм навчання з підкріпленням, який поєднує в собі глибокі нейронні мережі та Q-Learning для обробки складних просторів станів і підвищення стабільності навчання. Його використання відтворення досвіду та фіксованих цілей довело свою ефективність у багатьох додатках, і він залишається популярним підходом для вирішення проблем у навчанні з підкріпленням.

Python - потужна та багатофункціональна мова програмування, яка стала основним інструментом у сфері машинного навчання. Простота використання,

гнучкість та потужні бібліотеки роблять її чудовим вибором для науковців з даних, інженерів машинного навчання та дослідників. Незалежно від того, початківець ви чи досвідчений програміст, якщо ви хочете почати працювати з машинним навчанням, Python - чудове місце для старту.

Pygame - це універсальна і ефективна бібліотека для розробки ігор, яка надає широкий спектр можливостей і функцій, що допомагають розробникам створювати ігри різних жанрів. Простота використання, крос-платформенна підтримка, велика спільнота та відкритий вихідний код роблять її популярним вибором для розробників ігор усіх рівнів.

PyTorch - це сучасна та гнучка бібліотека для побудови, навчання та розгортання моделей машинного навчання. Завдяки динамічному графіку обчислень, зручним вбудованим модулям та потужній підтримці спільноти, вона є популярним вибором для дослідників та практиків у галузі машинного навчання.

3 МОДЕЛЮВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ В ЖАНРІ "АРКАДА" ТА AI ДЛЯ КЕРУВАННЯ ПЕРСОНАЖЕМ

3.1 Створення гри

Структура гри поділяється на три основні частини:

- папка media, яка містить усі зображення та звуки, що використовуються у грі;
- каталог модулів, який містить усі класи та інтерфейси, що використовуються у грі;
- файли config, game та objects, які містять конфігурацію гри, ігрову логіку та цикл гри, а також визначення об'єктів відповідно.

В папці media знаходяться дві папки:

- images;
- sounds,

В яких знаходяться всі спрайти, зображення та звуки відповідно.

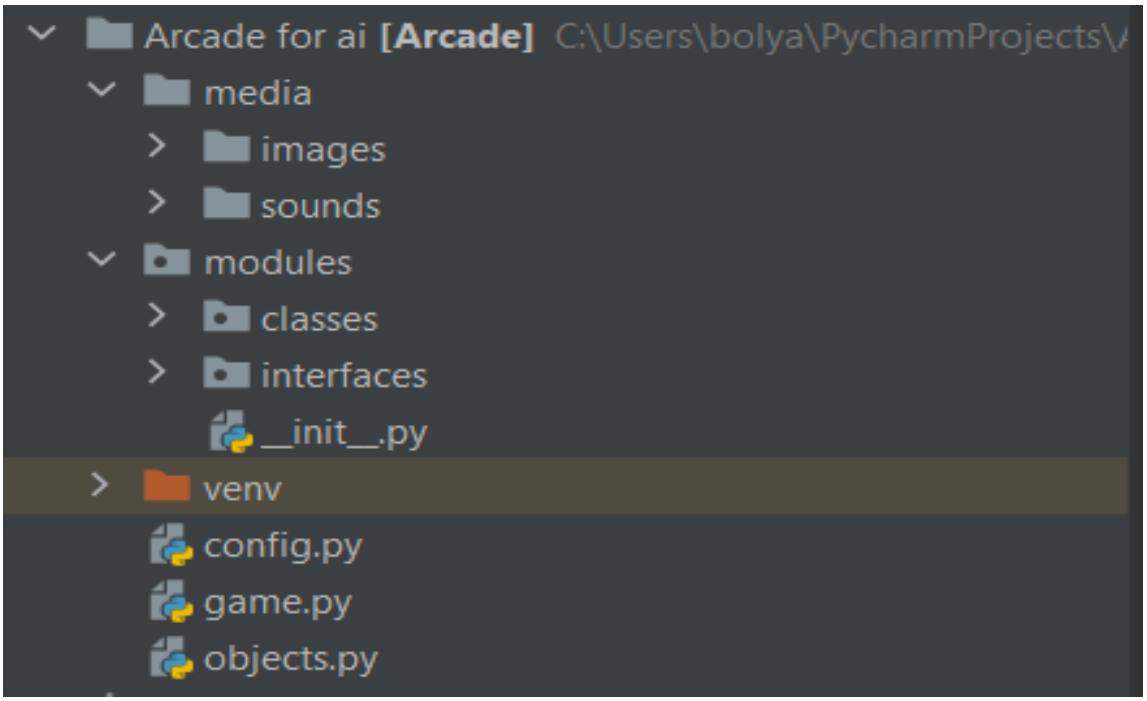

Рисунок 3.1 – Структура гри

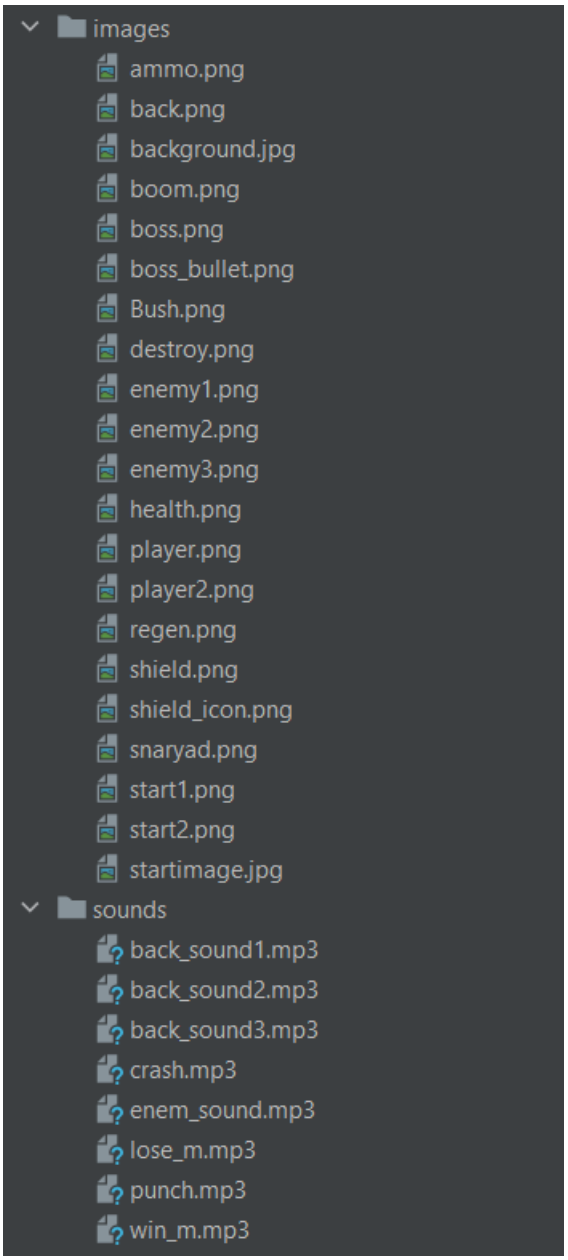

Рисунок 3.2 – Структура папки media

Тека "modules", містить різні модулі Python (файли з розширенням .py), кожен з яких визначає класи та/або функції, що використовуються у грі. Деякі з цих класів і функцій пов'язані з ігровими об'єктами, такими як персонаж гравця, вороги або предмети, тоді як інші пов'язані з загальною функціональністю гри, наприклад, обробкою даних, введених користувачем. Деякі з цих модулів визначають інтерфейси, які використовуються іншими частинами гри, або визначають утилітарні функції, які використовуються в усій грі.

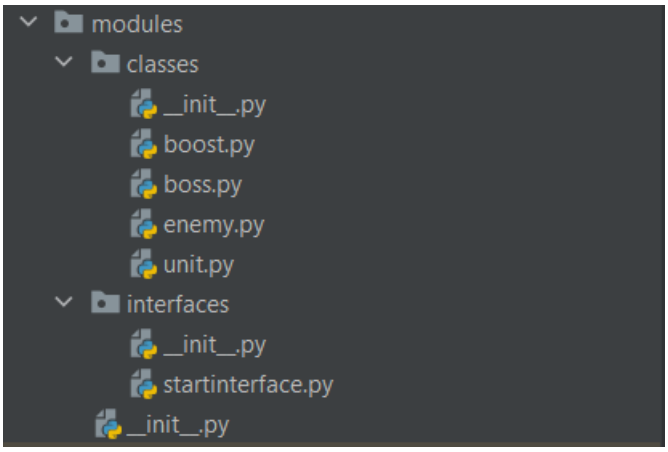

Рисунок 3.3 – Структура теки modules

Тека modules містить в собі два основні каталоги:

- classes;
- interfaces.

перший з яких містить в собі всі ігрові класи, а другий за зовнішній вигляд гри під час запуску застосунку.

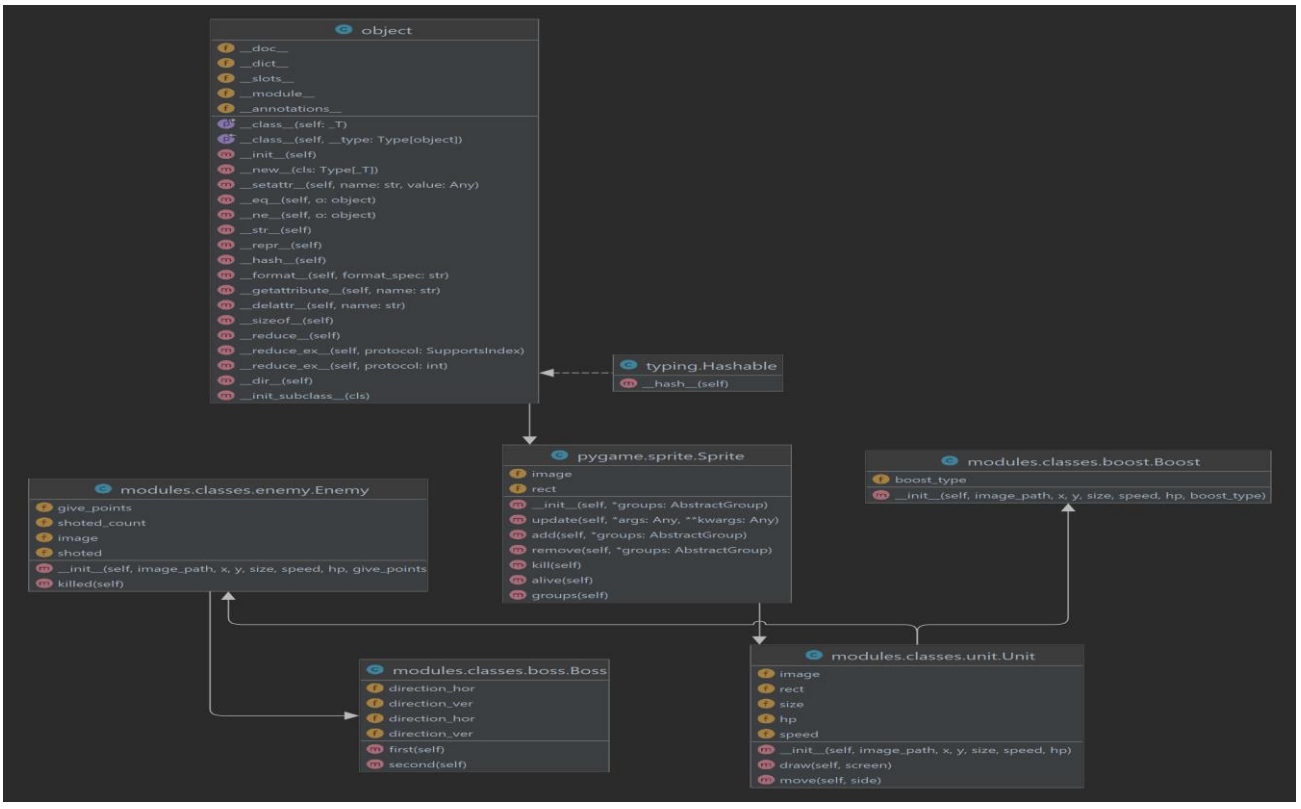

Рисунок 3.4 – Взаємозв’язок між класами гри

Клас Unit, успадковує клас sprite.Sprite з бібліотеки pygame. Він має декілька методів та параметрів:

- __init__: Цей метод є конструктором класу і викликається при створенні нового об'єкту цього класу;
- image_path: Це шлях до файлу зображення, який буде використано для модуля;
- x, y: Координати x та y розташування елемента;
- size: Розмір юніта в пікселях;
- speed: швидкість, з якою рухається юніт;
- hp: Хіт-поінтів юніта;
- draw: Цей метод отримує параметр екрану і виводить зображення юніта на екран у позиціях x та y, вказаних в атрибуті rect;
- move: Цей метод отримує параметр side і переміщує юніта вгору або вниз в залежності від значення side. Якщо параметр side дорівнює "ВГОРУ", то позиція по осі y зменшується на швидкість, а якщо параметр side дорівнює "ВНИЗ", то позиція по осі y збільшується на швидкість.

Клас Unit є базовим для всіх інших ігрових об'єктів. Він має властивості для зображення, розміру, положення, швидкості та хітів об'єкта. Клас також має методи для малювання об'єкта на екрані та переміщення його у визначеному напрямку.

Клас Enemy є підкласом класу Unit і додає властивості та методи, специфічні для ворогів у грі. Він має властивість відстежувати очки, які гравець заробляє, вбиваючи ворога, та іншу властивість, яка відстежує, чи був ворог поранений. Клас також має метод для оновлення зовнішнього вигляду ворога, коли його вбито.

Клас Boss є підкласом класу Enemy і додає властивості та методи, характерні для ворогів-босів у грі. Він має властивості для відстеження напрямку руху боса по горизонтальній і вертикальній осях і методи для переміщення боса в цих напрямках.

Клас Boost є підкласом класу Unit і додає властивості та методи, специфічні

для бонусів у грі. Він має властивість відслідковувати тип посилення, яке надає бонус.

В файлі game.py описано клас самої гри. Клас Game - це клас, який використовується для керування грою та її різними компонентами. Він використовує бібліотеку Pygame для створення ігрового екрану, а також модулі objects та config для керування об'єктами та конфігураціями. Клас зчислення Direction використовується для зберігання напрямку руху гравця.

Метод __init__ використовується для ініціалізації стану гри, включаючи екран, підписи, хронометраж та ігрові компоненти, такі як гравець, вороги, серця, постріли та бонуси. Метод встановлює значення за замовчуванням для позиції гравця, кількості ворогів, рівня складності та результату гри. Він також генерує початкову кількість очок здоров'я гравця.

Метод get_enemy_coordinates використовується для отримання координат x та y кожного ворога у грі.

Метод reset використовується для скидання гри до початкового стану, включаючи позицію гравця, кількість ворогів, рівень складності та результат гри.

Клас має різні змінні та лічильники, які використовуються для керування станом та поведінкою гри, такі як FPS, boss_reload, enem_count, koeficient, lines, points, game, win та інші.

У коді визначено метод з іменем play_step, який є частиною гри. Метод отримує один аргумент move, який використовується для визначення переміщення гравця у грі.

Метод спочатку обчислює час, що минув з моменту ініціації гри, використовуючи метод pygame.time.get_ticks() та self.init_time. Потім він створює декілька текстових міток, таких як "Час", "Очки" та їх відповідні значення, використовуючи метод font2.render. Метод також збільшує декілька лічильних змінних, таких як self.NUR_count, self.NBR_count та self.fps_fiks_count. Змінна винагороди також ініціалізується 0.

Далі метод перевіряє, чи пройшло 20 секунд і чи встановлено self.check_one_boost у True. Якщо так, то викликається функція boost_generate з

аргументом self.lines. Якщо пройшла 21 секунда, self.check_one_boost повертається до значення True. Метод також перевіряє, чи очки гравця кратні 100 і чи бос мертвий. Якщо обидві умови істинні, викликається функція release_boss, яка генерує нового боса.

Потім метод малює фон, гравця, бусти та ворогів на екрані за допомогою методу self.screen.blit. Він також переміщує посилення та ворогів вниз. Якщо бос наближається, він малює і переміщує боса і його кулі. Якщо в боса стріляють, його очки здоров'я зменшуються, а гравцеві нараховуються очки. Якщо куля боса зіштовхується з гравцем, то зменшується кількість життєвих балів гравця.

Потім метод генерує нових ворогів, якщо кількість ворогів менша за вказане число self.NUR, а бос не атакує. Він також переміщує і малює кулі гравця на екрані. Якщо буст зіштовхується з гравцем, йому присуджується додаткова винагорода, а його атрибути оновлюються відповідно до типу буста.

Потім метод оновлює позицію гравця на основі руху аргументу і малює на екрані текстові мітки та точки влучень гравця. Нарешті, метод повертає винагороду та стан гри у вигляді кортежу.

Якщо ворога підстрелили, код збільшує рахунок для видалення ворога та запускає анімацію знищення. Якщо лічільник досягає 5, ворог видаляється зі списку.

Після цього код перевіряє зіткнення ворога з будь-якою з куль у списку "bullets". Якщо виявлено зіткнення, код видаляє кулю, зменшує здоров'я ворога і додає очки гравцеві. Якщо здоров'я ворога досягає 0, він видаляється зі списку.

Нарешті, код перевіряє, чи досяг ворог нижньої межі ігрового екрану. Якщо так, то він зменшує здоров'я гравця, а ворога видаляє зі списку. Якщо у гравця є захисний щит, його здоров'я не зменшується.

Код також перевіряє зіткнення між гравцем і ворогом. Якщо зіткнення виявлено, здоров'я гравця зменшується, а ворог видаляється зі списку. Якщо у гравця є захисний щит, його здоров'я не зменшується.

Таблиця 3.1 – Зображення дій, які виконує функція play_step

Дія	**Опис**
Обчислити поточний час, що минув у грі	Відображення часу, що минув, у вигляді мітки та тексту на ігровому екрані
Відстеження кількість ігрових кроків, а також кількість об'єктів підсилення та босів	Ведення обліку ігрових кроків і кількості об'єктів підсилення та босів
Перевірка, чи настав час генерувати буст	Генерація нового буста, якщо настав час створити новий буст
Перевірка, чи очки гравця кратні 100	Породити боса, якщо кількість очок гравця кратна 100
Перемальовка ігрового екрану	Перемальовування персонажа, підсилень, ворогів і босів на ігровому екрані
Якщо бос зараз у грі, оновити його стан	Оновлення стану боса з рухом, пострілами та отриманою шкодою
Регулярна генерація нових ворогів	Генерація нових ворогів в залежності від складності гри
Оновлення куль	Оновлення руху та зіткнень усіх куль у грі
Оновлення стану всіх об'єктів підсилення у грі	Оновлення руху та зіткнення всіх буст-об'єктів з гравцем
Віднімання здоров'я гравця, якщо він зіткнеться з кулею боса, або іншими перешкодами	Налаштування рахунку та відображення здоров’я гравця, якщо воно віднімається через зіткнення з кулею боса.

3.2 Імплементація бота

model.py та agent.py - це два файли програми, яка реалізує ШІ для аркадної гри з використанням навчання з підкріпленням. model.py містить реалізацію

моделі ШІ, включаючи архітектуру нейронної мережі та процес навчання. З іншого боку, agent.py містить реалізацію ШІ-агента, включаючи логіку дій у грі та взаємодію з навколишнім середовищем. ШІ-модель навчається на основі даних, отриманих під час взаємодії ШІ-агента з грою, а агент використовує навчену модель для прийняття рішень у реальному часі під час ігрового процесу. Разом ці два файли складають систему штучного інтелекту для гри в жанрі “Аркада”.

model.py є PyTorch-реалізацією алгоритму глибокої Q-мережі (DQN), який є різновидом алгоритму навчання з підкріпленням, що використовується для навчання ШІ-агентів грати у відеоігри.

Він містить два основні класи:

- Linear_QNet: Цей клас представляє модель глибокої Q-мережі, яка є нейронною мережею з двома повністю пов'язаними шарами (fc). Перший шар використовує функцію активації ReLU для введення нелінійності в модель, тоді як другий шар виводить прогнозовані значення Q для кожної дії. Метод forward приймає вхідний стан і повертає прогнозовані значення якості;
- QTrainer: Цей клас представляє нейронну мережу, яка відповідає за оновлення параметрів моделі в процесі навчання. Він приймає модель, швидкість навчання (lr) та коефіцієнт дисконтування (gamma) як аргументи. Клас використовує оптимізатор Адама для мінімізації середньоквадратичної похибки (MSE) між цільовими значеннями добротності та прогнозованими значеннями добротності моделі. Цільові значення Q обчислюються за допомогою рівняння Беллмана, яке врівноважує негайну винагороду з передбачуваною майбутньою винагородою. Метод train_step виконує один навчальний крок і оновлює параметри моделі на основі наданої інформації про стан, дію, винагороду, наступний стан і виконану роботу.

Модель має два лінійних шари, перший з вхідним розміром "input_size" та прихованим розміром "hidden_size", а другий з прихованим розміром "hidden_size" та вихідним розміром "output_size".

```
def __init__(self, input_size, hidden_size, output_size):
    super().__init__()
    self.linear1 = nn.Linear(input_size, hidden_size)
    self.linear2 = nn.Linear(hidden_size, output_size)
```

Рисунок 3.5 – Код конструктору класу Linear_QNet

Метод forward отримує єдиний аргумент "x", який є входом у мережу. Вхід спочатку проходить через перший лінійний шар, де до виходу застосовується функція активації ReLU. Потім результат пропускається через другий лінійний шар, щоб отримати кінцевий вихід мережі.

```
def forward(self, x):
    x = F.relu(self.linear1(x))
    x = self.linear2(x)
    return x
```

Рисунок 3.6 – Код методу forward класу Linear_QNet

Модель має метод "save", який приймає необов'язковий аргумент "ім'я_файлу". Метод створює папку з назвою "model" у поточному каталозі, якщо її не існує, і зберігає поточний стан мережі у вигляді файлу контрольних точок моделі PyTorch із заданим ім'ям у папці "model". За замовчуванням файл має назву "model.pth".

```
def save(self, file_name='model.pth'):
    model_folder_path = './model'
    if not os.path.exists(model_folder_path):
        os.makedirs(model_folder_path)

    file_name = os.path.join(model_folder_path, file_name)
    torch.save(self.state_dict(), file_name)
```

Рисунок 3.7 – Код методу save класу Linear_QNet

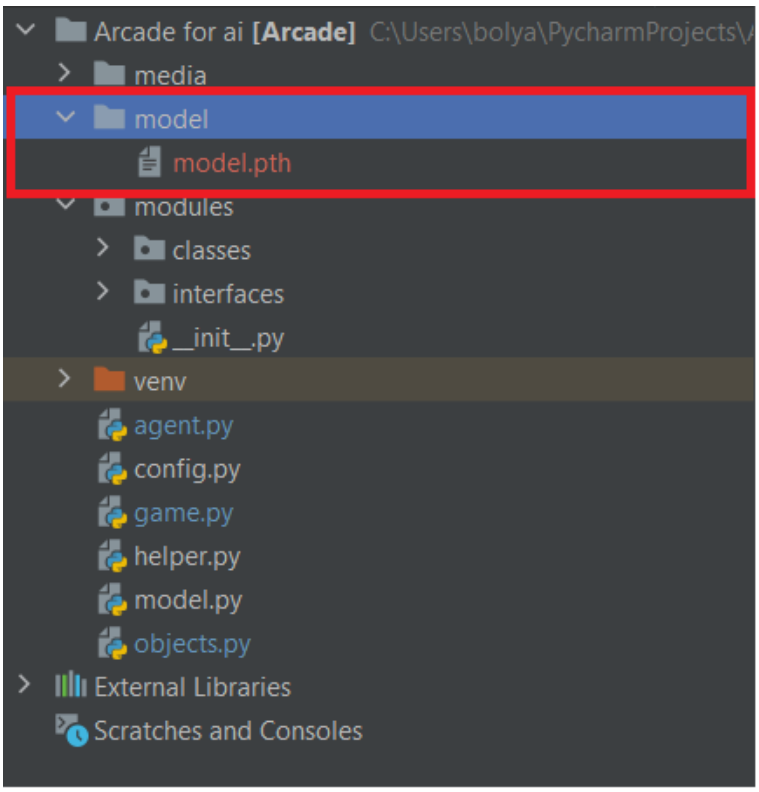

Рисунок 3.8 – Місце зберігання поточного стану мережі у вигляді файлу контрольних точок

Клас QTrainer - це тренер для навчання з підкріпленням (RL) з використанням Q-Learning. Він складається з моделі нейронної мережі, швидкості навчання lr та коефіцієнта дисконтування gamma.

Ключові елементи класу QTrainer:

- lr: швидкість навчання оптимізатора;
- gamma: коефіцієнт дисконтування, який використовується для обчислення нового значення Q на основі наступного прогнозованого значення Q;
- model: екземпляр моделі PyTorch, що представляє Q-мережу;
- optimizer: оптимізатор, який використовується для оновлення параметрів моделі. Це екземпляр torch.optim.Adam зі швидкістю навчання lr;
- criterion: функція втрат, яка використовується для обчислення різниці між цільовими значеннями Q та прогнозованими значеннями Q. Екземпляр nn.MSELoss;

- train_step: метод, який навчає мережу Q, обчислюючи нове значення Q, оновлюючи цільове значення Q, обчислюючи втрати та зворотне розповсюдження градієнтів. Він отримує п'ять вхідних даних: стан, дію, винагороду, наступний_стан і done.

Метод __init__ встановлює атрибути класу. Він встановлює швидкість навчання, коефіцієнт дисконтування та модель, яку потрібно навчити, створює оптимізатор Адама зі швидкістю навчання та встановлює критерій втрати середньоквадратичної помилки.

```
class QTrainer:
    Mykyta Boliak
    def __init__(self, model, lr, gamma):
        self.lr = lr
        self.gamma = gamma
        self.model = model
        self.optimizer = optim.Adam(model.parameters(), lr=self.lr)
        self.criterion = nn.MSELoss()
```

Рисунок 3.9 – Код конструктору класу QTrainer

Метод train_step навчає модель за один крок алгоритму Q-Learning. Він приймає на вхід поточний стан, дію, винагороду, наступний стан і прапорець, який вказує, чи завершився епізод. Вхідні дані спочатку перетворюються на тензори, і якщо тензор стану має лише один вимір, він змінюється, щоб мати розмірність одиниці.

```
Mykyta Boliak
def train_step(self, state, action, reward, next_state, done):
    state = torch.tensor(state, dtype=torch.float)
    next_state = torch.tensor(next_state, dtype=torch.float)
    action = torch.tensor(action, dtype=torch.long)
    reward = torch.tensor(reward, dtype=torch.float)
    # (n, x)

    if len(state.shape) == 1:
        # (1, x)
        state = torch.unsqueeze(state, 0)
        next_state = torch.unsqueeze(next_state, 0)
        action = torch.unsqueeze(action, 0)
        reward = torch.unsqueeze(reward, 0)
        done = (done,)

    # 1: predicted Q values with current state
    pred = self.model(state)

    target = pred.clone()
    for idx in range(len(done)):
        Q_new = reward[idx]
        if not done[idx]:
            Q_new = reward[idx] + self.gamma * torch.max(self.model(next_state[idx]))

        target[idx][torch.argmax(action[idx]).item()] = Q_new
```

Рисунок 3.10 – Код методу train_step класу QTrainer(1)

Потім модель генерує прогнозовані значення Q для поточного стану. Після цього обчислюються цільові значення Q, враховуючи отриману винагороду та максимальне значення Q для наступного стану, дисконтоване на гамма-фактор. Якщо епізод не завершився, то нове цільове значення якості дорівнює отриманій винагороді плюс дисконтоване максимальне значення якості для наступного стану. Якщо епізод завершився, то цільове значення Q дорівнює отриманій винагороді.

Згодом оптимізатор виконує один крок градієнтного спуску, використовуючи середньоквадратичну похибку між цільовим і прогнозованим значеннями якості як втрату. Градієнти обчислюються за допомогою методу backward() тензора втрат, а оптимізатор оновлює параметри моделі за допомогою

методу step().

```
    self.optimizer.zero_grad()
    loss = self.criterion(target, pred)
    loss.backward()

    self.optimizer.step()
```

Рисунок 3.11 – Код методу train_step класу QTrainer(2)

В файлі agent.py реалізовано ШІ-агента для керування персонажем. Код реалізує ШІ-агента для ведення гри. Агент використовує навчання з підкріпленням за допомогою алгоритму Q-навчання, щоб навчитися найкращим діям у кожному стані гри.

Код починається з імпорту необхідних бібліотек, включаючи torch для машинного навчання, numpy для чисельних обчислень та deque для структури даних двосторонньої черги. Константи MAX_MEMORY та BATCH_SIZE використовуються для керування розміром пам'яті агента, а константа LR визначає швидкість навчання Q-мережі.

```
import torch
import random
import numpy as np
from collections import deque
from game import *
from model import Linear_QNet, QTrainer
from helper import plot

MAX_MEMORY = 100_000
BATCH_SIZE = 1000
LR = 0.001
```

Рисунок 3.12 – Імпорт бібліотек та створення констант для класу Agent

Клас Agent визначено для реалізації агента ШІ. Він має декілька важливих атрибутів та методів. Атрибут n_games відстежує кількість ігор, зіграних агентом, epsilon - параметр випадковості, який контролює компроміс між дослідженням та експлуатацією, gamma - ставка дисконтування, яка визначає важливість майбутніх винагород, memory - дескриптор, який зберігає історію дій та винагород агента, model - Q-мережа, яку агент використовує для оцінки оптимальної дії в кожному стані, а trainer - об'єкт, який навчає Q-мережу.

```
class Agent:

    Mykyta Boliak *
    def __init__(self):
        self.n_games = 0
        self.epsilon = 0  # randomness
        self.gamma = 0.9  # discount rate
        self.memory = deque(maxlen=MAX_MEMORY)  # popleft()
        self.model = Linear_QNet(9, 256, 2)
        self.trainer = QTrainer(self.model, lr=LR, gamma=self.gamma)
        self.enems = []
```

Рисунок 3.13 – Код конструктору класу Agent

Метод get_state отримує на вхід поточну гру і повертає представлення стану гри, яке є 9-вимірним масивом, що містить інформацію про позицію гравця, позицію найближчого ворога, різницю в координаті x між гравцем і найближчим ворогом, здоров'я гравця, а також позиції бонусів і ворогів.

```
    Mykyta Boliak *
    def get_state(self, game):
        nearest_enemy_y = 0
        nearest_enemy_x = 200

        for enemy in game.enemies:
            if enemy.rect.y > nearest_enemy_y:
                nearest_enemy_x = enemy.rect.x
                nearest_enemy_y = enemy.rect.y
        current_enemies = game.get_enemy_coordinates(game.enemies)
```

Рисунок 3.14 – Код методу get_state класу Agent(1)

```
for enemy in current_enemies:
    found = False
    for i, e in enumerate(self.enems):
        if e[0] == enemy[0] and e[1] == enemy[1]:
            # Update the coordinates of an existing enemy
            self.enems[i] = enemy
            found = True
            break
    if not found:
        # Add a new enemy to the list
        self.enems.append(enemy)

for e in self.enems:
    print("Enemy coord x:". e[0], " y: ", e[1])
boosti = []
for boost in boosts:
    boosti.append((boost.rect.x, boost.rect.y))
```

Рисунок 3.15 – Код методу get_state класу Agent(2)

```
    state = [
        game.player_x,
        game.player_y,
        nearest_enemy_x,
        nearest_enemy_y,

        game.player_x < nearest_enemy_x,
        game.player_x > nearest_enemy_x,
        player.hp,
        boosti,
        self.enems,


    ]

    return np.array(state, dtype=float)
```

Рисунок 3.16 – Код методу get_state класу Agent(3)

Метод remember додає кортеж (стан, дія, нагорода, наступний стан, чи

закінчено) до деки пам'яті. train_long_memory тренує Q-мережу на випадковій вибірці пам'яті, тоді як train_short_memory тренує Q-мережу на одному кортежі (стан, дія, нагорода, наступний стан, чи закінчено).

```
Mykyta Boliak
def remember(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))  # popleft if MAX_MEMORY is reached
```

Рисунок 3.17 – Код методу get_state класу Agent(3)

Функція "train_long_memory" тренує модель, використовуючи вибірку досвіду з буфера пам'яті. Буфер пам'яті зберігає досвід агента у вигляді кортежів (state, action, reward, next_state, done). Якщо буфер пам'яті містить більше, ніж "BATCH_SIZE" досвідів, береться випадкова вибірка з "BATCH_SIZE" досвідів. Якщо у буфері пам'яті менше експериментів, ніж "BATCH_SIZE", то використовуються всі експерименти у буфері. Потім вибірка експериментів передається у функцію "train_step" об'єкта trainer, яка виконує крок навчання на моделі.

```
Mykyta Boliak
def train_long_memory(self):
    if len(self.memory) > BATCH_SIZE:
        mini_sample = random.sample(self.memory, BATCH_SIZE)  # list of tuples
    else:
        mini_sample = self.memory

    states, actions, rewards, next_states, dones = zip(*mini_sample)
    self.trainer.train_step(states, actions, rewards, next_states, dones)
    # for state, action, reward, nexrt_state, done in mini_sample:
    #     self.trainer.train_step(state, action, reward, next_state, done)
```

Рисунок 3.18 – Код методу train_long_memory класу Agent

Функція "train_short_memory" тренує модель, використовуючи один кортеж

досвіду (стан, дія, винагорода, наступний_стан, виконано), переданий як аргумент функції. Цей досвід негайно передається функції "train_step" об'єкта trainer для навчання.

```
Mykyta Boliak
def train_short_memory(self, state, action, reward, next_state, done):
    self.trainer.train_step(state, action, reward, next_state, done)
```

Рисунок 3.19 – Код методу train_short_memory класу Agent

Метод get_action отримує на вхід поточний стан і повертає дію, яку повинен виконати агент. Метод використовує епсілон-жадібне дослідження, що означає, що з ймовірністю епсілон виконується випадкова дія, а з ймовірністю 1 - епсілон, дія обирається Q-мережею.

```
Mykyta Boliak *
def get_action(self, state):
    # random moves: tradeoff exploration / exploitation
    self.epsilon = 80 - self.n_games
    final_move = [0, 0]
    if random.randint(0, 200) < self.epsilon:
        move = random.randint(0, 1)
        final_move[move] = 1
    else:
        state0 = torch.tensor(np.array(state), dtype=torch.float)
        prediction = self.model(state0)
        move = torch.argmax(prediction).item()
        final_move[move] = 1

    return final_move
```

Рисунок 3.20 – Код методу get_action класу Agent

Функція навчання призначена для запуску агента ШІ та його навчання шляхом багаторазового повторення гри. Функція ініціалізує агента та гру, потім входить у цикл, який триває доти, доки агент не досягне певного рівня

продуктивності. На кожній ітерації циклу функція отримує поточний стан гри, вибирає дію за допомогою методу get_action агента, виконує цю дію в грі, тренує агента на отриманому кортежі (стан, дія, винагорода, наступний_стан, зроблено) і додає інформацію в пам'ять. Оцінки кожної гри та середні оцінки за період гри будуються та оновлюються після кожної окремої гри.

```
def train():
    plot_scores = []
    plot_mean_scores = []
    total_score = 0
    record = 0
    agent = Agent()
    game = Game()
    while True:
        # get old state
        state_old = agent.get_state(game)

        # get move
        final_move = agent.get_action(state_old)

        # perform move and get new state
        reward, done, score = game.play_step(final_move)
        state_new = agent.get_state(game)

        # train short memory
        agent.train_short_memory(state_old, final_move, reward, state_new, done)

        # remember
        agent.remember(state_old, final_move, reward, state_new, done)
```

Рисунок 3.21 – Код методу train класу Agent(1)

```
    if done:
        # train long memory, plot result
        game.reset()
        agent.enems = []
        agent.n_games += 1
        agent.train_long_memory()

        if score > record:
            record = score
            agent.model.save()

        print('Game', agent.n_games, 'Score', score, 'Record:', record)

        plot_scores.append(score)
        total_score += score
        mean_score = total_score / agent.n_games
        plot_mean_scores.append(mean_score)
        plot(plot_scores, plot_mean_scores)
```

Рисунок 3.22 – Код методу train класу Agent(2)

3.3 Система оцінювання дій бота

Оцінювання дій в RL є важливим фактором, який визначає ефективність навчання бота. Правильне оцінювання дій може прискорити і поліпшити процес навчання, тоді як неправильне оцінювання може ускладнити або навіть перешкоджати навчанню.

Наприклад, якщо бот отримує занадто маленьку винагороду за правильні дії або занадто великий штраф за неправильні дії, він може не мати достатньої мотивації для навчання. Аналогічно, якщо бот отримує несподівану винагороду або штраф, він може приймати неоптимальні рішення.

Важливо, щоб оцінювання дій було чесним, обґрунтованим і відповідало цільовому значенню. Тільки тоді бот зможе ефективно навчатися

При створенні бота для гри була розроблена така система оцінювання:

Таблиця 3.2 – Зображення дії які можуть бути оцінені при тренуванні бота

Дія	Оцінка
Зіткнення кулі бота з ворогом	5
Підбір посилення	10
Знищення найближчого за віссю у ворога	10
Досягнення ворогом нижньої частини екрану	-10
Зіткнення бота та ворога	-10
Досягнення ботом кількості очок кратних 30	15

3.4 Результати роботи та тестування

В цьому розділі будуть наведені результати роботи даної системи та описано принцип роботи файлу helper.py, який було створено для побудови графіку та виведення консоль результатів кожної гри.

helper.py є допоміжним файлом, який містить функції та код для побудови графіків у певному форматі. Він використовується для візуалізації результатів роботи програми.

У наведеному вище файлі визначено функцію plot, яка будує графіки двох масивів даних, scores і mean_scores. Код використовує бібліотеку matplotlib для побудови графіків і бібліотеку IPython для відображення.

Рядок plt.ion() встановлює режим побудови графіків matplotlib на інтерактивний, що дозволяє оновлювати графік у реальному часі. Функція plot має два входи, scores і mean_scores, які є масивами даних для побудови графіків.

Рядки display.clear_output(wait=True) і display.display(plt.gcf()) очищають попередні результати побудови графіка і повторно відображають поточні дані. Рядок plt.clf() очищає поточну фігуру, щоб можна було побудувати нову. Рядки plt.title, plt.xlabel та plt.ylabel задають назву графіка, підписи осі x та осі y відповідно.

Рядок plt.plot(scores) відображає дані балів, а рядок plt.plot(mean_scores) - дані mean_scores. Рядок plt.ylim(ymin=0) встановлює мінімальне значення осі y на 0. Рядки plt.text(len(scores)-1, scores[-1], str(scores[-1]) та plt.text(len(mean_scores)-1, mean_scores[-1], str(mean_scores[-1])) додають текстові коментарі до графіка, виводячи останнє значення балів та середніх балів.

Рядок plt.show(block=False) показує графік, а рядок plt.pause(.1) призупиняє виконання на 0.1 секунди. Ця функція дозволяє оновлювати графік у реальному часі, коли генеруються нові дані, що робить його корисним для відстеження прогресу під час навчання або подібних процесів.

```python
import matplotlib.pyplot as plt
from IPython import display

plt.ion()

Mykyta Boliak
def plot(scores, mean_scores):
    display.clear_output(wait=True)
    display.display(plt.gcf())
    plt.clf()
    plt.title('Training...')
    plt.xlabel('Number of Games')
    plt.ylabel('Score')
    plt.plot(scores)
    plt.plot(mean_scores)
    plt.ylim(ymin=0)
    plt.text(len(scores)-1, scores[-1], str(scores[-1]))
    plt.text(len(mean_scores)-1, mean_scores[-1], str(mean_scores[-1]))
    plt.show(block=False)
    plt.pause(.1)
```

Рисунок 3.23 – Код методу plot для побудови графіку

Рисунок 3.24 – Загальний вигляд гри при старті

Під час гри можуть з’являтись посилення для персонажа, при підборі яких гравець отримує різноманітні бонуси Приклад бонусу для збільшення очок здоров’я показано на рисунку 3.25.

Рисунок 3.25 – Приклад бонусу в грі

Після вибивання всіх очок здоров'я ворога програється анімація його

знищення. Приклад наведено на рисунку 3.26.

Рисунок 3.26 – Приклад знищення ворога

"Бос" - це термін, який використовується в багатьох відеоіграх для позначення сильного або винятково складного супротивника, який є особливим етапом або завданням у грі. В даній грі можна зустріти боса після досягання певного порогу очок. Бос має 2 типи поведінки, які змінюються в залежності від кількості очок здоров'я даного супротивника. На відміну від інших супротивників, бос не намагається дойти до нижньої частини ігрового екрану, а намагається знищити персонажа своїми снарядами. Приклад вигляду боса наведено на рисунку 3.27.

Рисунок 3.27 – Приклад вигляду боса

Для того щоб запустити навчання бота, потрібно стартувати метод main() в класі Agent. Після запуску по центру з’являється вікно гри та будується графік в правій частині виробничої середи PyCharm. В консолі відображаються кількість очок отриманих за попередню гру та найкращий показник очок за всі ігри. Також

в консолі виводиться винагорода за певні дії бота.

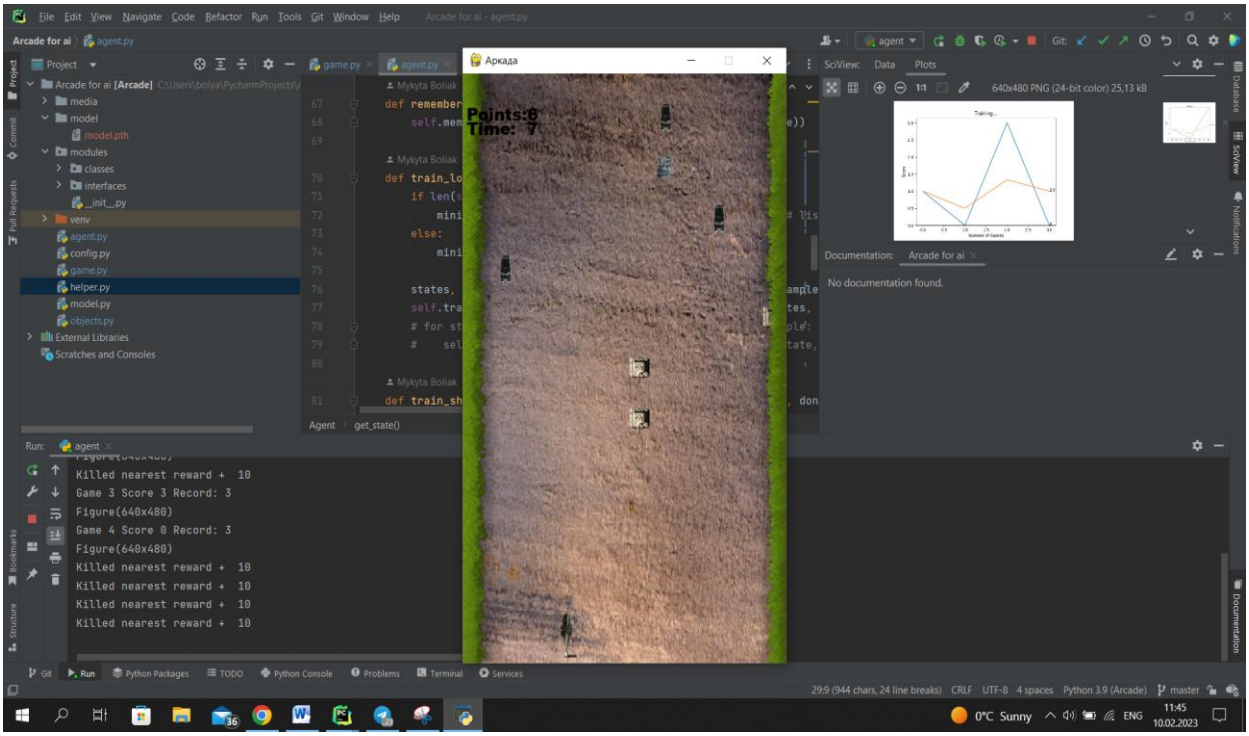

Рисунок 3.28 – Запуск навчання бота

В процесі навчання більш ніж за 200 ігр бот показав результати представлені на рисунку 3.29.

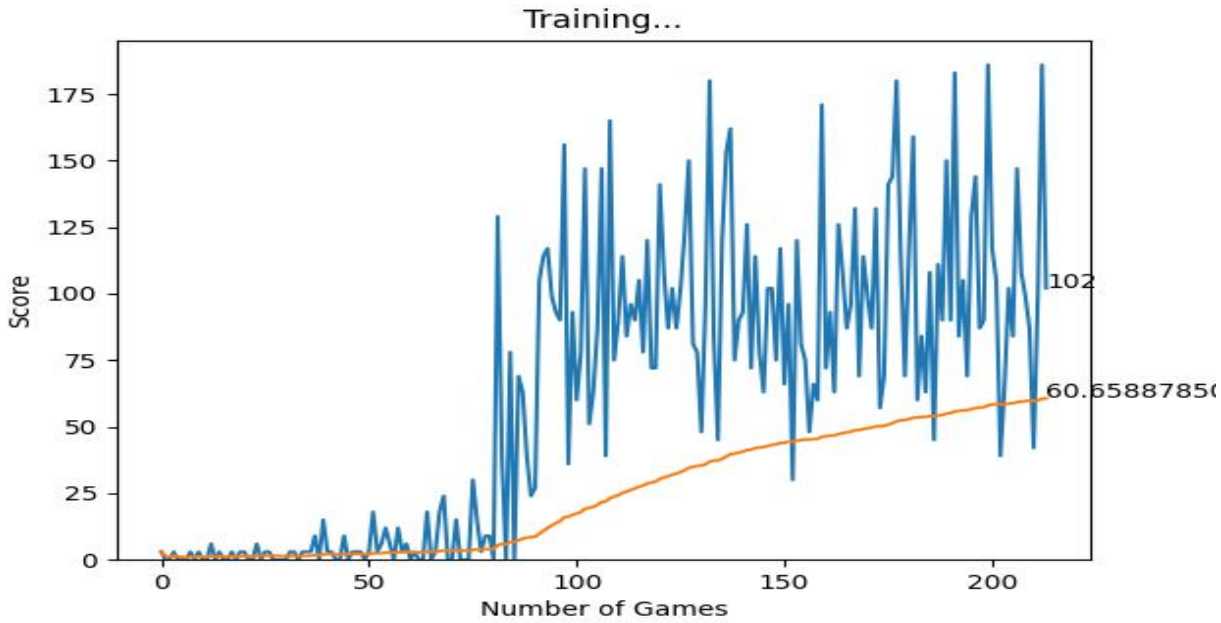

Рисунок 3.29 – Графік результату навчання бота

Висновки до розділу 3

Можна зробити висновок, що в ігровій індустрії зростає тенденція до використання штучного інтелекту та машинного навчання. Зокрема, навчання з підкріпленням дозволяє боту вчитися на власному досвіді та вдосконалювати свій ігровий процес з часом, що робить його цінним інструментом для створення більш захопливих ігрових вражень. Цей розвиток свідчить про те, що ігрова індустрія активно досліджує нові та інноваційні способи створення захопливих і розважальних ігор, і що штучний інтелект і машинне навчання відіграють дедалі більшу роль у цих зусиллях.

Було застосовано структурований та організований підхід до розробки гри та бота. Створення різних класів, таких як boost, boss, enemy і unit, а також конфігураційних файлів (config.py) і основного ігрового файлу (game.py), демонструє чіткий розподіл обов'язків і зосередженість на створенні добре продуманої ігрової архітектури. Крім того, розробка класів Linear_QNet, QTrainer та Agent для реалізації бота, а також надання графічного функціоналу у файлі helper.py свідчать про те, що було докладено значних зусиль для створення якісного та ефективного ігрового досвіду з використанням ШІ. Ці розробки свідчать про сильну прихильність до інновацій та досконалості в ігровій індустрії.

Крім того, використання навчання з підкріпленням при створенні бота свідчить про прагнення використовувати передові технології та інноваційні підходи при створенні гри. Створення бота з можливостями навчання з підкріпленням також демонструє визнання зростаючої важливості штучного інтелекту та машинного навчання в ігровій індустрії та бажання залишатися на крок попереду з точки зору технологій та інновацій.

ВИСНОВКИ

В галузі ігрових ботів та штучного інтелекту відбувається різкий розвиток завдяки методам машинного навчання, таким як глибоке навчання. Нові технології дозволяють створювати ігрових ботів, які майже не відрізняються від реальних гравців, дозволяючи гравцям відчувати себе частинами команди навіть в відсутності товаришів по грі. Нейронні мережі являються одним з ключових інструментів машинного навчання, вони складаються з взаємопов'язаних нейронів, які відтворюють функції людського мозку, та можуть вивчати складні закономірності між входами та виходами, що дозволяє ботам діяти з високою реалістичністю.

Однак, глибоке навчання та нейронні мережі відіграють важливу роль не тільки в секторі відеоігор, але також в деяких інших галузях, таких як медицина, транспорт, маркетинг та ін. Вони можуть використовуватися для аналізу великих об'ємів даних, визначення закономірностей та трендів, а також для здійснення прогнозів та автоматизації процесів.

В загалом, глибоке навчання та ігрові боти, пов'язані з ним, стають все більш вражаючими та важливими в сучасному світі. Можливості, які вони відкривають, необмежені і спрямовані на розвиток технологій, які будуть відігравати велику роль в нашому житті в майбутньому.

В розробленому проекті можна помітити широке використання принципів штучного інтелекту та машинного навчання, які дозволяють реалізувати бота, який може вчитися на власному досвіді, розвиватися та вдосконалювати свої ігрові процеси. Також, існує чітке розподілення обов'язків між класами та організований підхід до створення гри та бота, що дозволяє бачити інноваційність та професіоналізм у розробці. Таким чином, можна зробити висновок, що ігрова індустрія напрямлена на розвиток та використання штучного інтелекту та машинного навчання, та що цей проект є добрим прикладом того, як це можливо реалізувати на практиці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. gaming [Електронний ресурс]. — Режим доступу: https://www.techtarget.com/whatis/definition/gaming (дата звернення: 10.01.2023)

2. 11 positive effects of video games [Електронний ресурс]. — Режим доступу: https://gamequitters.com/positive-effects-of-video-games/ (дата звернення: 10.01.2023)

3. AI in gaming: Utilizing game bots and game marketing [Електронний ресурс]. — Режим доступу: https://www.uahero.ai/blog/ai-in-gaming-utilizing-game-bots-and-game-marketing (дата звернення: 10.01.2023)

4. Are bots in games powered by machine learning? [Електронний ресурс]. — Режим доступу: https://www.quora.com/Are-bots-in-games-powered-by-machine-learning (дата звернення: 10.01.2023)

5. The Beginner's Guide to Bots in Gaming [Електронний ресурс]. — Режим доступу: https://www.gamedesigning.org/gaming/game-bots/ (дата звернення: 10.01.2023)

6. John, P. M. Artificial Intelligence For Dummies (For Dummies (Computer/Tech)) / P. M. John, Massaron Luca. — 2., 2021. — 368 с.

7. Perry, Xiao Artificial Intelligence Programming with Python: From Zero to Hero / Xiao Perry., 2022. — 720 с.

8. Albert, C. L. Understanding Artificial Intelligence: Fundamentals and Applications / C. L. Albert, Ming,Kin Oscar, Law Iain. — : Wiley-IEEE Press, 2022. — 224 с.

9. What Was Deep Blue? [Електронний ресурс]. — Режим доступу: https://www.easytechjunkie.com/what-was-deep-blue.htm (дата звернення: 10.01.2023)

10. Bernard, Marr Artificial Intelligence in Practice: How 50 Successful Companies Used AI and Machine Learning to Solve Problems / Marr Bernard, Ward Matt., 2019. — 352 с.

11. OpenAI API [Електронний ресурс]. — Режим доступу: https://beta.openai.com/docs/guides/code/introduction (дата звернення: 10.01.2023)

12. Aggarwal C. C. Neural Networks and Deep Learning. Cham : Springer International Publishing, 2018. URL: https://doi.org/10.1007/978-3-319-94463-0 (дата звернення: 15.02.2023).
13. Bishop C. M. Pattern Recognition and Machine Learning. Springer, 2016. 758 p.
14. Kelleher J. D. Deep Learning. MIT Press, 2019. 296 с.
15. Lee W.-M. Python Machine Learning. Wiley & Sons, Incorporated, John, 2019. 320 с.
16. Murphy K. P. Machine Learning: A Probabilistic Perspective. MIT Press, 2012. 1104 p.
17. Dong H., Ding Z., Zhang S. Deep Reinforcement Learning: Fundamentals, Research and Applications. Springer, 2020. 544 с.
18. Ravichandiran S. Hands-On Reinforcement Learning with Python: Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow. Packt Publishing, 2018. 318 с.
19. Sutton R. S., Barto A. G., Bach F. Reinforcement Learning: An Introduction. MIT Press, 2018. 552 с.
20. Brown B., Zai A. Deep Reinforcement Learning in Action. Manning Publications, 2020. 325 с.
21. Habib N. Hands-On Q-Learning with Python: Practical Q-Learning with OpenAI Gym, Keras, and TensorFlow. Packt Publishing, Limited, 2019. 212 с.
22. Nakamoto P. Neural Networks and Deep Learning: Neural Networks & Deep Learning, Deep Learning, Blockchain Blueprint. Createspace Independent Publishing Platform, 2018. 152 с.
23. ANSARI H. Learn Python-3: Python Technologies. Independently Published, 2020. 307 с.
24. Matthes E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. No Starch Press, Incorporated, 2015. 560 с.
25. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming / ред.: M. Blanchette, R. Roumeliotis. O'Reilly Media, 2015. 792 с.

26. Severance D. C. R. Python for Everybody: Exploring Data in Python 3. CreateSpace Independent Publishing Platform, 2016. 244 с.

27. VanderPlas J. Python Data Science Handbook: Essential Tools for Working with Data / ред. D. Schanafelt. O'Reilly Media, 2016. 548 с.

28. Beginning Python Games Development: With PyGame. New York : Apress Media, 2015. 308 с.

29. Craven P. V. Program Arcade Games. Berkeley, CA : Apress, 2016. URL: https://doi.org/10.1007/978-1-4842-1790-0 (дата звернення: 15.02.2023).

30. Harris A. Game Programming: The L Line, The Express Line to Learning (The L Line: The Express Line To Learning). Wiley, 2007. 570 с.

31. Howse J., Paz A. R. d. Python Game Programming By Example. Packt Publishing, 2015. 230 с.

32. Kelly S. Python, PyGame and Raspberry Pi Game Development. Apress, 2016. 220 с.

33. Making Games with Python & PyGame: A guide to programming with graphics, animation, and sound. CreateSpace Independent Publishing Platform, 2012. 366 с.

34. Sweigart A. Invent Your Own Computer Games with Python. 4-те вид. San Francisco : No Starch Press, 2016. 376 с.

35. Bakker I. D. Python Deep Learning Cookbook: Over 75 practical recipes on neural network modeling, reinforcement learning, and transfer learning using Python. Packt Publishing, 2017. 330 с.

36. Geron. Hands-On machine learning with scikit-learn and tensorflow: concepts, tools, and techniques to build intelligent systems. Shroff - O'Reilly, 2017. 568 с.

37. Python Deep Learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow / I. Vasilev та ін. 2-ге вид. Packt Publishing, 2019. 386 с.

38. Raschka S., Mirjalili V. Python machine learning: machine learning and deep learning with python, scikit-learn, and tensorflow, 2nd edition. Packt Publishing, 2017. 622 с.

39. Mishra P. PyTorch recipes: a problem-solution approach. Berkeley, California : Apress, 2019. 204 с.
40. Pointer I. Beginner's guide to using pytorch for deep learning: creating and deploying deep learning applications. O'Reilly Media, Incorporated, 2019. 300 с.
41. Stevens E. E., Antiga L. L., Viehmann T. T. Deep learning with pytorch. Manning Publications Company, 2020. 520 с.
42. Subramanian V. Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Packt Publishing, 2018. 262 с.
43. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженостітрудового процесу //Охорона праці. – 2001. –№ 12. – С. 12-20.
44. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин.
45. Arkanoid, Arcade Video game by Taito (1986) [Електронний ресурс]. — Режим доступу: https://beta.openai.com/docs/guides/code/introduction (дата звернення: 11.02.2023)