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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти

на тему

“Дослідження та програмна реалізація системи штучного інтелекту для комп’ютерної гри на рушії Unity”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-21М1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Шевченко О.О.
«____» _____ 20__р.

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

м. Кропивницький

№ рядку	Формат	Позначення	Найменування	Кількість аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ВКРМ-123.22.0027.00.00.ПЗ	Пояснювальна записка	87	—	
2	А4	ВКРМ-123.22.0027.00.00.ТЗ	Технічне завдання	6	—	Додаток А
3	А4	ВКРМ-123.22.0027.00.00.РП	Робоча програма	42	—	Додаток Б
			Графічні документи			
4	А4	ВКРМ-123.22.0027.00.00.НН	Наукова новизна	1	—	
5	А4	ВКРМ-123.22.0027.00.00.Е1	Структурна схема	1	—	
6	А4	ВКРМ-123.22.0027.00.00.Е2	Функціональна схема	1	—	
7	А4	ВКРМ-123.22.0027.00.00.Д1	Діаграма процесів	1	—	
8	А4	ВКРМ-123.22.0027.00.00.Д2	Блок-схема основної програми	1	—	
9	А4	ВКРМ-123.22.0027.00.01.Д2	Блок-схема роботи підпрограми	1	—	
10	А4	ВКРМ-123.22.0027.00.00.Д3	Показники економічної ефективності	1	—	

Вим.	Арк.	№ документа	Підпис	Дата	ВКРМ-123.22.0027.00.00.ВП		
Розробив		Шевченко О.О.			Дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity	Літ.	Аркуш	Аркушів
Перевірив		Мелешко Є.В.				М		1
Н. Контр.		Гермак В.С.				ЦНТУ КІ-21М1,4		
Затв.		Смірнов О.А.						

АНОТАЦІЯ

Шевченко О.О. Дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній магістерській роботі розроблено програмне забезпечення, яке призначено для реалізації системи штучного інтелекту для комп'ютерної гри на рушії Unity.

Метою розробки є дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity.

Об'єктом дослідження є процес прийняття рішень в інтелектуальних системах шахових комп'ютерних програм.

Предметом дослідження є методи та алгоритми штучного інтелекту для знаходження вірних ходів у грі в шахи при змаганні комп'ютера проти людини, а також методи та алгоритми візуалізації гри у шахи на рушії Unity.

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

Результат роботи – програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity.

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

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

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

Програму розроблено на мові програмування C# у ігровому рушії Unity.

Ключові слова: комп'ютерна інженерія, штучний інтелект, комп'ютерна гра, ігровий рушій Unity.

ABSTRACT

Shevchenko O.O. Research and software implementation of an artificial intelligence system for a computer game on the Unity engine. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi 2022.

In this master's thesis, software designed for an artificial intelligence system for a computer game on the Unity engine.

The purpose of the development is the research and program implementation of an artificial intelligence system for a computer game on the Unity engine.

The object of research is the decision-making process in intelligent systems of chess computer programs.

The subject of research is artificial intelligence methods and algorithms for finding correct moves in a game of chess when competing against a computer, as well as methods and algorithms for visualizing a game of chess on the Unity engine.

Research methods are based on artificial intelligence methods, software development methods, object-oriented programming methods, and computer game development methods.

The result of the work is the software implementation of an artificial intelligence system for a computer game on the Unity engine.

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

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

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

The program was developed in the C# programming language in the Unity game engine.

Keywords: computer engineering, artificial intelligence, computer game, game engine Unity.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 3
ВСТУП 4
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 6
1.1 Призначення системи 6
1.2 Область застосування 6
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 8
2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми магістерської роботи 8
2.1.1 Огляд алгоритмів, які використовуються в основі роботи аналогічних існуючих систем 13
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування 18
2.3 Розгорнута постановка завдання 20
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 22
3.1 Опис функціонування системи 22
3.2 Розробка структурної схеми 30
3.3 Розробка функціональної схеми 32
3.4 Розробка діаграми процесів 33
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ 35
4.1 Блок-схеми та опис алгоритмів функціонування системи 35
4.2 Захист розробленого програмного забезпечення 45

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ 47
6 НАУКОВА НОВИЗНА 49
7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ 50
7.1 Техніко-економічне обґрунтування теми магістерської роботи 50
7.2 Розрахунок трудомісткості розробки програмної продукції 52
7.3 Визначення чисельності виконавців і планового фонду зарплати 54
7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника 58
7.5 Визначення собівартості розробки та ціни програмної продукції 62
7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції 66
7.7 Визначення експлуатаційних витрат 66
7.8 Визначення економічної ефективності програмної продукції 68
7.9 Висновки 70
8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ 71
8.1 Вступ 71
8.2. Шкідливі і небезпечні фактори при роботі з комп’ютером 72
8.3. Аналіз умов праці на робочому місці програміста 74
8.4. Розрахункова частина 77
8.5 Висновки до розділу 79
9 ОСНОВНІ ВИСНОВКИ 80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

AI – штучний інтелект;

DFS – алгоритм Пошуку в глибину;

GPU – графічний процесор;

GUI – графічний інтерфейс користувача;

MCTS – алгоритм пошуку по дереву Монте Карло;

NNUE – ефективно оновлювана нейронна мережа;

PVS – алгоритм пошуку основних відхилень;

Unity – ігровий рушій;

ЕОМ – електронно-обчислювальна машина;

ОС – операційна система;

ПК – персональний комп’ютер.

ВСТУП

Актуальність теми. Дослідження у сфері штучного інтелекту не втрачають своєї актуальності. Інтелектуальні комп'ютерні системи починають використовуватися у найрізноманітніших галузях діяльності людини, не виключенням стали і комп'ютерні ігри. Використання штучного інтелекту в комп'ютерних іграх потребує особливого підходу, адже комп'ютерний суперник має стати цікавим, відносно складним та непередбачуваним для гравця. Метою даної роботи є програмна реалізація штучного інтелекту для гри в шахи, зі зручним та мінімалістичним додатком.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на ігровому рушії Unity.

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

– Дослідження існуючих систем штучного інтелекту для прийняття рішень в комп'ютерних іграх у шахи.

– Розробка методів та алгоритмів системи штучного інтелекту для комп'ютерної гри в шахи на ігровому рушії Unity.

– Програмна реалізація комп'ютерної гри в шахи на ігровому рушії Unity та системи штучного інтелекту для неї.

Об'єктом дослідження є процес прийняття рішень в інтелектуальних системах шахових комп'ютерних програм.

Предметом дослідження є методи та алгоритми штучного інтелекту для знаходження вірних ходів у грі в шахи при змаганні комп'ютера проти людини, а також методи та алгоритми візуалізації гри у шахи на рушії Unity.

Методи дослідження базуються на методах штучного інтелекту, методах розробки програмного забезпечення, методах об'єктно-орієнтованого

програмування та методах розробки комп'ютерних ігор.

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

1. Удосконалено метод реалізації та використання штучного інтелекту для гри в комп'ютерні шахи та адаптовано його для реалізації на рушії Unity.

2. Розроблено вітчизняний продукт реалізації штучного інтелекту для комп'ютерної гри в шахи, реалізований на ігровому рушії Unity, який орієнтований на отримання більш цікавого, корисного та пізнавального ігрового досвіду.

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

Достовірність наукових результатів підтверджена теоретичними викладками, результатами комп'ютерного імітаційного моделювання, а також результатами тестування розробленого програмного забезпечення, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на ігровому рушії Unity, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній магістерській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне забезпечення, розроблене у цій кваліфікаційній магістерській роботі, призначене для реалізації системи штучного інтелекту для гри з комп'ютером у шахи. Ця система допоможе гравцям отримати більш "живий" ігровий досвід на базі розробленої комп'ютерної гри на ігровому рушії Unity.

Розроблена система призначена для використання на персональних комп'ютерах різної потужності.

Існує великий потенціал використання запропонованої системи штучного інтелекту й в інших логічних іграх. Це може використовуватись для отримання гравцем більш якісного ігрового досвіду при грі супроти комп'ютера.

В навчанні програмне забезпечення може використовуватись як посібник щодо розробки штучного інтелекту в комп'ютерних іграх.

1.2 Область застосування

Розроблене в ході роботи програмне забезпечення можна використовувати на будь-яких ПК під управлінням ОС Windows. Подібна система може бути корисною в якості приклада реалізації системи штучного інтелекту з використанням сучасного підходу до розробки інтелектуальних систем.

Особливо доцільним, на мій погляд, буде використання запропонованого програмного забезпечення та положень описаних в пояснювальній записці для навчальних закладів середнього та вищого рівнів. Воно надасть можливість користувачам і студентам комп'ютерних спеціальностей наочно зрозуміти основні положення розробки логічних ігор для ПК та ігрового штучного інтелекту під час навчання та досліджень.

Приведені дослідження та аналіз літературних джерел прискорить засвоєння та закріплення навчального матеріалу з вказаної тематики. Крім того надасть можливість розроблювати більш оптимальні алгоритми програм при розробці власних програмних і апаратних систем.

Вказані положення підтверджують актуальність поставленої мети досліджень та задачі магістерської роботи і їх виконання має підтвердити рівень атора роботи.

Використані в ході роботи над проектом алгоритми та прийоми програмування, а також супроводжений докладними коментарями програмний код і пояснювальна записка будуть корисні починаючим програмістам і можуть бути використані при підготовці студентів зі спеціальності комп'ютерна інженерія.

Слід згадати що програмне забезпечення, розроблене в процесі роботи над кваліфікаційною магістерською роботою, не має великих вимог до робочих станцій – мінімальні системні вимоги такі ж як і для встановлення ОС.

Програма може працювати на персональних комп'ютерах з Windows 7, або більш нової версії даної ОС.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи штучного інтелекту для комп'ютерної гри на рушії Unity, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній магістерській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Перед початком процесу розробки необхідно провести аналіз існуючих аналогів і джерел інформації. Аналіз особливостей цих програм, їх сильних та слабких сторін дасть змогу внести відповідні виправлення в розробку, що дозволить створити оптимальний програмний засіб.

Розроблювана програма має багато аналогів, які відрізняються алгоритмами пошуку виграшних кроків, оцінкою ситуації на ігровій дошці, стилем гри, кількісною оцінкою можливостей комп'ютерного рушія за допомогою рейтингу Ело (методу розрахунку відносної сили гравців в іграх, у яких беруть участь двоє гравців, зокрема, шахах), можливістю аналізу зіграної гри в шахи, наявністю графічного інтерфейсу, кількістю розрахунків, а також різним використанням потужностей комп'ютеру.

Велика кількість топових шахових рушіїв не має власного графічного інтерфейсу тому використовують через сторонні сервіси, сайти, рушії з графічним інтерфейсом та за допомогою окремих GUI для шахів.

Для аналізу були вибрані такі програми як Stockfish, Komodo, GNU Chess, Fruit, Houdini.

Stockfish

Stockfish – безкоштовний шаховий рушій з підтримкою UCI та відкритим вихідним кодом, доступний для різних десктопних та мобільних платформ. Він був розроблений Марко Костальбою, Джоном Кійськи, Гері Лінскоттом і Тордом Ромстадом, а також при підтримці спільноти розробників.

Stockfish займає перші місця серед більшості рейтингових списків і змагань серед комп'ютерних шахових програм, і визнається найсильнішим

шаховим рушієм, що не використовує GPU. Його рейтинг Ело становить 3552 для версії Stockfish 20221120 64-bit 4CPU.

Stockfish багаторазовий переможець неофіційного чемпону світу з комп'ютерних шахів.

Stockfish може використовувати до 1024 потоків ЦП у багатопроцесорних системах. Максимальний розмір таблиці транскрипції становить 128 ГБ. Рушій Stockfish використовує розширений алгоритм альфа-бета-пошуку і використовує бітові дошки.

На відміну від інших шахових рушіїв, Stockfish відрізняється більшою глибиною пошуку, завдяки більш агресивному алгоритму альфа-бета-відсікання, і скороченням пізніх кроків. Stockfish не має власного інтерфейсу, але його можна запустити через різні шахові GUI.

У серпні 2020 під впливом бурхливого розвитку шахових рушіїв на основі II Stockfish зазнав суттєвих змін в архітектурі і став використовувати нейронні мережі NNUE, оптимізовані під CPU та алгоритм пошуку Stockfish разом з традиційною функцією оцінки рушія. Stockfish перетворився на гібридний рушій, що використовує функцію оцінки на базі нейронних мереж (можливо відключити в налаштуваннях) для матеріально збалансованих позицій і традиційну функцію оцінки в інших випадках. Додавання використання нейронних мереж підвищило рівень гри рушія.

Стиль гри Stockfish, подібний до інших сильних шахових рушіїв, — універсальний, але зі спеціалізацією на тактику гри.

Код програми виконується за допомогою алгоритму розпаралелювання LAZY_SMP, який ефективно розпаралелюється на великій кількості ядер/потоків.

Код Stockfish був переписаний на мовах програмування асемблері та C. Ці версії оновлюються майже паралельно з основним проектом.

Stockfish має двадцять рівнів складності. Так як рівень гри Stockfish значно перевищує рівень гри людини, то доцільно порівнювати його тільки з іншими шаховими рушіями. Для прикладу якщо порівнювати стиль гри з рушієм

Komodo, сильною стороною якого є позиційна гра, Stockfish робить більшу увагу на тактику.

Komodo

Komodo – UCI-сумісний шаховий рушій, який розробили Дон Дейлі та Марк Лефлер(Рис 2.1). Розробку також підтримує гросмейстер Ларрі Кауфман, який окрім основних завдань розробки окремо налаштовує функцію оцінки. Безкоштовний до версії 12 включно. Є одним з найсильніших рушіїв. Разом зі Stockfish і Houdini, займає лідируючі позиції у відомих рейтингових списках рушіїв. Його рейтинг Ело становить 3525 для версії Dragon by Komodo 3 64-bit 4CPU.

Рисунок 2.1 – Аналіз зіграної гри за допомогою рушія Comodo

Головною особливістю рушія Komodo є орієнтація більшою мірою на оцінку позиції фігур на дошці, ніж на глибину пошуку допустимих кроків, і тому має характерний позиційний стиль гри. Дон Дейлі вважав, що у Komodo найкраща функція оцінки.

GNU Chess

GNU Chess – безкоштовна вільна шахова програма, написана на мові програмування C++ (Рис 2.2).

Рисунок 2.2 – Вікно Chess GUI з рушієм GNU Chess

GNU Chess не має графічного інтерфейсу, тому для комфортної гри необхідно підключати одну з графічних оболонок, наприклад, PyChess.

В основі роботи рушія лежить модифікація алгоритму альфа-бета мінімаксу професора Тоні Марсланда, під назвою Пошук Основних Відхилень (Principal Variation Search – PVS). Цей алгоритм дає надійний результат. Його рейтинг Ело становить 2818 для версії GNU Chess 5.60 64-bit.

Оцінка позиції фігур в версіях 5-х і більш пізніх трохи відрізняється від попередніх. Минулі версії використовували таблиці "фігура-поле" з неповною оцінкою кінцевих вузлів. В цих таблицях зберігаються значення, які відповідають важливості присутності фігур на даному полі. Таблиця заповнюється один раз на початку процесу пошуку кроку. Недолік цих таблиць в тому, що інформація, яка

в них зберігається зазвичай втрачає свою цінність зі зростанням глибини пошуку, тому що позиція на дошці змінюється дуже швидко. Зі збільшенням швидкодії комп'ютерів стає можливим все більш глибокий пошук, і таким чином таблиці можуть вводити програму в оману, видаючи ходи, що не відповідають позиції.

Нещодавно розробники GNU Chess повернулися до ідеї повного обліку кінцевих вузлів. GNU Chess використовує бітові дошки для представлення властивостей шахової дошки. Це схоже на початок розвитку комп'ютерних шахів, коли величезні ЕОМ 1960-х років використовували растрові зображення (бітові карти) для опису позицій фігур.

При оцінці кроку крім стандартних оцінок отриманих, за допомогою комбінування бітових дошок, використовуються більш затратні по ресурсам, але дуже важливі обчислення можливих переміщень короля та пішаків.

Fruit

Fruit – комп'ютерна шахова програма, розроблена у березні 2004 року Фаб'єном Летузьє. Райан Бенітес взяв участь у розробці у грудні 2005 року. Йоахім Ренг є основним тестувальником рушія Fruit від початку роботи над рушієм.

Fruit використовує класичний Negascout (PVS) ітераційний алгоритм із поглибленням по дереву гри разом з алгоритмом евристичного нуль-переходу. В оригінальній версії використовується спрощена функція оцінки з механізмами пошуку. У пізніших версіях функцію оцінки було вдосконалено. Але розмір шахової дошки відрізняється - Fruit використовує дошку розміром 16x16.

До версії 2.1 Fruit був відкритим програмним забезпеченням. Вихідний код версії 2.1 у вільному доступі, що стало великим внеском у розвиток комп'ютерних шахів в останні роки.

Houdini

Houdini – UCI-сумісний шаховий рушій для Windows, розроблений програмістом Робером Удар. Безкоштовний для некомерційного використання до версії 1.5a, проте більш пізні версії є комерційними.

Стиль гри Houdini схожий на стиль романтичної ери шахів, де переважав атакуючий, жертовний стиль. За словами розробника, перевага Houdini перед іншими топ-движками полягає в урахуванні рухливості фігури, що призводить до більш агресивного стилю гри та активного перебігу шахматної партії». Його рейтинг Ело становить 3383 для версії Houdini 6 64-bit 4CPU.

2.1.1 Огляд алгоритмів, які використовуються в основі роботи аналогічних існуючих систем

Основну роль в роботі штучного інтелекту для гри в шахи займає алгоритм пошуку кроків. Кожній позиції будь-якої фігури на дошці призначено певну числену вагу. Але можливих комбінацій розміщення фігур на ігровій дошці занадто багато(10^{120}), що робить грубий перебір досить затратним по часу та ресурсам комп'ютеру. Тому було розроблено досить велику кількість алгоритмів для пошуку виграшних кроків. Для зручності кроки гравця можна представити у вигляді бінарного дерева. Загальним принципом роботи подібних алгоритмів є пошук кроку чи комбінації кроків, які дадуть найбільшу перевагу над суперником, або мінімізують можливі втрати. Цей принцип відомий під назвою Мінімакс.

Для вибору та модифікації алгоритму пошуку кроків необхідно розглянути інші існуючі алгоритми, розібрати принцип їх роботи, визначити їх сильні та слабкі сторони.

Для аналізу були вибрані такі алгоритми як Minimax, Alpha-beta pruning, Alpha-beta pruning with late move reductions, Principal Variation Search, MCTS.

Minimax

Алгоритм Мінімакс(Minimax) – це рекурсивний алгоритм, областю застосування якого є задачі прийняття рішень та теорія ігор. Він підбирає оптимальний хід для гравця, припускаючи, що супротивник також веде оптимальну гру.

Алгоритм Мінімакс переважно використовується в іграх зі штучним інтелектом. Наприклад в шахах, шашках, хрестиках-нуликах, го та інших іграх з двома гравцями. Алгоритм передбачає що кожен гравець намагається зробити найкращий з можливих ходів у грі,.що в кінцевому результаті призведе до перемоги. Алгоритм Мінімаксу для обходу дерева гри використовує алгоритм Пошуку в глибину(DFS).

Властивості алгоритму Мінімаксу:

1. Алгоритм Мінімаксу скінченний. Якщо рішення існує в дереві пошуку то алгоритм його знайде.

2. Алгоритм Мінімаксу є оптимальним, якщо обидва гравці роблять оптимальні кроки.

3. Скінченність алгоритму за часом та кількістю кроків. Так, як для пошуку по дереву гри використовується алгоритм DFS, то тимчасова складність алгоритму Мінімаксу становить $O(b^m)$, де b - коефіцієнт розгалуження дерева гри, а m - максимальна глибина дерева. Просторова складність алгоритму Мінімаксу також відповідає алгоритму DFS і становить O(bm).

Головний недолік алгоритму Мінімаксу полягає в тому, що він повільно працює в складних іграх по типу шах, або го. Подібні ігри мають величезний коефіцієнт розгалуження, і гравець має багато варіантів вибору. Цей недолік алгоритму Мінімаксу можна виправити за допомогою Альфа-бета скорочення.

Alpha-beta pruning

Альфа-бета відсікання (alpha-beta pruning) - алгоритм пошуку, метою роботи якого є скорочення кількості вузлів, які проходять оцінку в дереві пошуку алгоритмом Мінімаксу. Використовується для реалізації штучного інтелекту в іграх. Принцип алгоритму оснований на ідеї, що оцінювання гілки дерева пошуку може бути достроково припинено, якщо результат цієї гілки гірший ніж попередньої. Альфа-бета відсікання є оптимізацією, тому що не впливає на коректність роботи алгоритму.

На ефективність алгоритму відчутно впливає завчасне сортування вершин

дерева (без перебору або з перебором на меншу глибину) — при сортуванні чим більше на початку розглянуто «вдалих» варіантів, тим більше «безперспективних» гілок може бути відсічено без подальшого аналізу.

Alpha-beta pruning with late move reductions

У комп'ютерних шахах та інших іграх, де можуть грати комп'ютери, Скорочення пізніх кроків є неспецифічним для гри удосконаленням алгоритму альфа-бета відсікання, яке має більш ефективно дослідити дерево пошуку гри. Воно використовує правило, що ефективний порядок ходів для конкретної гри дає підстави програмі шукати найвдаліші кроки раніше. Якщо в пошуку відбудеться відсікання, перші кілька кроків, викличуть його з більшою ймовірністю. У таких іграх, як шахи, більшість програм шукають виграшні взяття і «вбивчі ходи». Скорочення пізніх ходів зменшить глибину пошуку для ходів, які знаходяться глибше в даному вузлі. Це дає змогу програмі глибше шукати критичні лінії і грати краще.

Більшість шахових рушіїв шукатимуть перші кілька ходів у вузлі на повну глибину. Часто вони не скорочують ходи, які вважаються дуже тактичними, такі як захоплення чи просування. Якщо оцінка руху на зменшеній глибині менша, ніж альфа, хід вважається поганим. Однак, якщо оцінка більша, ніж альфа, скорочений пошук нічого не говорить нам, тому нам доведеться виконати повний пошук.

Це скорочення пошуку може призвести до іншого простору ходів, ніж чистий метод альфа-бета відсікання, який може дати інші результати. Необхідно бути обережними при виборі критеріїв скорочення, інакше при пошуку будуть пропущені деякі важливі ходи.

Principal Variation Search

Пошук основного варіанту (PVS) – це негамакс-алгоритм, який може бути більш швидким, ніж алгоритм альфа-бета відсікання. Подібно до алгоритму альфа-бета відсікання, PVS є алгоритмом спрямованого пошуку для обчислення мінімаксного значення вузла в дереві. Він домінує над альфа-бета відсіканням у

тому сенсі, що він ніколи не буде перевіряти вузол, який може бути відсіяним алгоритмом альфа-бета відсікання; однак він покладається на точне впорядкування вузлів, щоб отримати вигоду з цієї переваги.

PVS працює найкраще, коли є гарний порядок ходів. Насправді порядок ходів часто визначається попередніми більш поверхневими пошуками. Він дає більше відкинутих ходів, ніж алгоритм альфа-бета відсікання, припускаючи, що перший досліджений вузол є найкращим. Іншими словами, передбачається, що перший вузол знаходиться в основному варіанті. Потім він може перевірити, чи вірне це припущення, шляхом пошуку вузлів, що залишилися в порожньому просторі (також відомому як простір розвідки; коли альфа і бета рівні), що швидше, ніж пошук в звичайному просторі алгоритму альфа-бета відсікання. Якщо припущення хибне, значить, першого вузла не було в основній варіації, і пошук продовжується у звичайному режимі алгоритму альфа-бета відсікання. Отже, PVS працює найкраще, коли порядок ходів гарний. З випадковим порядком ходів PVS займе більше часу, ніж алгоритм альфа-бета відсікання. Хоча цей алгоритм не перевірятиме жодного вузла, який алгоритм альфа-бета відсікання не пропустив би, йому доведеться повторно обшукати багато вузлів.

MCTS

Пошук по дереву Монте-Карло (MCTS) – евристичний алгоритм пошуку для деяких видів процесів прийняття рішень, наприклад, для настільних ігор. У цій сфері MCTS використовується для вирішення дерева гри.

Алгоритм MCTS фокусується на аналізі найкращих ходів та розширенні дерева пошуку на основі випадкової вибірки простору пошуку. Застосування пошуку по дереву Монте-Карло в іграх засноване на безлічі симуляцій ігрової ситуації. Кожна симуляція виконується до остаточного завершення гри, з випадковим вибором кроків. На основі кінцевого ігрового результату кожної симуляції проходить переважування вузлів дерева, для того щоб збільшити ймовірність знаходження виграшних ходів.

Найпростіший спосіб використання симуляцій - застосовувати однакову

кількість симуляцій після кожного допустимого ходу поточного гравця, а потім вибирати хід, який призвів до найбільшої кількості перемог. Ефективність цього методу під назвою «Чистий пошук гри за методом Монте-Карло», зростає ійзі збільшенням кількості симуляцій, оскільки більше симуляцій призначається ходам, які часто призводили до перемоги гравця відповідно до попередніх симу. Ляцій. Кожен раунд пошуку по дереву Монте-Карло складається з чотирьох кроків:

1. Вибір. Почніть з кореня R і виберіть послідовні дочірні вузли, поки не буде досягнуто кінцевий вузол L. Корінь - це поточний стан гри, а лист - це будь-який вузол, у якого є потенційний дочірній вузол, з якого ще не було ініційовано моделювання. У наступному розділі більше йдеться про спосіб усунення вибору дочірніх вузлів, який дозволяє дереву гри розширюватися у бік найбільш перспективних ходів, що є суттю пошуку по дереву Монте-Карло.

2. Розширення. Якщо L не завершує гру рішуче (наприклад, перемога/програш/нічия) для будь-якого гравця, створіть один (або кілька) дочірніх вузлів та виберіть вузол C з одного з них. Дочірні вузли це будь-які допустимі ходи з ігрової позиції, визначеної L .

3. Моделювання. Завершіть одне випадкове відтворення з вузла C. Цей крок іноді називають відтворенням або розгортанням. Гра може бути такою ж простою, як вибір однакових випадкових ходів до моменту поки гра не буде завершена.

4. Зворотнє розповсюдження: використовуйте результат відтворення для оновлення інформації у вузлах по дорозі від C до R.

Симуляції повторюються до тих пір, поки залишається час, відведений на хід. Після цього вибирається хід з найбільшою кількістю симуляцій.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки будь-якого додатку слід визначитись з декількома основними рішеннями щодо засобів розробки програмного забезпечення. Найбільш важливими з них є вибір середовища розробки та мови програмування.

Вибір мови програмування залежить від декількох чинників:

1. Тип розроблюваного програмного забезпечення.

2. Складність структури та функціоналу розроблюваного програмного забезпечення.

3. Наявність кваліфікованого спеціаліста, який може виконати поставлені завдання на вибраній мові програмування.

Середовище розробки також відіграє важливу роль у процесу розробки, але його вибір більш орієнтується на звички та досвід роботи програміста з ним. Правильно підібрані засоби розробки, які підібрані під технічне завдання та можливості розробника можуть зберегти багато часу, зусиль та відповідно коштів, що витрачаються на розробку.

Розроблюване програмне забезпечення системи штучного інтелекту для гри в шахи не має особливих вимог до середовища розробки та мови програмування. Тому в якості середовища було вибрано рушій Unity та мову програмування C#.

C# – об’єктно-орієнтована мова програмування загального призначення. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів, делегати, атрибути, події, змінні, властивості, узагальнені типи та методи, ітератори, LINQ, винятки, коментарі у форматі XML.

Запозичивши певні особливості своїх попередників – мов C++, Delphi і, особливо, Java - C#, спираючись на практику їх використання, виключає деякі підходи, з якими виникали проблеми при розробці програм, наприклад, C# на відміну від C++ не підтримує множинне спадкування класів (між тим

допускається множинна реалізація інтерфейсів).

Програми C# виконуються в .NET, віртуальній системі виконання, яка в своїй основі використовує загальномовне середовище виконання (CLR), який дозволяє абстрагуватися від конкретної мови програмування та виконувати код однаково на всіх платформах та набір бібліотек класів. Середовище CLR – це реалізація загальномовної інфраструктури мови (CLI), яка є міжнародним стандартом від корпорації Майкрософт. CLI є основою для створення середовищ виконання та розробки, у яких мови та бібліотеки прозоро працюють одна з одною.

Вихідний код, написаний мовою C# компілюється в проміжну мову (IL), яка відповідає специфікаціям CLI. Код на мові IL та ресурси, у тому числі растрові зображення та рядки, зберігаються у збірці у вигляді файлів бібліотек, зазвичай з розширенням .dll. Збірка обов'язково має файл маніфесту з інформацією про певні особливості цієї збірки.

Під час виконання програми C# збірка завантажується у середу CLR. Середовище CLR виконує JIT-компіляцію з коду мовою IL в інструкції машинної мови. Середовище CLR також виконує інші операції, наприклад, автоматичне складання сміття, обробку винятків та управління ресурсами. Код, який виконується середовищем CLR, іноді називають "керованим кодом". "Некерований код" компілюється на машинну мову, призначену для конкретної платформи.

Основні переваги мови C#:

1. Відносно проста для вивчення мова програмування, яка дозволяє виконувати задачу майже з будь-якої сфери

2. Платформа .Net здатна запропонувати широкий спектр функціоналу, який доповнюється синтаксичними конструкціями.

3. Реглярне оновлення функціоналу мови відносно до новітніх стандартів.

4. Завдяки C Sharp можна успішно розробити як стандартні додатки для веб-сайтів, так і сучасні мобільні додатки. Завдяки наявності величезної

різноманітності інструментів та бібліотек, розробити можна навіть нейромережі.

Unity – міжплатформне середовище розробки комп'ютерних ігор, розроблене американською компанією Unity Technologies.

Основними перевагами Unity є наявність візуального середовища розробки, міжплатформної підтримки та модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами та утруднення при підключенні зовнішніх бібліотек.

Середовище розробки Unity підходить для розробки штучного інтелекту для гри в шахи через простоту реалізації усіх програмних рішень, та велику кількість прикладів їх використання.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на магістерську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи штучного інтелекту для комп'ютерної гри на рушії Unity.

В процесі розробки магістерської роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У ході виконання цієї кваліфікаційної випускної роботи було розроблено комп'ютерну гру у шахи на ігровому рушії Unity, а також штучний інтелект для гри в неї з комп'ютером у якості супротивника.

Для створення штучного інтелекту для гри у комп'ютерні шахи було реалізовано наступні методи:

1. Переміщення.
2. Оцінка дошки.
3. Мінімакс.
4. Альфа-бета-відсікання.
5. Багатошарова нейронна мережа.

Переміщення

Кожен тип фігури має схему руху. Використовується для створення ходів. Для всіх фігур гравця створюється список псевдоправильних ходів. Законні ходи зберігаються.

Оцінка дошки

Насправді краще оцінювати ходи відразу під час генерації ходів, а не під час пошуку в системі. Це тому, що пошук набагато ефективніший, коли ходи впорядковані. Коли спочатку оцінюються кращі ходи, більш імовірно, що пошук може проігнорувати багато ходів.

Оцінка дає бал. Якщо чорне краще, це позитивно і негативно для позицій на користь білих. Кожна фігура має різну цінність, наприклад, королева – 9, пішка – 1. Матеріальна оцінка – це сума фігур чорних, віднятих від суми білих. Позиція фігур також важлива. Оцінка також стосується закінчення гри та того, хто є переможцем.

Алгоритм Мінімакс

Мінімаксний алгоритм (MinMax) є алгоритмом, який відноситься до теорії ігор для двох гравців з нульовою сумою (і повною інформацією) ігор мінімізації втрат максимальних (тобто скажімо, в гіршому випадку). Для великого сімейства ігор мінімаксна теорема фон Неймана гарантує існування такого алгоритму, навіть якщо практично часто буває нелегко знайти його. Шістнадцяткова гра – це приклад, в якому встановлено існування такого алгоритму, і він показує, що перший гравець завжди може виграти, не знаючи про цю стратегію.

Це змушує комп'ютер використовувати всі можливості для обмеженої кількості ходів і надавати їм значення, що враховує вигоди для гравця та його супротивника. Тоді найкращий вибір – це той, який мінімізує втрати гравця, припускаючи, що противник, навпаки, прагне їх максимізувати (гра з нульовою сумою).

Існують різні алгоритми, засновані на MinMax, для оптимізації пошуку кращого ходу шляхом обмеження кількості вузлів, що відвідуються в дереві гри, найбільш відомим з яких є альфа-бета-обрізка. На практиці дерево часто дуже велике, щоб його можна було повністю вивчити (як у грі в шахи або го). Потім досліджується лише частина дерева.

У разі великих дерев II (експертна система, оцінка шляхом навчання з прикладів тощо. буд.) може використовуватися обрізання певних гілок з урахуванням оцінки їхньої корисності. Це те, що використовується, наприклад, у контексті го.

Принцип

Алгоритм мінімаксу звертається до дерева ігор, щоб знайти в корні значення (зване «ігровим значенням»), яке рекурсивно обчислюється так:

- minimax (p) = f (p), якщо p - лист дерева, де f - функція оцінки позиції в грі;
- minimax (p) = max (minimax (O_1),…, minimax (O_n)), якщо p - вузол Player з дочірніми елементами O_1 ,…, O_n ;

– minimax (p) = min (minimax (O_1),…, minimax (O_n)), якщо p є протилежним вузлом з дочірніми елементами O_1 ,…, O_n .

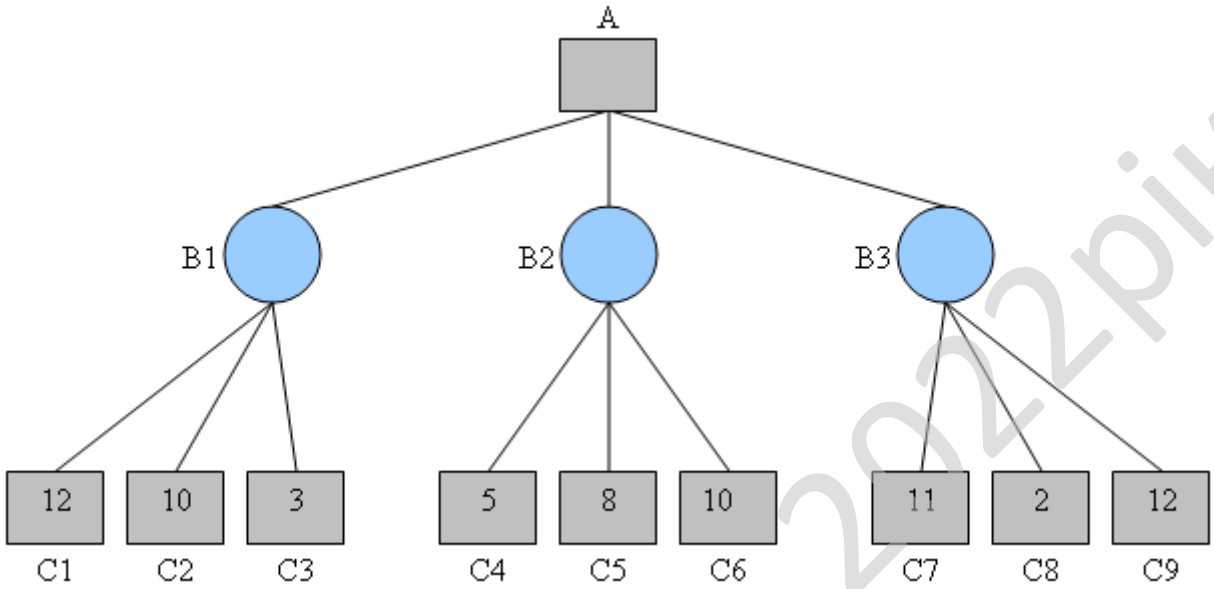

Рисунок 3.1 – Приклад представлення дерева гри в алгоритмі Мінімакс

На схемі вище сірі вузли представляють вузли гравців, а сині – вузли, що протистоять. Щоб визначити значення вузла A ми вибираємо максимальне значення набору вузлів B (A – вузол гравця). Отже, необхідно визначити значення вузлів B, кожен з яких набуває мінімального значення, що зберігається в їх дочірніх вузлах (вузли B протилежні). Вузли C є листям, тому їх значення може бути обчислено функцією оцінки.

Псевдокод

Псевдокод алгоритму мінімаксного обмеження глибини показаний нижче:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
```

```
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Альфа-бета відсікання

В інформатиці, точніше в штучному інтелекті та теорії ігор, альфа-бета-обрізання (скорочено αβ-обрізання) – це метод, що дозволяє зменшити кількість вузлів, що оцінюються мінімаксним алгоритмом. Він використовується у комп'ютерних програмах, які грають у ігри для двох, такі як шахи чи шашки.

Принцип

Алгоритм мінімакса виконує повне дослідження пошуку дерева до заданого рівня. Відсікання альфа-бета може значно оптимізувати алгоритм мінімаксу без зміни результату. І тому він проводить лише часткове дослідження дерева. При дослідженні немає необхідності перевіряти піддерева, які призводять до конфігурацій, значення яких не сприятиме обчисленню посилення в корені дерева. Іншими словами, відсікання αβ не оцінює вузли, які можна подумати, якщо функція оцінки приблизно вірна, що їх якість буде нижчою, ніж якість вже оціненого вузла.

На цьому етапі важливо розуміти, що для ідеальної функції оцінки (наприклад, у Марієнбаді) не потрібно обходу дерева, а для повного обходу дерева (наприклад, у хрестики-нуліки) не потрібно ніякої функції d». Отже, йдеться про поєднання в кращому разі евристики (функція оцінки, більш менш довільна, але глобального масштабу) і жорсткої логіки (дослідження дерева можливостей), що можливе тільки в області, яку ми раніше обмежили.

Ми об'єднаємо їх, щоб прискорити дослідження, за умови, що функція оцінки не така вже й погана. Візьмемо α і β, що належать до області приходу оціночної функції, такі, що $\alpha < \beta$. Ми визначаємо функцію AlphaBeta наступним чином:

– AlphaBeta (P, α, β) = g (P), якщо P - лист дерева, а g – функція оцінки вузла.

– AlphaBeta (P, α, β) = min (β, max (-AlphaBeta (, -β, -α))), де – дочірні елементи вузла P.

Ми називаємо вікном αβ пару (α, β), де α і β – два параметри виклику функції. Відсічені вузли – це ті, які викликатимуться з таким вікном, як α ≥ β. Є 3 типи вузлів, які не можна обрізати:

– Тип вузла 1: вікно виклику: (-∞, + ∞).

– Тип вузла 2: вікно виклику: (-∞, β) з β ≠ + ∞.

– Тип вузла 3: вікно виклику: (α, + ∞) з α ≠ -∞.

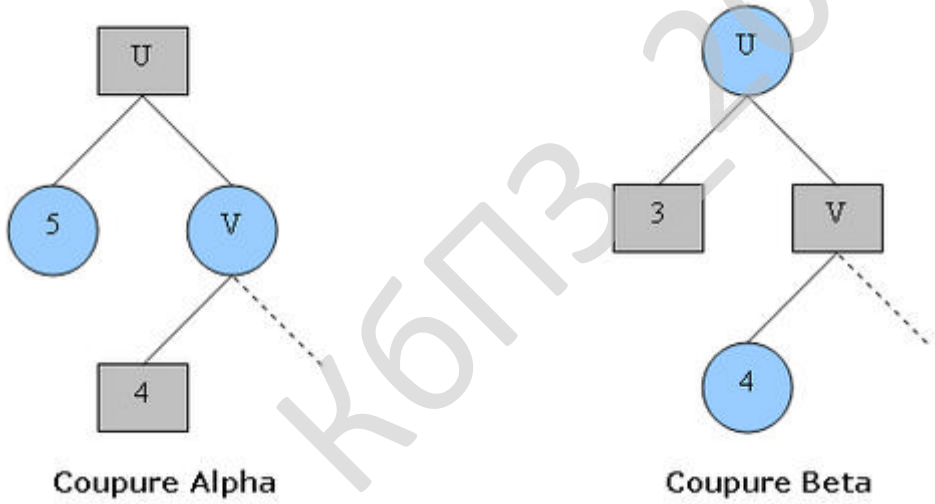

Рисунок 3.2 – Приклад роботи алгоритму Альфа-бета відсікання

На схемі вище показано два можливі типи розрізів. Вузли Min представлені синім кружком, а вузли Max – сірим квадратом. Нагадування: вузли Min набувають мінімального значення своїх дочірніх вузлів (і, відповідно, максимальне значення для вузлів Max).

Альфа-відсікання: перший дочірній елемент вузла Min V має значення 4, тому значення V буде не більше 4. Таким чином, вузол Max U прийматиме значення 5 (максимум між 5 і значенням, меншим або рівним 4).

Бета-відсікання: перший дочірній елемент вузла Max V має значення 4, отже V буде не менше 4. Вузол Min U, отже, прийме значення 3 (мінімум між 3 і

значенням більше або рівним 4).

Псевдокод

Нижче наведено псевдокод альфа-бета-алгоритму: ініціалізується значенням -∞, а - значенням + ∞.

```
fonction alphabeta(nœud, α, β)
   si nœud est une feuille alors
       retourner la valeur de nœud
   sinon si nœud est de type Min alors
           v = +∞
           pour tout fils de nœud faire
               v = min(v, alphabeta(fils, α, β))
               si α ≥ v alors  /* coupure alpha */
                   retourner v
               β = min(β, v)
    sinon
           v = -∞
           pour tout fils de nœud faire
               v = max(v, alphabeta(fils, α, β))
               si v ≥ β alors /* coupure beta */
                   retourner v
               α = max(α, v)
    retourner v
```

Так само як алгоритм мінімакса можна замінити на NegaMax, можна спростити Alpha-Beta. Це дає такий алгоритм:

```
fonction alphabeta(nœud, i)
   si nœud est une feuille alors
       retourner la valeur de nœud
   sinon
       j = -∞
       pour tout fils de nœud faire
           j = max(j, alphabeta(fils, j))
           si -j ⩽ i alors
               retourner -j
       retourner -j
```

Ілюстрація алгоритму наведена на дереві нижче, що вже позначене

значеннями мінімаксу, рис. 3.3.

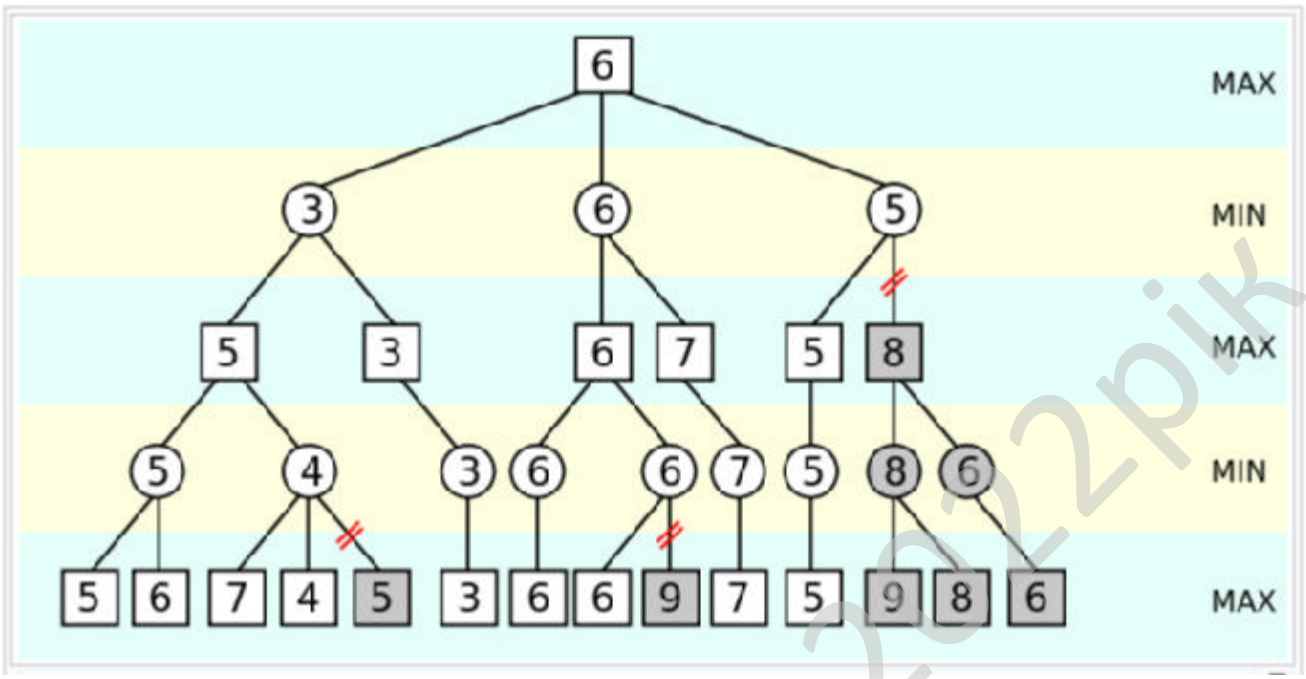

Рисунок 3.3 – Minimax з альфа-бета-відсіканням

У прикладі з рис. 3.3 було зроблено кілька розрізів. Зліва направо:

1. Вузол MIN щойно оновив своє поточне значення до 4. Це значення, яке може лише зменшуватись, вже менше α = 5, поточного значення попереднього вузла MAX. Тому останній, який шукає максимально можливу цінність, у жодному разі не вибере його.

2. Вузол MIN щойно оновив своє поточне значення до 6. Це значення, яке може лише зменшуватись, вже дорівнює α = 6, поточному значенню попереднього вузла MAX. Той, який шукає вищого значення, він все одно не оновлюватиме свого значення, якщо цей вузол коштує 6 або менше.

3. Вузол MIN щойно оновив своє поточне значення до 5. Це значення, яке може лише зменшуватись, вже менше α = 6, поточного значення попереднього вузла MAX. Останній, хто шукає максимально можливу цінність, у жодному разі не вибере її.

Згорткові нейронні мережі

Згорткові нейронні мережі зручно використовувати для відбору ходів у

логічних іграх. Справжня революція, що призвела до перемоги програми над людиною в Го, відбулася, коли почали застосовувати згорткові нейронні мережі. У Го дуже важко вигадати структуру оціночної функції, яка без перебору зможе з достатньою якістю оцінити, наскільки хороша позиція. Потрібно якось оцінювати патерни, структури, які утворюються фішками на дошці. Знайти досить хороше рішення не вдавалося, поки не з'явилися згорткові нейронні мережі, що добре справляються з цим завданням. Отже, і для гри в шахи ці нейронні мережі досить добре підійдуть.

Нейронні мережі імітують людський мозок для вирішення складних проблем та пошуку закономірностей у даних. За останні кілька років вони замінили багато алгоритмів машинного навчання. Базова модель нейронної мережі складається з нейронів, організованих шарами. Кожна нейронна мережа має вхідний та вихідний шар та кілька прихованих шарів, доданими до неї залежно від складності проблеми. При передачі даних через шари нейрони навчаються і розпізнають ознаки. Це уявлення нейронної мережі називається моделлю. Після того, як модель навчена, ми просимо мережу зробити прогнози на основі тестових даних.

Згорткова нейронна мережа являє собою особливий тип нейронної мережі. Ян Лекун запропонував їх у 1998 році, де вони розпізнавали число, яке є у вхідному зображенні. Також вони застосовуються для розпізнавання мови, сегментації зображення та обробки тексту. До створення згорткових нейронних мереж багатошарові персептрони використовувалися при побудові класифікаторів зображень. Багатошарові персептрони займають багато часу для пошуку інформації у зображеннях, оскільки кожен вхід має бути пов'язаний з кожним нейроном у наступному шарі. Згорткові нейронні мережі обійшли їх, використовуючи концепцію, яка називається локальною зв'язністю. Це означає, що ми підключимо кожен нейрон лише до локальної області входів. Це мінімізує кількість параметрів, дозволяючи різним частинам мережі спеціалізуватися на високорівневих ознаках, таких як текстура або візерунок, що повторюється.

Згортки не використовують повністю пов’язані шари. Вони використовують розріджені шари, які приймають матриці як вхідні дані, що дає перевагу над звичайними нейронними пережами. Вихідний шар об’єднує отримані дані кожного прихованого вузла, щоб знайти закономірності. Нижче наведено зображення, як шари пов’язані.

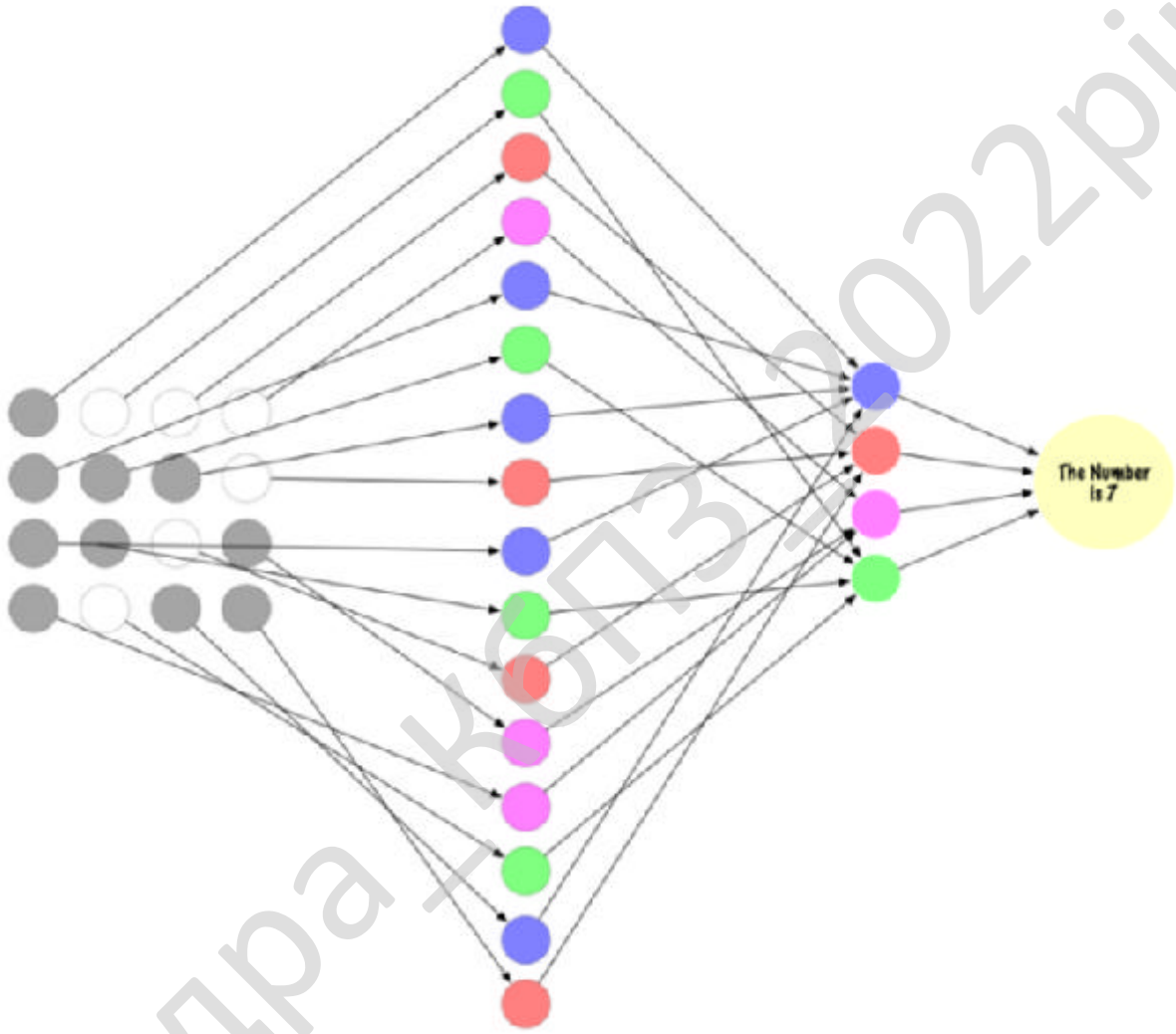

Рисунок 3.4 – Приклад згорткової нейронної мережі

3.2 Розробка структурної схеми

Структурна схема розробленого програмного забезпечення зображена на рисунку 3.5.

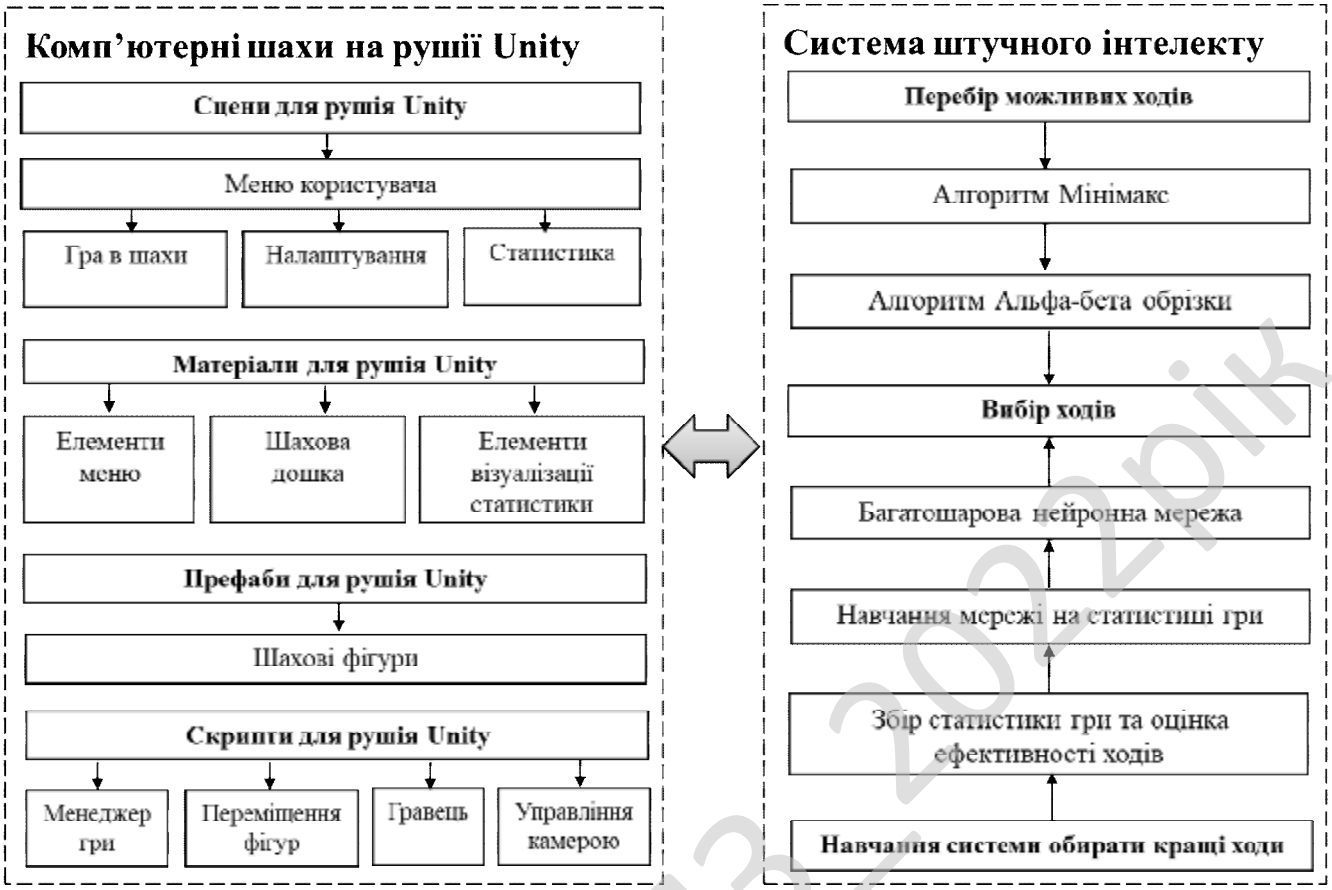

Рисунок 3.5 – Структурна схема системи

На рисунку видно, що система складається з наступних частин:

– Комп'ютерні шахи на рушії Unity.

– Система штучного інтелекту.

Блок «Комп'ютерні шахи на рушії Unity» містить:

– Сцени для рушія Unity: «Меню користувача», «Гра в шахи», «Налаштування», «Статистика».

– Матеріали для рушія Unity: «Елементи меню», «Шахова дошка», «Елементи візуалізації статистики».

– Префаби для рушія Unity: «Шахові фігури».

– Скрипти для рушія Unity: «Менеджер гри», «Переміщення фігур», «Гравець», «Управління камерою».

Блок «Система штучного інтелекту» містить:

– Перебір можливих ходів: Алгоритм Мінімакс, Алгоритм Альфа-бета обрізки.

– Навчання системи обирати кращі ходи: Збір статистики гри та оцінка ефективності ходів, Навчання мережі на статистиці гри, Багатошарова нейронна мережа.

– Інтелектуальний вибір ходів.

3.3 Розробка функціональної схеми

Функціональна схема системи наведена на рис. 3.6.

Рисунок 3.6 – Функціональна схема системи

Розроблена система складається з наступних блоків:

1. Блок інтерфейсу користувача та управління шахами:

– Меню користувача та налаштування.

– Візуалізація шахової дошки та фігур.

– Візуалізація статистики ігор.

2. Блок логіки гри:

– Правила переміщення фігур та обробки результатів ходів.

– Управління переміщенням фігур.

– Застосування правил після ходу.

– Оцінка позиції фігури.

– Виведення результатів гри та збереження статистики.

3. Блок системи штучного інтелекту:

– Модуль пошуку та вибору ходів (Алгоритм Мінімакс, Алгоритм Альфа-бета обрізки).

– Модуль навчання системи у ході гри (Збір статистики ігор, Оцінка ефективності ходів, Багатошарова нейронна мережа).

3.4 Розробка діаграми процесів

На рисунку 3.7 зображена діаграма процесів, що описує розроблену систему.

Робота розробленого програмного забезпечення складається з наступних основних процесів:

– «Головне вікно програми».

– «Обрання параметрів гри».

– «Гра у шахи».

– «Статистика гри».

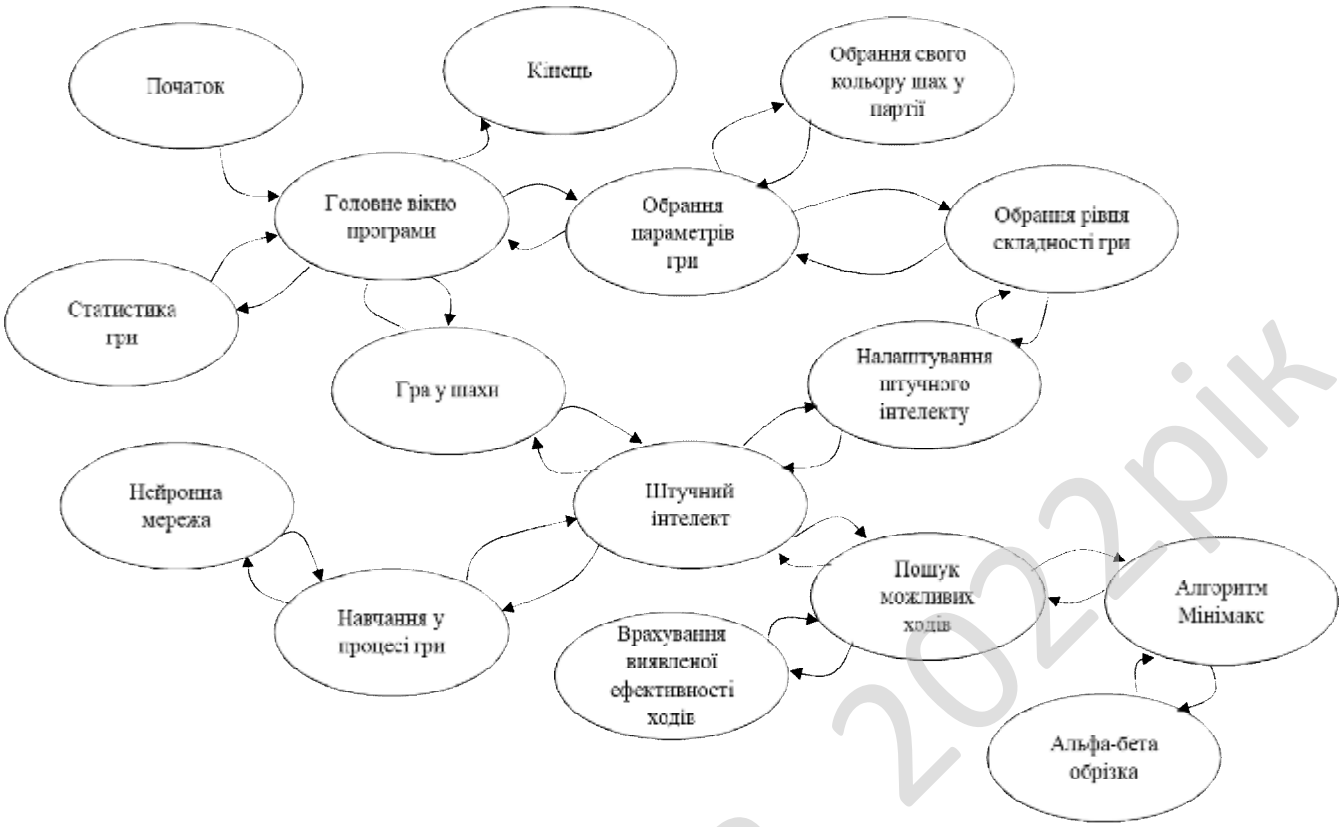

Рисунок 3.7 – Діаграма процесів системи

Процес «Обрання параметрів гри» містить у собі наступні процеси:

– «Обрання свого кольору шах у партії».

– «Обрання рівня складності гри».

Процес «Гра у шахи» запускає процес «Штучний інтелект» для реалізації гри комп’ютера проти людини. Процес «Штучний інтелект» містить наступні процеси:

– Налаштування штучного інтелекту.

– Пошук можливих ходів.

– Алгоритм Мінімакс.

– Альфа-бета обрізка.

– Врахування виявленої ефективності ходів.

– Навчання у процесі гри.

– Нейронна мережа.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 зображено блок-схему основної програми розробленого програмного забезпечення.

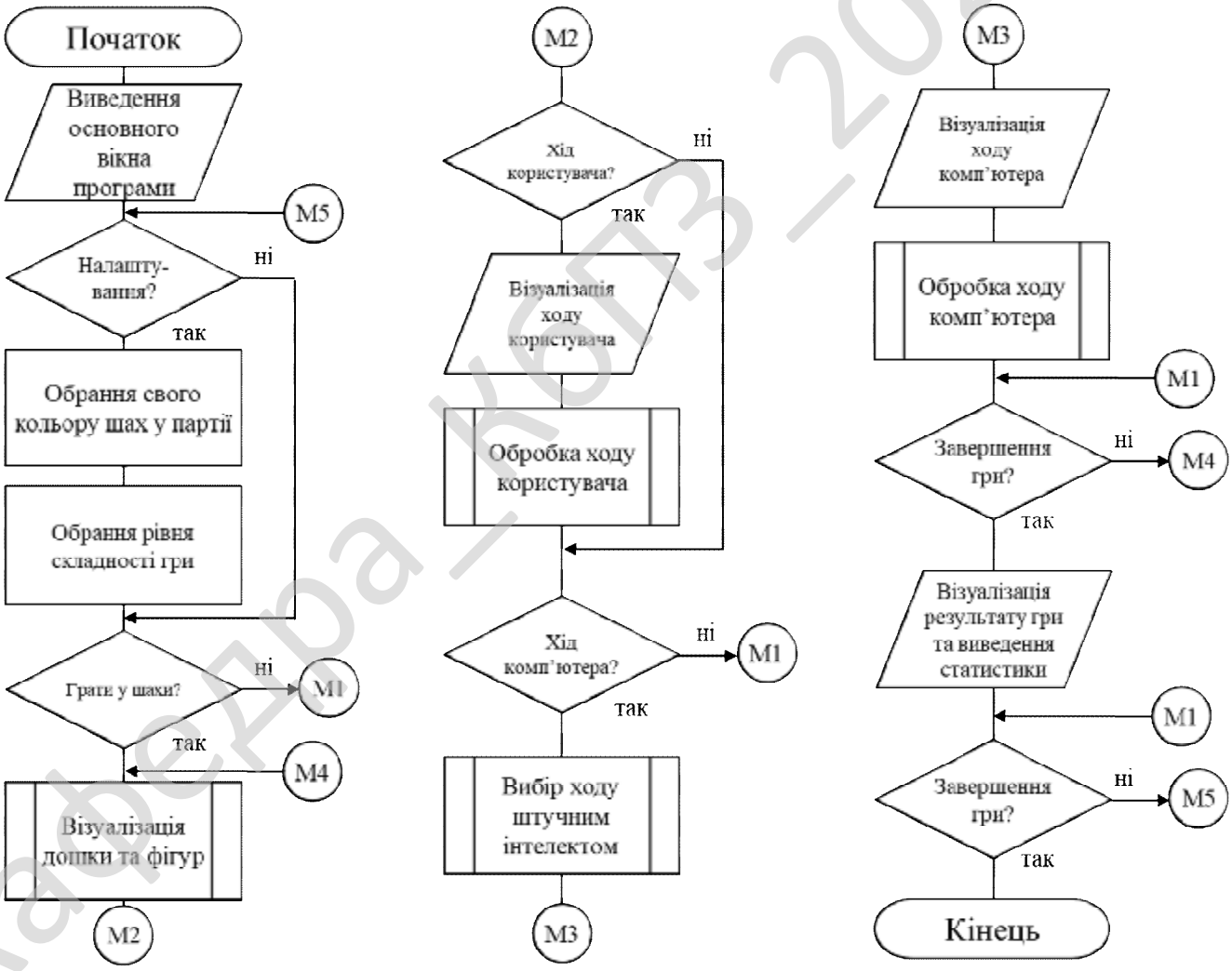

Рисунок 4.1 – Блок-схема основної програми

Роботу розробленого програмного забезпечення умовно можна поділити на ряд етапів, наведених нижче.

Етап 1. Візуалізація шахівниці з генерацією ходів.

На цьому етапі ми будемо використовувати модулі для створення ходів і для візуалізації дошки. Бібліотека, яка відповідає за генерацію ходів, дозволяє застосовувати всі шахові правила, тому ми можемо розраховувати кожну дію для конкретного розташування фігур. При натисканні на малюнку вона відкриється в повній роздільній здатності. Робота з цими бібліотеками дозволяє сконцентруватися на головному завданні – пошуку та створенні алгоритму, який дозволяє знайти оптимальний хід. Роботу починаємо з написання функції, яка повертає випадковий перебіг зі списку всіх можливих.

Етап 2. Оцінка позиції.

Тепер давайте розберемося, яка сторона має перевагу в тому чи іншому положенні. Найпростіший шлях – підрахувати відносну силу фігур на дошці, це можна зробити за допомогою таблиці з рис. 4.2.

Рисунок 4.2 – Таблиця підрахунку відносної сили фігур на дошці

Використовуючи функцію оцінки, ми маємо можливість створити алгоритм, який вибирає хід із максимальною оцінкою.

Етап 3. Дерево пошуку з мінімакс.

Після цього ми створюємо дерево пошуку. Тепер програма може вибрати із нього найкращий хід. Це робиться за допомогою мінімакс-алгоритму. Тут рекурсивне дерево з відображенням усіх можливих ходів аналізується до заданої глибини. Позиція ж оцінюється за листям нашого дерева. Далі ми повертаємо мінімальне чи максимальне значення нащадка у батьківський вузол. Все залежить від того, хід якої сторони зараз прораховується. Інакше кажучи, результат максимізується чи мінімізується кожному з рівнів.

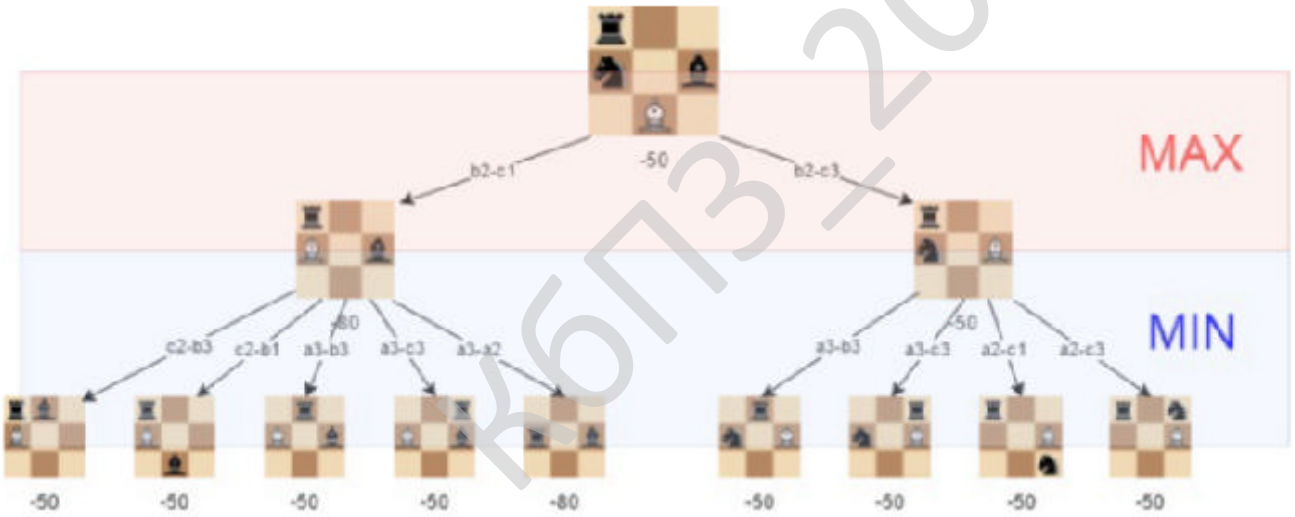

Рисунок 4.3 – Ілюстрація прикладу роботи алгоритму Мінімакс

Тут найкращим ходом для білих є b2-c3, оскільки він гарантує, що гравець дістанеться позиції з оцінкою -50.

З мінімакс-алгоритмом штучний інтелект шахів стає розуміти базову тактику гри. Варто зазначити, що ефективність мінімакс-алгоритму збільшується з глибиною пошуку. За це відповідає наступний етап.

Реалізація алгоритму Мінімакс:

```
        //minimax algorithm

        private int Minimax(Board board, int depth, int alpha, int beta, bool
isMaximizingPlayer)
        {
            if (depth == 0)
                return CalculatePoint(board);

            if (isMaximizingPlayer)
            {
                int bestValue = int.MinValue;

                List<Move> possibleMoves = Board.GetAllLegalMoves(Player.Black,
board);

                OrderMoves(possibleMoves, board);
                foreach (var move in possibleMoves)
                {
                    Board newBoard = GenerateMovedBoard(board, move);

                    int value = Minimax(newBoard, depth - 1, alpha, beta, false);

                    bestValue = Math.Max(value, bestValue);

                    alpha = Math.Max(alpha, value);

                    if (beta <= alpha)
                    {
                        break;
                    }
                }

                return bestValue;
            }
            else
            {
                int bestValue = int.MaxValue;

                List<Move> possibleMoves = Board.GetAllLegalMoves(Player.White,
board);

                OrderMoves(possibleMoves, board);
                foreach (var move in possibleMoves)
                {
                    Board newBoard = GenerateMovedBoard(board, move);

                    int value = Minimax(board, depth - 1, alpha, beta, true);

                    bestValue = Math.Min(value, bestValue);

                    beta = Math.Min(beta, value);

                    if (beta <= alpha)
                    {
                        break;
                    }
                }
```

```
                return bestValue;
            }
        }

        public Move GetBestMove(Board board)
        {
            int bestValue = int.MinValue;
            Move bestMove = null;
            bool turn;
            if (board.Turn == Player.Black)
            {
                turn = false;
            }
            else
            {
                turn = true;
            }

            List<Move> possibleMoves = Board.GetAllLegalMoves(board.Turn, board);

            OrderMoves(possibleMoves, board);
            foreach (var move in possibleMoves)
            {
                Board newBoard = GenerateMovedBoard(board, move);

                int value = Minimax(newBoard, depth, int.MinValue, int.MaxValue,
turn);

                if (value >= bestValue)
                {
                    bestValue = value;
                    bestMove = move;
                }
            }

            return bestMove;
        }


    }
```

Етап 4. Альфа-бета-відсікання.

Це метод оптимізації мінімакс-алгоритму, що дозволяє ігнорувати деякі гілки в дереві пошуку. І це дозволяє збільшити глибину пошуку, витрачаючи колишній обсяг ресурсів.

Альфа-бета-відсікання ґрунтується на ситуації, коли ми можемо зупинити оцінку певної гілки, якщо виявляється, що новий хід призведе до гіршої ситуації, ніж та, яку ми бачили при оцінці попереднього.

На результат мінімаксу оптимізація не впливає, але все починає працювати швидше.

Цей алгоритм набагато ефективніший у тому випадку, якщо спочатку перевірити шляхи, що ведуть до хороших ходів.

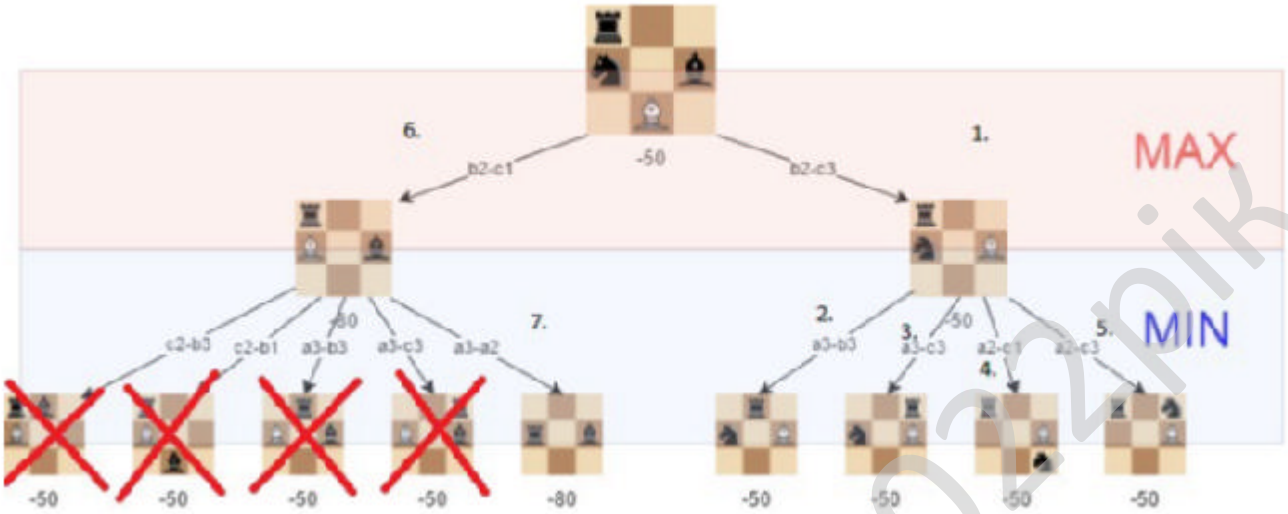

Рисунок 4.4 – Ілюстрація прикладу роботи алгоритму Альфа-бета-відсікання

Зображення демонструє ходи, які стають непотрібними у процесі використання альфа-бета-відсікання.

Як видно, з альфа-бета-відсіканням мінімакс оптимізується, і дуже значно.

Етап 5. Поліпшена функція оцінки.

Початкова функція оцінки досить проста, оскільки вона просто враховує оцінки фігур, що знаходяться на дошці. Для її оптимізації можна враховувати становище фігур. Наприклад, якщо розмістити коня у центрі дошки, він стає дорожче – спектр доступних ходів цієї фігури розшириться.

Етап 6. Застосування нейронної мережі.

Була використана багатошарова згорткова нейронна мережа. Мережа на вхід одержує оброблену шахову дошку як вхідні дані (матрицю 64 на 64). Вихідний прошарок мережі описує хід, який можна зіграти на шахівниці. Ми можемо розглядати стовпці як квадрати, з яких можна переставити фігуру, а рядки як квадрат, яким закінчиться хід. Звичайно, не всі ходи є допустимими в

позиції. Щоб отримати дозволені ходи, потрібно використати всі попередні етапи з попередніми алгоритмами.

Нейронна мережа навчається на всіх зіграних іграх у розробленому програмному забезпеченні. Алгоритм навчання нейронної мережі зображено на рис. 4.5.

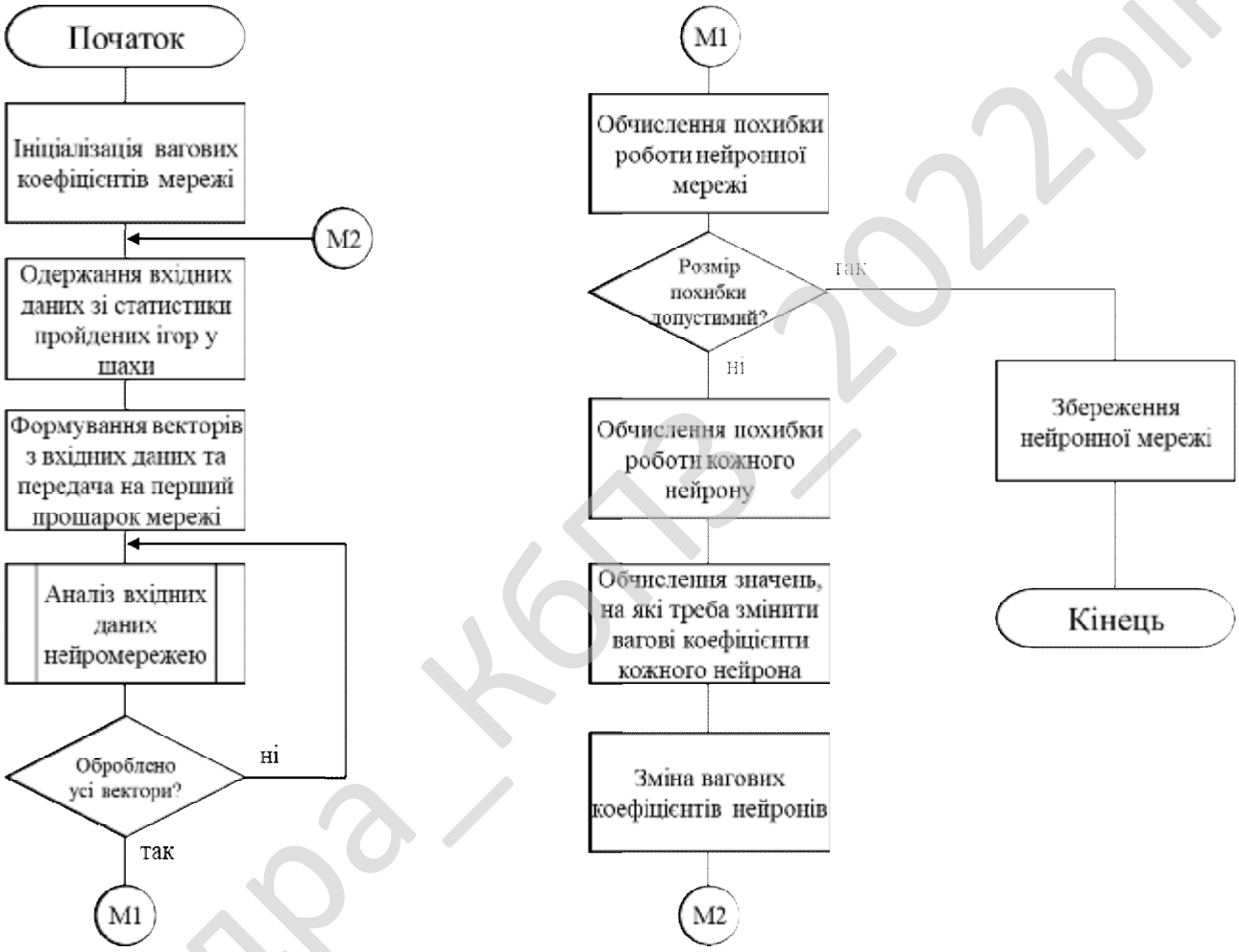

Рисунок 4.5 – Блок-схема алгоритму навчання нейронної мережі для вибору ходів у комп’ютерних шахах

Клас реалізації нейронної мережі наведено нижче.

```
public class NeuralNet
{
     public double LearnRate { get; set; }
     public double Momentum { get; set; }
     public List<Neuron> InputLayer { get; set; }
     public List<List<Neuron>> HiddenLayers { get; set; }
```

```
        public List<Neuron> OutputLayer { get; set; }

        private static readonly System.Random Random = new System.Random();

        public NeuralNet(int inputSize, int hiddenSize, int outputSize, int
numHiddenLayers = 1, double? learnRate = null, double? momentum = null)
        {
             LearnRate = learnRate ?? .4;
             Momentum = momentum ?? .9;
             InputLayer = new List<Neuron>();
             HiddenLayers = new List<List<Neuron>>();
             OutputLayer = new List<Neuron>();

             for (var i = 0; i < inputSize; i++)
                  InputLayer.Add(new Neuron());

        for (int i = 0; i < numHiddenLayers; i++)
        {
             HiddenLayers.Add(new List<Neuron>());
             for (var j = 0; j < hiddenSize; j++)
                  HiddenLayers[i].Add(new
Neuron(i==0?InputLayer:HiddenLayers[i-1]));
        }

             for (var i = 0; i < outputSize; i++)
                  OutputLayer.Add(new Neuron(HiddenLayers[numHiddenLayers-1]));
        }

        public void Train(List<DataSet> dataSets, int numEpochs)
        {
             for (var i = 0; i < numEpochs; i++)
             {
                  foreach (var dataSet in dataSets)
                  {
                       ForwardPropagate(dataSet.Values);
                       BackPropagate(dataSet.Targets);
                  }
             }
        }

        public void Train(List<DataSet> dataSets, double minimumError)
        {
             var error = 1.0;
             var numEpochs = 0;

             while (error > minimumError && numEpochs < int.MaxValue)
             {
                  var errors = new List<double>();
                  foreach (var dataSet in dataSets)
                  {
                       ForwardPropagate(dataSet.Values);
                       BackPropagate(dataSet.Targets);
                       errors.Add(CalculateError(dataSet.Targets));
                  }
                  error = errors.Average();
                  numEpochs++;
             }
        }

        private void ForwardPropagate(params double[] inputs)
        {
             var i = 0;
             InputLayer.ForEach(a => a.Value = inputs[i++]);
```

```
            foreach (var layer in HiddenLayers)
                layer.ForEach(a => a.CalculateValue());
            OutputLayer.ForEach(a => a.CalculateValue());
        }

        private void BackPropagate(params double[] targets)
        {
            var i = 0;
            OutputLayer.ForEach(a => a.CalculateGradient(targets[i++]));
            foreach(var layer in
HiddenLayers.AsEnumerable<List<Neuron>>().Reverse() )
            {
                layer.ForEach(a => a.CalculateGradient());
                layer.ForEach(a => a.UpdateWeights(LearnRate, Momentum));
            }
            OutputLayer.ForEach(a => a.UpdateWeights(LearnRate, Momentum));
        }

        public double[] Compute(params double[] inputs)
        {
            ForwardPropagate(inputs);
            return OutputLayer.Select(a => a.Value).ToArray();
        }

        private double CalculateError(params double[] targets)
        {
            var i = 0;
            return OutputLayer.Sum(a =>
Mathf.Abs((float)a.CalculateError(targets[i++])));
        }

        public static double GetRandom()
        {
            return 2 * Random.NextDouble() - 1;
        }
    }

    public enum TrainingType
    {
        Epoch,
        MinimumError
    }
```

Робота штучного нейрону реалізована наступним чином:

```
public class Neuron
 {
        public List<Synapse> InputSynapses { get; set; }
        public List<Synapse> OutputSynapses { get; set; }
        public double Bias { get; set; }
        public double BiasDelta { get; set; }
        public double Gradient { get; set; }
        public double Value { get; set; }

        public Neuron()
            {
                InputSynapses = new List<Synapse>();
                OutputSynapses = new List<Synapse>();
```

```
                Bias = NeuralNet.GetRandom();
            }

            public Neuron(IEnumerable<Neuron> inputNeurons) : this()
            {
                foreach (var inputNeuron in inputNeurons)
                {
                    var synapse = new Synapse(inputNeuron, this);
                    inputNeuron.OutputSynapses.Add(synapse);
                    InputSynapses.Add(synapse);
                }
            }

            public virtual double CalculateValue()
            {
                return Value = Sigmoid.Output(InputSynapses.Sum(a => a.Weight
* a.InputNeuron.Value) + Bias);
            }

            public double CalculateError(double target)
            {
                return target - Value;
            }

            public double CalculateGradient(double? target = null)
            {
                if(target == null)
                    return Gradient = OutputSynapses.Sum(a =>
a.OutputNeuron.Gradient * a.Weight) * Sigmoid.Derivative(Value);

                return Gradient = CalculateError(target.Value) *
Sigmoid.Derivative(Value);
            }

            public void UpdateWeights(double learnRate, double momentum)
            {
                var prevDelta = BiasDelta;
                BiasDelta = learnRate * Gradient;
                Bias += BiasDelta + momentum * prevDelta;

                foreach (var synapse in InputSynapses)
                {
                    prevDelta = synapse.WeightDelta;
                    synapse.WeightDelta = learnRate * Gradient *
synapse.InputNeuron.Value;
                    synapse.Weight += synapse.WeightDelta + momentum *
prevDelta;
                }
            }

    }

    public class Synapse
    {
        public Neuron InputNeuron { get; set; }
        public Neuron OutputNeuron { get; set; }
        public double Weight { get; set; }
        public double WeightDelta { get; set; }

        public Synapse(Neuron inputNeuron, Neuron outputNeuron)
        {
            InputNeuron = inputNeuron;
            OutputNeuron = outputNeuron;
```

```
                Weight = NeuralNet.GetRandom();
            }
        }

        public static class Sigmoid
        {
            public static double Output(double x)
            {
                return x < -45.0 ? 0.0 : x > 45.0 ? 1.0 : 1.0 / (1.0 +
    Mathf.Exp((float)-x));
            }

            public static double Derivative(double x)
            {
                return x * (1 - x);
            }
        }

        public class DataSet
        {
            public double[] Values { get; set; }
            public double[] Targets { get; set; }

            public DataSet(double[] values, double[] targets)
            {
                Values = values;
                Targets = targets;
            }
        }
```

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення вирішено поширювати по вільній ліцензії, тому воно не потребує захисту від несанкціонованого використання та поширення.

Ліцензування – це видача на певних умовах дозволів (ліцензій) на право здійснення певних операцій; передача прав однією особою іншій особі в обмін на гонорар, винагороду або ліцензійний платіж. Іншими словами це дозвіл на право здійснити якусь діяльність.

Різні громадські організації та автори по-різному описують вільні ліцензії, вкладаючи у них подібний, але не однаковий смисл. У преамбулі універсальної ліцензії GNU йдеться про надання дозволу використання твору будь-якими способами, зокрема, шляхом його поширення, переробки та дослідження вихідного тексту та надання всім бажаючим екземпляру вихідного тексту.

Обов’язок авторів похідних творів поширювати такий твір на тих самих умовах (“copyleft”) також включено до тексту GNU-ліцензії, але невід’ємною ознакою вільних програм для ЕОМ, на думку GNU, не є.

Подібні умови включені до ліцензії Attribution-ShareAlike 4.0. International (CCBY-SA 4.0) від CreativeCommons. Користувачам надається право використовувати твір самостійно, розповсюджувати його екземпляри та створювати переробки, але всі переробки мають поширюватись на тих самих умовах без будь-яких додаткових обмежень (“copyleft”). Однак, тут немає обов’язку правовласника надати доступ до вихідного тексту. Є Creative Commons і ще більш вільна ліцензія – заява про відмову автора від усіх авторських прав, але юридична доля такої заяви в багатьох країнах, сумнівна.

Ліцензія Массачусетського технологічного університету (Massachusetts Institute of Technology) одна з найкоротших за обсягом. Її умови зводяться до дозволу користувачам використовувати твір у будь-який спосіб, у тому числі шляхом переробки. Але обмежень на ліцензійні умови щодо перероблених творів немає, як і обов’язку відкрити доступ до вихідного коду. MIT-License була розрахована лише на програми для ЕОМ.

Існує безліч інших варіантів та визначень вільних ліцензій.

Вільна ліцензія – це ліцензійний договір, за яким правовласник безоплатно надає користувачеві дозвіл на використання твору будь-якими способами, у тому числі шляхом поширення та створення похідних творів. Типовою, але не обов’язковою умовою такого виду ліцензій є обов’язок автора надати доступ до вихідного тексту програми та обов’язок автора похідних творів поширювати такий твір на умовах ліцензії на оригінальну роботу. Крім того, часто включається умова про те, що ліцензійний договір не може бути розірваний ліцензіаром в односторонньому порядку (безвідкликаність). Обов’язковість останніх умов визнання ліцензійного договору вільним залежить від погляду тієї чи іншої громадської організації чи автора.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблене програмне забезпечення представлене у вигляді відкритого сирцевого коду на C# у формі проекту для рушія Unity. Для впровадження розробленої системи в експлуатацію можна відкомпілювати її для будь-якої платформи, що підтримується у Unity.

Шахи часто реалізуються у вигляді простої 2D-ігри. Однак у цій роботі в 3D імітуєтьсяо гравець, який сидить за столом і грає зі своїм другом.

Відкрийте сцену Main із папки Scenes, ви побачите об’єкт Board, що представляє собою ігрову дошку, та об’єкт для GameManager. До цих об’єктів прикріплено скрипти на C#.

Наповнення проекту на Unity представлене наступними елементами:

– Prefabs: тут містяться дошка, окремі фігури та квадрати-індикатори для виділення клітин, які використовуються у процесі вибору ходу.

– Materials: тут знаходяться матеріали для шахівниці, фігур і клітин.

– Scripts: містить компоненти, які вже прикріплені до об’єктів ієрархії.

– Board: контролює візуальне відображення фігур. Цей компонент відповідає за виділення окремих фігур.

– Geometry.cs: допоміжний клас, який керує перетвореннями між записом рядів/стовпців і точок Vector3.

– Player.cs: контролює фігури гравця та взяті гравцем фігури. Крім того, містить напрямок руху фігур, для яких важливий напрямок, наприклад, для пішаків.

– Piece.cs: базовий клас, який визначає перерахування для всіх екземплярів фігур. Також містить логіку визначення допустимих ходів у грі.

– GameManager.cs: зберігає ігрову логіку, таку як допустимі ходи, вихідне розташування фігур на початку гри та інше. Це синглтон, тому іншим класам його зручно викликати.

На рисунку 5.1 наведено вікно програмного забезпечення під час безпосередньо гри у шахи.

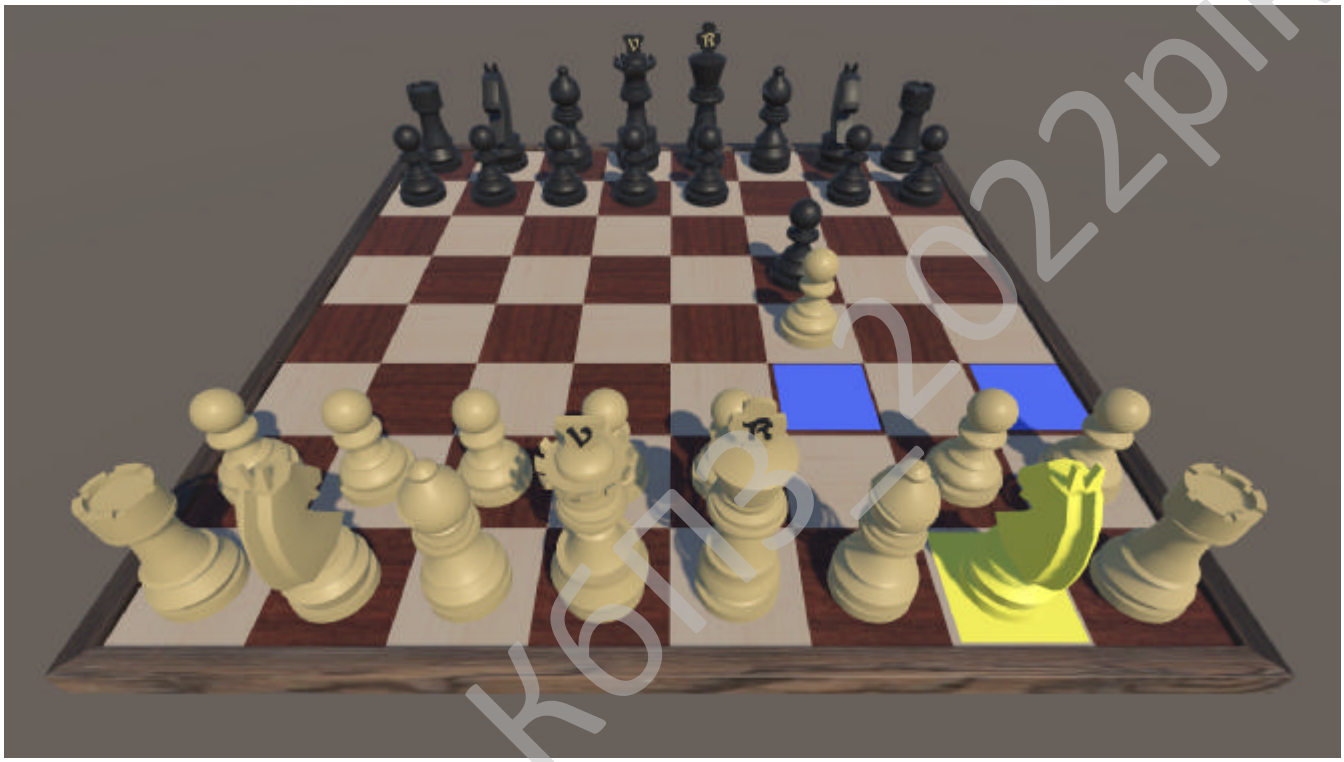

Рисунок 5.1 – Головне вікно програмного забезпечення (хід гри у шахи)

6 НАУКОВА НОВИЗНА

Метою роботи є дослідження та програмна реалізація системи штучного інтелекту для комп’ютерної гри на ігровому рушії Unity.

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

– Дослідження існуючих систем штучного інтелекту для прийняття рішень в комп’ютерних іграх у шахи.

– Розробка методів та алгоритмів системи штучного інтелекту для комп’ютерної гри в шахи на ігровому рушії Unity.

– Програмна реалізація комп’ютерної гри в шахи на ігровому рушії Unity та системи штучного інтелекту для неї.

Об’єктом дослідження є процес прийняття рішень в інтелектуальних системах шахових комп’ютерних програм.

Предметом дослідження є методи та алгоритми штучного інтелекту для знаходження вірних ходів у грі в шахи при змаганні комп’ютера проти людини, а також методи та алгоритми візуалізації гри у шахи на рушії Unity.

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

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

1. Удосконалено метод реалізації та використання штучного інтелекту для гри в комп’ютерні шахи та адаптовано його для реалізації на рушії Unity.

2. Розроблено вітчизняний продукт реалізації штучного інтелекту для комп’ютерної гри в шахи, реалізований на ігровому рушії Unity, який орієнтований на отримання більш цікавого, корисного та пізнавального ігрового досвіду.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми магістерської роботи

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 24 днів (один місяць).

В магістерській роботі було проведено дослідження та виконана програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity.

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

а) невеликий розмір;

б) невеликі системні потреби;

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

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	30
3. Запланований термін розробки, днів	Fpq	24 (1 місяць)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Г
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	8
8. Кількість форм вихідної інформації.	–	6
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	1
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	3
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	4
17. Складність кінцевого програмного продукту (1-6)	–	5
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	3
20. Вимоги до швидкодії ПП (1-6)	–	3
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	4
23. Професійний рівень аналітиків (1-6)	–	3
24. Професійний рівень програмістів (1-6)	–	4
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	1
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	3
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	30000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	40
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

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

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

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

де: A – коефіцієнт Боема, A = 2,45;

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

B – показник ступеня, що визначається співвідношенням:

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

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

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+4{,}22+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}2^{1{,}027} = 5{,}5 \text{ люд-міс.}$$

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

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

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

$$T_{уточн} = 5{,}5 \cdot (1 \cdot 1{,}09 \cdot 1{,}30 \cdot 0{,}91 \cdot 1 \cdot 1 \cdot 1 \cdot 1{,}15 \cdot 1 \cdot 0{,}87 \cdot 1{,}10 \cdot$$
$$\cdot 1{,}22 \cdot 1{,}12 \cdot 1{,}10 \cdot 1 \cdot 1 \cdot 1{,}10) = 12{,}9 \text{ люд-міс.}$$

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

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

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

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

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

$$T_{РП} = 0{,}3 \cdot 3{,}23 \cdot 12{,}9^{0{,}33+0{,}2(1{,}027-1{,}01)} \cdot 30 = 68 \text{ люд/день.}$$

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

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	15	Д7
Робочий проект	68	Ф 7.1-7.4
Впровадження	15	Д13
Всього	117	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

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

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

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

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

$$Ч = \frac{117 \cdot 1}{24\text{-}3} = 5,6 \text{ ставки.}$$

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

робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

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

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	8	720	12
Монітор	60	8	480	8
Клавіатура	30	8	240	4
Маніпулятор «мишка»	30	8	240	4
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	1	120	2
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор– маршрутизатор	30	2	60	1
Кабельні господарства ЛВС на 1 м. п.	2,5	250	625	10,42
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	45,08

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

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

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

$$Ф_{др}^{с} = \frac{45 \cdot 1}{1,2} = 37,5 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

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

$$Ч_{ел} = 37{,}5/(24 \cdot 8) = 0{,}2 \text{ ставки.}$$

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

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

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2008 R2, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN, PPPoE, Frame Relay, Wi-Fi	0,4	0,1
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,2	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,1	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	0,1	
Всього		0,8	

Продовження таблиці 7.4

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

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	0,25	7084	1771
Продакт-менеджер	0,2	6750	1350
Інженер-програміст	5,6	6700	37520
Інженер-електронщик	0,2	6750	1350
Інженер-системотехнік	0,1	6750	675
Адміністратор мережі	0,1	6750	675
Дизайнер WEB	0,1	6750	675
Всього за період розробки	$R_{сп}$ = 6,55	-	$Ф_{роб}$ = 44016

Розрахуємо середньоденну зарплату одного виконавця:

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

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

$$З_{сд} = \frac{44016}{6,55 \cdot 24} = 280 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

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

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

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

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

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

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 13 \cdot 8 \cdot 20000 = 2080000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 208000 грн. Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

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

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

$$I_{нв} = 13 \cdot 3500 = 45500 \text{ грн.}$$

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

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Компбест за 20.11.22 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп'ютер		11457
Системний блок		7509
Процесор	Intel Core i7-4790 (4(8) ядра по 3.6 - 4.0 GHz); Cache Memory 8 MB	-

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Системна плата	1st Player ATX NEW	-
Відеокарта	AMD Radeon HD 8570, 1 GB GDDR3, 128-bit / LowProfile / DP, DVI	-
Жорсткий диск	SSD: 480 Gb	-
Оперативна пам’ять	Kingston DDR3 4GB (KVR1333D3N9) Intel/AMD 2 модулі	-
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bulk 22x, SecurDisc, black	-
Корпус	ATX Middle Tower GIGABYTE GZ-X4 Silver 500W (GZ-X4 Silver)	-
Кулер	–	-
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	-
інше	Клавіатура, мишка	Подарунок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1, 170/160, D-SUB, Wide)	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37 Photo	2970
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійн. живлення	UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS)	1348

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

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	13	11457	14894,1	163835,1
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2970	297	3267
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	185653,6

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відраху-вання, грн.
1	2	3	4
Група 3			
1. Будівлі	2080000	-	-
2. Передавальні пристрої	208000	-	-
Всього по групі	2288000	5	114400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	185654	-	-
Всього по групі	185654	50	92827
Група 5, 6			
4. Вимірювальні пристрої	3999	25	-
5. Транспортні засоби	0	20	-
6. Господарський інвентар	45500	25	-
Всього по групі	49499	-	12374,75
7. Нематеріальні активи	30000	10	3000
Разом	$К_р$ = 2553153		$А_р$ = 222602

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

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

де: N_e – кількість екземплярів програм, шт.

$$З_о = 182 \cdot 117/30 = 1092 \text{ грн.}$$

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

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

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

$$З_д = 1092 \cdot 10 \cdot 0{,}01 = 109{,}2 \text{ грн.}$$

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

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

де: H_c – відрахування на соціальні потреби, %.

$$C_{оц} = 0{,}01 \cdot 22(1092+109{,}2) = 444 \text{ грн.}$$

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

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

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 1092 \cdot 15 \cdot 0{,}01 = 246 \text{ грн.}$$

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

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

де: $З_{М1}$ – вартість паперу, грн.;

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

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

$N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,2 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки:

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

$$З_{М1} = 200 \cdot 1 \cdot 0{,}2 = 40 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо 3):

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR box – 21,5 грн./шт., DVD-R box – 31 грн./шт.

$$З_{М2} = 31 \cdot 3 = 93 \text{ грн.}$$

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

$$З_{МЗ} = \sum Ц_{з}., \quad (7.18)$$

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

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

$$З_{М} = (40 + 93 + 1702)/30 = 61 \text{ грн.}$$

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

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

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

$$О_п = 1092 \cdot 15 \cdot 0{,}01 = 164 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 30 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \quad (7.20)$$

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

$$А_м = 222602 \cdot 1/(30 \cdot 12) = 618 \text{ грн.}$$

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

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

$$С_п = 1092+109{,}2+444+164+61+164+618 = 2652 \text{ грн.}$$

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

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1	2	3
1. Основна зарплата виконавців	$З_о$	1092
2. Додаткова зарплата виконавців	$З_д$	109,2
3. Відрахування на соціальні потреби	$С_{оц}$	444
4. Загальногосподарські витрати	$Г_{осп}$	164
5. Витрати на матеріали	$З_М$	61
6. Освоєння нових операційних систем, мов програмування	$О_п$	164
7. Амортизація основних фондів	$А_м$	618
8. Повна собівартість програмного забезпечення	$С_п$	2652
9. Плановий прибуток	$П_р$	1061
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	3713
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	743
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	4456

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

Для даного програмного забезпечення рівень рентабельності складає 40%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \qquad (7.22)$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 40 \cdot 2652 = 1061 \text{ грн.}$$

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.9.

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

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	4456
Всього капітальних витрат	–	4456

7.7 Визначення експлуатаційних витрат

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

Витрати на профілактичні роботи:

$$З_р = T_р \cdot З_г \cdot (1 + 0{,}01 \cdot H_q) \cdot (1 + 0{,}01 \cdot H_c), \qquad (7.23)$$

де: $T_р$ – кількість годин обслуговування кожного комп’ютера за рік, *год.*;

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

Після купівлі нового програмного забезпечення кількість профілактичних годин робіт зменшилася з 240 годин на рік до 150 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\,баз} = 240 \cdot 60 \cdot 1{,}1 \cdot 1{,}22 = 19325 \; грн,$$

до:

$$З_{р\,нов} = 150 \cdot 60 \cdot 1{,}1 \cdot 1{,}22 = 12078 \; грн.$$

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	19325	12078
2. Витрати на електроенергію	$З_{ел}$	22680	22050
3. Витрати на амортизацію	$З_{ам}$	0	1114
Всього витрат за рік	$І$	42005	35242

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

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

$$З_{ел\ баз} = 10 \cdot 0{,}15 \cdot 7200 \cdot 2{,}1 = 22680 \text{ грн.}$$

$$З_{ел\ нов} = 10 \cdot 0{,}15 \cdot 7000 \cdot 2{,}1 = 22050 \text{ грн.}$$

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

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн., за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	4456	–	1114
Всього відрахувань	-	–	4456	–	1114

7.8 Визначення економічної ефективності програмної продукції

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

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

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (3713 - 2652) \cdot 30 - (0,05 \cdot 2288000 + 0,5 \cdot 185654 +$$
$$+0,25 \cdot 49499 + 0,1 \cdot 30000) \cdot 1/12 = 13280 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника без врахування вартості ОФ третьої групи, так як їх строк служби на порядок більший ніж період розробки ПЗ.

$$T_в = \frac{265153}{(3713 - 2652) \cdot 30 \cdot 12 / 1} = 0,7 \text{ року}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

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

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

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

$$E_{сп} = (42005\text{-}35242)\text{-}0,25 \cdot 4456 = 5649 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у

споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{К_н - К_б}{І_б - І_н}, \quad (7.28)$$

$$T_{сп} = \frac{4456}{42005 - 35242} = 0,7 \ року.$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	30
2. Повна собівартість розробленої програми	Грн.	2652
3. Ціна розробленої програми	Грн.	3713
4. Плановий прибуток від реалізації розробленої програми	Грн.	1061
5. Рентабельність програмної продукції	%	40
6. Об’єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2553153
7. Загальний прибуток від реалізації програмної продукції	Грн.	31830
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	13280
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	0,7
10.Об’єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	4456
11.Величина економічного ефекту у користувача програмної продукції	Грн.	5649
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,7

7.9 Висновки

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

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Ознакою сучасного науково-технічного прогресу є масове впровадження комп'ютерних технологій в усіх сферах життя і діяльності людини. Застосування персональних комп'ютерів і ЕОМ дозволило значно підвищити продуктивність праці, змінити характер і зміст праці.

Впровадження комп'ютерних технологій принципово змінило характер праці різних категорій фахівців. Працівники, використовують комп'ютерну техніку, на своєму досвіді оцінили її величезні можливості. Одночасно виникла певна безтурботність при її експлуатації.

До можливих недоліків умов праці користувачів комп'ютерної техніки можна віднести:

- недостатню площу і обсяг виробничого приміщення;
- недотримання вимог, мікроклімату на робочих місцях;
- низький рівень освітленості у приміщеннях і на робочих поверхнях апаратури;
- підвищений рівень низькочастотних магнітних полів від моніторів;
- порушення вимог організації робочих місць;
- недотримання вимог до режимам праці та відпочинку;
- надмірне виробничу навантаження працівників;
- відсутність навичок зниження впливу психоемоційного напруги.

Відповідно до ст.14 Закону «Про охорони праці» [53] на роботодавця покладено обов'язок забезпечити: безпеку працівників при експлуатації устаткування; застосування коштів індивідуальної захисту працівників; відповідні вимоги охорони праці, умови праці в кожному робоче місце; дотримання режиму праці та відпочинку працівників; навчання безпечним методам і прийомам

виконання; інструктаж з охорони праці; організацію контролю над станом умов праці в робочих місць; проведення атестації робочих місць в умовах праці.

Програмісти в процесі роботи знаходяться під дією електромагнітного випромінювання, нервово-емоційної напруженості, високого інтелектуального навантаження, монотонності праці, шуму, статичного навантаження на кістково-м'язовий апарат, штучного освітлення, ризику виникнення пожежі, ризику ураження електромагнітним струмом. Ці шкідливі фактори можуть привести до професійних захворювань.

Запобігти цьому можливо методом аналізу умов праці програміста. При цьому розглядають конкретне місце роботи програміста аналізують загальні його характеристики – розміру приміщення, температуру, шум, потужність тощо. Оцінюють небезпеки та шкідливості, встановлюють причини імовірних професійних захворювань. Шляхом аналізу небезпечних і шкідливих факторів розробляють заходи з охорони праці. До них обов'язково включать вентиляцію приміщення, його освітлення, параметри повітряного середовища та інше. На підставі таких досліджень організують робоче місце програміста. Необхідно щоб робоче місце програміста відповідало усім вимогам до нього, враховуючи розроблені заходи і діюче законодавство України.

Максимально зменшити кількість шкідливих впливів на людину при високій продуктивності праці, створити комфортні умови для роботи людей – ось одна з головних задач охорони праці.

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

Електронно-обчислювальни машини (ЕОМ) та інше обладнання є джерелами небезпеки ураження електричним струмом.

На робочому місці програміста виникають небезпечні та шкідливі фактори: підвищений рівень шуму, несприятливі мікрокліматичні умови, недостатній рівень освітленості, шкідливі речовини, підвищений рівень

електромагнітних випромінювань радіочастот, висока напруга електричної мережі, статична електрика та інші.

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

- ризик виникнення надзвичайних ситуацій природного або штучного характеру на об'єкті або території.

- ризик виникнення пожежі;

- негативний вплив на органи зору людини;

- ризики ураження електричним струмом;

- недостатня, або надмірна освітленість робочого місця;

- електромагнітні (у т.ч. високочастотні) електромагнітні випромінювання (коливання);

- несприятливі мікрокліматичні умови;

- нервово-емоційна напруженість праці;

- інтелектуальні навантаження;

- монотонність праці;

- невідповідність ергономічних показників робочого місця діючим вимогам;

- шум;

- статичні навантаження на кістково-м'язовий апарат.

Вивчення умов праці на робочому місці програміста є необхідною умовою запобігання впливу небезпечних та шкідливих факторів на організм персоналу.

Дослідження санітарно-гігієнічних умов у приміщенні звичайно проводять з використанням пристроїв вимірювання параметрів мікроклімату – вологи повітря (психрометр), освітленості (люксметр) та шуму (шумомір), а також відомих розрахункових методик.

8.3. Аналіз умов праці на робочому місці програміста

Робота програміста пов'язана з постійною роботою на ЕОМ, яка відбувається у кімнаті розмірами 3,8 м×5,2 м×2,9 м. Одна з її більших стін має шість двостулкових вікон, розмірами 1,4 м×1,9 м, які виходять на південний захід. Вікна розташовані рівномірно по всій довжині стіни. Підлога в кімнаті покрита леноліумом, всі стіни пофарбовані світло синього кольору до висоти 2,8 м, а далі розташована підвісна стеля. Уздовж стін розташовані комп'ютерні столи. На них розташовуються 2 персональні комп'ютери й інша оргтехніка (сканер, принтери). Столи мають пластикове покриття. Габарити їхньої робочої поверхні 1230 мм×845 мм. Висота столів 755 мм. Висота стільців від рівня підлоги становить 420 мм.

Згідно НПАОП 0.00 – 1.28 – 10 «Правила охорони праці під час електронно-обчислювальних машин» площа повинна задовольняти умові - не менш 6 $м^2$ на одне робоче місце. Кратність повітрообміну в приміщенні вузла також регламентується ДСанПіН 3.3.2.007-98 [52], вона повинна становити 20 $м^3$/годину на одне місце. Виконання даних вимог забезпечить підтримку в приміщенні вузла оптимального значення вологості й складу повітря.

Відповідно ДБН В.2.5 – 28 – 2006 [51] роботу програміста можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення вузла можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при сполученому висвітленні), повинен становити 0,5%, освітленість при штучному висвітленні повинна становити 300 лк.

За результатами виміру освітленості відділом охорони праці величина освітленості від системи загального штучного висвітлення лежить у межах 200-250 лк, що не відповідає вимогам, які пред'являються до приміщення.

Відповідно ДСанПіН 3.3.2.007-98 [52] рівні звукового тиску в робочому приміщенні не повинні перевищувати в октавних смугах із середньо геометричними частотами наступних значень, наведених у таблиці 8.1.

Таблиця 8.1 – Допустимі спектри рівнів звукового тиску

Робоче місце	**Рівень звукового тиску, дБ, в октавних смугах із середньогеометричними частотами, Гц**								**Рівень звуку і еквівалентний рівень звуку, дБА**
	63	125	250	500	1000	2000	4000	8000	
Приміщення конструкторських бюро, програмістів обчислювальних машин, лабораторій для теоретичних робіт і опрацювання експериментальних даних, прийому хворих в медпунктах	71	61	54	49	45	42	40	38	50

У приміщенні перебувають наступні джерела шуму: електродвигуни внутрішнього вентилятора ЕОМ; працюючі принтери; працюючі дисководи. Шум, вироблений вентилятором можна класифікувати як постійний, всі інші джерела шуму, як імпульсні. Відповідно паспорта на приміщення рівень звуку, Дб(А), обмірюваний за шкалою (А) шумоміра досяг величини 28,3 Дб(А) при роботі всього встаткування вузла, включаючи й ксерокс. Це дозволяє зробити висновок про відповідність рівня звуку в приміщенні вимогам нормативних актів.

Ергономічні вимоги до робочого місця працюючого з ВДТ ЕОМ і ПЕОМ нормуються НПАОП 0.00 – 1.28 – 10**.** Оптимальне положення тіла того, що

працює забезпечується відповідною конструкцією робочого місця, а також регуляцією висоти робочої поверхні, сидіння, простори й підставки для ніг. Даного місця програміста не мають регульованих параметрів. Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту дані в таблиці 8.2.

Таблиця 8.2 – Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту

Ріст людини, см	**Висота робочої поверхні мм,**	**Висота простору для ніг, мм**	**Висота робочого сидіння, мм**
175	765(740)	655(600)	450(440)

У дужках зазначені реальні значення параметрів робочого місця; всі вони не відповідають параметрам, зазначеним у стандарті.

Параметри мікроклімату можуть мінятися в широких межах, тоді як необхідною умовою життєдіяльності людини є підтримка сталості температури тіла завдяки властивості терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище.

У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах ДСН 3.3.6.042 - 99 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 8.3).

Таблиця 8.3 - Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	**Параметр мікроклімату**	**Величина**
Холодний	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	22 - 24°С 40 - 60% до 0,1 м/с
Теплий	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	23 - 25°С 40 ... 60% 0,1 ... 0,2 м / с

8.4. Розрахункова частина

Для захисного штучного заземлення застосовуються ветикальні електроди: метелевий куток *80·50·6 мм*., (згідно з ДСТУ 8769:2018 «Кутики сталеві гарячекатані нерівнополичні. Сортамент») довжиною *L=1,7 м.,* та горизонтальний електрод — метелева полоса з перетином *60·5 мм*. Напруга — *220/380 В*. Розрахункова схема розташування заземлюючих електродів — у ряд.

Розрахунок проведем за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта — чорнозем, нижнього шару грунта — глина (питомий опір ρ_2 *= 40 Ом·м)*. Умовна товщина верхнього шару грунта: *H=0,5 м*. Відстань між вертикальними заземлювачами (електродами) *A=1,7 м.* Глибина закладення горизонтального контура заземлення *t=0,6 м*. Опір заземлювача, який нормується: $R_{ЗН}$ *= 4 Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,6+1,7/2=1,45\ м.$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi\rho = 1,36 \cdot 40 = 54,5\ Ом \cdot м.$$

де ψ *= 1,36* - табличне значення коефіцієнта сезонності для відповідной климатичной зони у багатошаровому грунті [62]; ρ_2*= 40 Ом·м*. - табличне значення питомого опору нижнього шару грунта (глина) [62].

Еквівалентний діаметр ветикальньного електрода (кутка) [62]:

$$Dв=0,95 \cdot K= 0,95 \cdot 65 = 59,85\ мм. = 0,62\ м.$$

де *K= (80+50)/2=65 мм.* - середній розмір полиці метелевого кутка.

Відношення *A/L=1,7/1,7=1*.

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [62]:

$$R_O=0,366(\rho/L)[lg(2L/Dв)+(1/2)lg((4T+L)/(4T-L))]=$$

$$=0,366(54,5/1,7)[(lg(2\cdot1,7/0,62)+(1/2\ )lg((4\cdot1,45+1,7)/(4\cdot1,45-1,7))]=$$

$$=21,9\ Ом.$$

Визначаємо коефіцієнт екранування ветикальньних електродів $Кев=0,8$ при орієтовній кількості ветикальньних електродів, яке дорівнює 5 [62].

Визначаємо необхідну кількість вертикальних електродів заземлювача (без вхарування горизонтального заземлювача), при $R_{ЗН}$ = 4 *Ом*:

$$N=R_O\ /(Кев\ R_{ЗН})=\ 21,9\ /\ (0,8\cdot4)=\ 6,85\approx 7\ шт.$$

Визначаємо длину з'єднуючої полоси:

$$L_П\ =1,05\cdot A\cdot N=\ 1,05\cdot1,7\cdot7=12,22\approx 12\ м.$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$ [62]:

$$R_П\ =0,366(\rho\cdot K_П/L_П\ )lg(2\cdot L_П{}^2\ /(B\cdot t))=$$

$$=0,366(40\cdot5/12)\cdot lg((2\cdot12\ ^2\ )/(0,06\cdot0,6\ ))=23,5\ Ом.$$

де $K_П=5$ - табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної климатичной зони для з'єднуючої полоси [62]:

$B = 60\ мм. = 0,06\ м.$ – ширина з'єднуючої полоси (задана).

Загальньний опір розтіканню електричного струму заземлювача [62]:

$$R\ =(R_O\cdot R_П)/(\ R_O\cdot\eta_П+\ N\cdot R_П\cdot Кев)=$$

$$=(21,9\cdot\ 23,5\ )/(21,9\cdot\ 0,75+\ 9\cdot\ 23,5\ \cdot\ 0,8)=3,48\ Ом.$$

де $\eta_П = 0,75$ - табличне значення коефіцієнта екранування з'єднуючої полоси [62].

Умова $R \le R_{ЗН}$ виконується ($3,48 \le 4$).

Так як при 7 вертикальних електродах R суттево менше $R_{ЗН}$, зменшимо кількість вертикальних електродів N до *6* і виконаємо перерахунок. У результаті остаточно отримали: $R =3,98\ Ом.$ при кількості вертикальних електродів $N=6$.

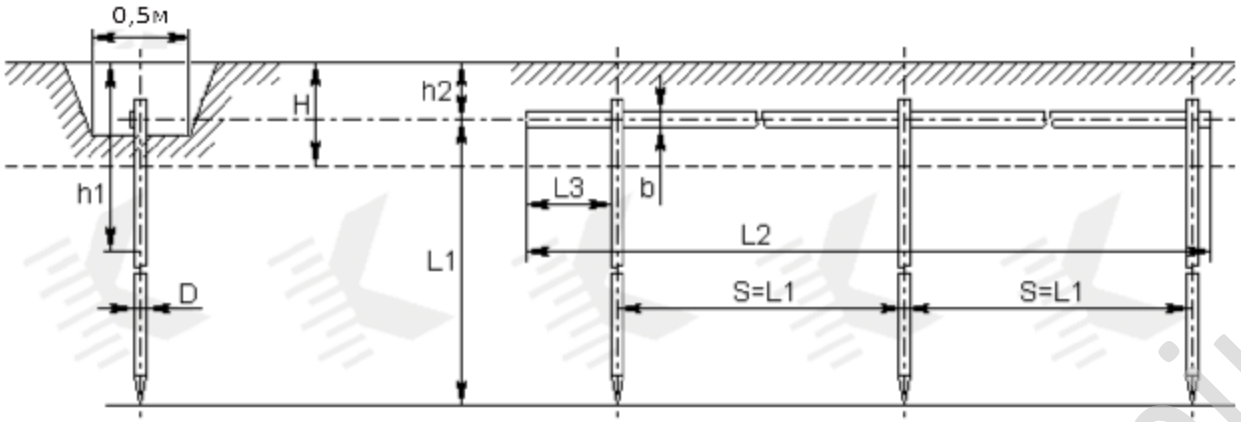

Рисунок 8.1 – Схема штучного заземлення

8.5 Висновки до розділу

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

З цих міркувань було здійснено аналіз умов праці на робочому місці програміста, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання магістерської роботи, призначено для реалізації системи штучного інтелекту для комп'ютерної гри на рушії Unity.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У магістерській роботі наведені теоретичне узагальнення й рішення наукового завдання дослідження методів штучного інтелекту для комп'ютерної гри у шахи.

Рішення даного завдання полягало у вирішенні наступних задач:

– Було проведене дослідження існуючих систем штучного інтелекту для комп'ютерної гри у шахи та методи реалізації комп'ютерних ігор на рушії Unity.

– На основі проведеного дослідження розроблено методи та алгоритми для системи штучного інтелекту та комп'ютерної гри у шахи на рушії Unity.

– Використовуючи розроблені методи та алгоритми, створена програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity.

Розроблені під час виконання магістерської роботи алгоритми дозволяють успішно вирішувати завдання реалізації комп'ютерної шахової гри зі штучним інтелектом на рушії Unity.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

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

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

Програма реалізована на мові високого рівня C# у середовищі розробки Unity. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Розроблене програмне забезпечення вирішено поширювати по вільній ліцензії, тому воно не потребує захисту від несанкціонованого використання та поширення.

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

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 5649 грн. З урахуванням вартості розробки програми та обладнання, строк окупності становить 0,7 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Довідник по C # | Microsoft Docs, Ключові слова C #, Директиви препроцесора, Параметри компілятора C #

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

3. Седерхольм, Д. Пуленепробиваемый дизайн. Библиотека специалиста / Д. Седерхольм. - СПб.: Питер, 2012. - 304 с.

4. Джозеф Хокинг, Unity в дії. Глава 10 Звукові ефекти та музика 242с.

5. Джозеф Хокинг, Unity в дії. Глава 11 Об'єднання фрагментів в готову гру 267с.

6. Джозеф Хокинг, Unity в дії. Глава 12 Розгортання ігор на пристроях гравців 298с.

7. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, глава 5. Рівень архітектури команд 334с.

8. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Глава 8. Архітектури комп'ютерів паралельної дії 556с.

9. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток А. Двійкові числа 663с.

10. Е. Таненбаум, Архітектура комп'ютерів 4 видання, Москва, Санкт-Петербург, Кії, Харків, Мінськ 2003, Додаток Б. Числа з плаваючою точкою 674с.

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

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

13. Гудфеллоу Я.Бенджіо І.Курвілль А., Глибоке навчання 2017р. Глава 5.

Основи машинного навчання 96с.

14. Бхаргава А. Грокаем алгоритми. (Пітер)

15. Кормен Томас Х., Лейзерсон Чарльз І., Ривест Рональд Л., Штайн Кліффорд, Алгоритми. Побудова і аналіз

16. «Плоский дизайн»: с чего начать? Пять основных принципов Flat дизайна [Електронний ресурс] – Режим доступу до ресурсу: http://powerbranding.ru/design/flat-design-june13/.

17. «Альтернативы для замены ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://ruprogi.ru/software/arcgisdesktop.

18. «.NET Framework Guide» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/.

19. «Common Language Runtime (CLR) overview» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/clr.

20. Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – Киев: Питер, 2008. – 656 с. – (2).

21. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: вильямс, 2014. – 1312 с. – (6).

22. «Visual Studio» [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/.

23. «Windows Forms overview» [Електронний ресурс] – 10. Режим доступу до ресурсу: https://docs.microsoft.com/enus/dotnet/framework/winforms/windows-forms-overview.

24. «Entity Framework Documentation» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/.

25. «SQL Server 2017» [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/ru-ru/sql-server/sql-server-2017.

26. «About ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://www.esri.com/ru-ru/arcgis/about-arcgis/overview.

27. «Understanding Onion Architecture» [Електронний ресурс] – Режим

доступу до ресурсу: https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understandingonion-architecture.html.

28. «Концептуальная модель системы» [Електронний ресурс] – Режим доступу до ресурсу: http://studepedia.org/index.php?vol=1&post=2123.

29. «MVC, MVP and MVVM Design Pattern» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvmdesign-pattern-6e169567bbad.

30. «UnitOfWork And Repository Pattern » [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern305cd8ecfa7a.

31. «What is ArcGIS, and where is it available at IU?» [Електронний ресурс] – Режим доступу до ресурсу: https://kb.iu.edu/d/avzt.

32. Кузін А.В «Базы данных, 5-е издание» / Кузін А.В., Левонисова С.В. – К. : «Академия», 2012. – 317 с.

33. Гольцман В.І. «MySQL 5.0. Библиотека программиста» / Гольцман В.І. – К. : «Питер», 2010. – 253 с.

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

35. Sedgewick, Robert. Algorithms in C++. Parts 5: Graph Algorithms. 3rd Ed. Addison-Wesley, 2002.

36. Роберт Седжвік. Фундаментальные алгоритмы на C++. Части 1-4: Анализ/Структуры данных/Сортировка/Поиск. - К.: Издательство ДиаСофт?, 2001.

37. Роберт Седжвік. Фундаментальные алгоритмы на C++. Часть 5: Алгоритмы на графах. - К.: Издательство ДиаСофт?, 2002.

38. Роберт Седжвік. Фундаментальные алгоритмы на C. Части 1-4: Анализ/Структуры данных/Сортировка/Поиск. - К.: Издательство ДиаСофт?, 2003.

39. Роберт Седжвік. Фундаментальные алгоритмы на C Часть 5:

Алгоритмы на графах. - К.: Издательство ДиаСофт?, 2003.

40. Джон Макгрегор, Девід Сайке. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие. - К.: Издательство ДиаСофт?, 2002.

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

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

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

44. L"Ecuyer, Pierre. "Efficient and Portable Combined Random Number Generators." Communications of the ACM, Vol. 31 (1988), pp. 742-749, 774.

45. Nelson, Mark. The Data Compression Book. M& T Publishing, 1991.

46. Park, S.K., and K.W. Miller. "Random Number Generators: Good Ones are Hard to Find." Communications of the ACM, vol. 31 (1988), pp. 1192-1201.

47. Pham, Thuan Q. and Pankaj K. Garg. Multithreaded Programming with Win32. Prentice Hall, 1999.

48. Pugh, William. "Skip Lists: A Probabilistic AItemative to Balanced Trees." Communications of the ACM, Vol. 33 (1990), pp. 668-676.

49. Robbins, John. Debugging Applications. Microsoft Press, 2000.

50. Wood, Derick. Data Structures, Algorithms, and Performance. Addison-Wesley, 1993.

51. Державні будівельні норми України: ДБН В.2.5-28:2018. - Режим доступу до ресурсу: https://goo.su/9AkQ (дата звернення 19.10.22).

52. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98 (дата звернення 19.10.22).

53. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. -

Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення 19.10.22).

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

55. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». - Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508 (дата звернення 19.10.22).

56. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. - Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення 19.09.22).

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

58. Оришака О.В. Охорона праці в галузі та цивільний захист / О.В Оришака, Г.П. Горбачова, О.М. Мезенцева , К.М. Марченко, К.О. Буравченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький: Видавець Лисенко В.Ф., 2019. − 226 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/9258 (дата звернення 19.09.22).

59. Методичні рекомендації до виконання розділу "Заходи з охорони праці та техніки безпеки" випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти для здобувачів вищої освіти спеціальностей 123 "Комп'ютерна інженерія" та 122 "Комп'ютерні науки" / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та програм. забезпечення; [укл. О.В. Оришака, К.М. Марченко]. - Кропивницький: ЦНТУ, 2022. — 19 с. [Електронний ресурс]. – Режим доступу:

http://dspace.kntu.kr.ua/jspui/handle/123456789/12240 (дата звернення 19.09.22).

60. Охорона праці. Ч. 1. Захисне заземлення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд. ; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. - Кіровоград : КІСМ, 1997. - 20 с. – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358 (дата звернення 19.09.22).

61. Охорона праці. Ч. 2. Занулення : метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ−сумісного типу / [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака, А. Е. Солових, С. Е. Катеринич] ; Мін-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - 2−ге вид., перероб. та доп. - Кропивницький : ЦНТУ, 2019. - 27 с. – Режим доступу: http://dspace.kntu.kr.ua/jspui/handle/123456789/8769 (дата звернення 19.09.22).

62. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

63. Наказ Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 31.10.2016 «Про затвердження ДБН В.1.1-702016» - Режим доступу до ресурсу: ttps://ips.ligazakon.net/document/fn025551 (дата звернення 19.09.22).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Економічні вимоги.....5
8 Вимоги щодо охорони праці.....5
9 Перелік документів, що розробляються.....6
10 Етапи розробки.....6
11 Порядок контролю та приймання.....6

					ВКРМ-123.22.0027.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Шевченко О.О.			Дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Мелешко Є.В.				М	
Н. Контр.		Гермак В.С.				ЦНТУ КІ-21М1,4	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи штучного інтелекту для комп'ютерної гри на рушії Unity.

2 Підстава для розробки

Підставою для розробки служить завдання на магістерську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. №19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою магістерської роботи є дослідження та програмна реалізація системи штучного інтелекту для комп'ютерної гри на рушії Unity.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– техніко-економічне обґрунтування доцільності прийнятого до розробки

програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– роботу системи штучного інтелекту для шахової комп'ютерної гри на рушії Unity;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Мова програмування C#, середовище розробки Unity.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

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

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 року.

8 Вимоги щодо охорони праці

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

9 Перелік документів, що розробляються

– Наукова новизна.
– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Показники економічної ефективності.
– Пояснювальна записка.

10 Етапи розробки

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

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

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання магістерської роботи на попередній захист 10.12.2022 р.

11.2 Подання магістерської роботи на захист 23.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ
Керівник випускної кваліфікаційної роботи
за другим (магістерським) рівнем вищої освіти
_____ Є.В. Мелешко

Дослідження та програмна реалізація системи штучного інтелекту для комп’ютерної гри на рушії Unity

Лістинг програми

Код документу 12

Носій: DVD-диск

Загальна кількість аркушів: 42

Літера: РП

Кропивницький – 2022 року

Файл GameManager.cs

```
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoBehaviour
{
    public static GameManager instance;
    public GameObject Camera;

    public Board board;

    public GameObject whiteKing;
    public GameObject whiteQueen;
    public GameObject whiteBishop;
    public GameObject whiteKnight;
    public GameObject whiteRook;
    public GameObject whitePawn;

    public GameObject blackKing;
    public GameObject blackQueen;
    public GameObject blackBishop;
    public GameObject blackKnight;
    public GameObject blackRook;
    public GameObject blackPawn;

    public GameObject[,] pieces;
    private List<GameObject> movedPawns;

    private Player white;
    private Player black;
    public Player currentPlayer;
    public Player otherAIPlayer;

    public static int DEPTH = 4;

    public float score = 0.0f;


//////////////////////////////////////////////////////////////////////////////
<summary>
    public static float[,] reverseArray(float[,] arr)
    {
        float[,] arr1 = new float[8, 8];
        for (int i = 0; i < 8; i++)
        {
            int start = 0;
            int end = 7;

            while (start < end)
            {
                float temp = arr[start, i];
                arr1[start, i] = arr[end, i];
                arr1[end, i] = temp;
                start++;
                end--;
            }
        }
        return arr1;
    }

    public static float[,] rookEvalBlack = {
        { 0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f },
        { 0.5f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
```

```
    { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
    { 0.0f,   0.0f, 0.0f,  0.5f,  0.5f,  0.0f,  0.0f,  0.0f }
};
public static float[,] rookEvalWhite = reverseArray(rookEvalBlack);

public static float[,] queenEval =
{
    { -2.0f, -1.0f, -1.0f, -0.5f, -0.5f, -1.0f, -1.0f, -2.0f},
    { -1.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -1.0f},
    { -1.0f,  0.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -1.0f},
    { -0.5f,  0.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -0.5f},
    {  0.0f,  0.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -0.5f},
    { -1.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -1.0f},
    { -1.0f,  0.0f,  0.5f,  0.0f,  0.0f,  0.0f,  0.0f, -1.0f},
    { -2.0f, -1.0f, -1.0f, -0.5f, -0.5f, -1.0f, -1.0f, -2.0f}
};

public static float[,] kingEvalBlack =
{
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -2.0f, -3.0f, -3.0f, -4.0f, -4.0f, -3.0f, -3.0f, -2.0f},
    { -1.0f, -2.0f, -2.0f, -2.0f, -2.0f, -2.0f, -2.0f, -1.0f},
    {  2.0f,  2.0f,  0.0f,  0.0f,  0.0f,  0.0f,  2.0f,  2.0f },
    {  2.0f,  3.0f,  1.0f,  0.0f,  0.0f,  1.0f,  3.0f,  2.0f }
};
public static float[,] kingEvalWhite = reverseArray(kingEvalBlack);

public static float[,] bishopEvalBlack =
    {
        { -2.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -2.0f},
        { -1.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -1.0f},
        { -1.0f,  0.0f,  0.5f,  1.0f,  1.0f,  0.5f,  0.0f, -1.0f},
        { -1.0f,  0.5f,  0.5f,  1.0f,  1.0f,  0.5f,  0.5f, -1.0f},
        { -1.0f,  0.0f,  1.0f,  1.0f,  1.0f,  1.0f,  0.0f, -1.0f},
        { -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f, -1.0f},
        { -1.0f,  0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.5f, -1.0f},
        { -2.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -2.0f}
    };
public static float[,] bishopEvalWhite = reverseArray(bishopEvalBlack);

public static float[,] knightEval =
{
    { -5.0f, -4.0f, -3.0f, -3.0f, -3.0f, -3.0f, -4.0f, -5.0f},
    {-4.0f, -2.0f,  0.0f,  0.0f,  0.0f,  0.0f, -2.0f, -4.0f},
    {-3.0f,  0.0f,  1.0f,  1.5f,  1.5f,  1.0f,  0.0f, -3.0f},
    {-3.0f,  0.5f,  1.5f,  2.0f,  2.0f,  1.5f,  0.5f, -3.0f},
    {-3.0f,  0.0f,  1.5f,  2.0f,  2.0f,  1.5f,  0.0f, -3.0f},
    {-3.0f,  0.5f,  1.0f,  1.5f,  1.5f,  1.0f,  0.5f, -3.0f},
    {-4.0f, -2.0f,  0.0f,  0.5f,  0.5f,  0.0f, -2.0f, -4.0f},
    {-5.0f, -4.0f, -3.0f, -3.0f, -3.0f, -3.0f, -4.0f, -5.0f}
};

public static float[,] pawnEvalBlack = {
    { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f },
    { 5.0f, 5.0f, 5.0f, 5.0f, 5.0f, 5.0f, 5.0f, 5.0f },
    { 1.0f, 1.0f, 2.0f, 3.0f, 3.0f, 2.0f, 1.0f, 1.0f },
    { 0.5f, 0.5f, 1.0f, 2.5f, 2.5f, 1.0f, 0.5f, 0.5f },
    { 0.0f, 0.0f, 0.0f, 2.0f, 2.0f, 0.0f, 0.0f, 0.0f },
    { 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, -1.0f, -0.5f, 0.5f },
    { 0.5f, 1.0f, 1.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.5f },
    { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f }
};
public static float[,] pawnEvalWhite = reverseArray(pawnEvalBlack);
```

```
//////////////////////////////////////////////////////////////////////////////
</summary>


    void Awake()
    {
        instance = this;
    }

    void Start ()
    {
        pieces = new GameObject[8, 8];
        movedPawns = new List<GameObject>();

        white = new Player("white", true);
        black = new Player("black", false);

        currentPlayer = white;
        otherAIPlayer = black;

        InitialSetup();
    }

    private void InitialSetup()
    {
        AddPiece(whiteRook, white, 0, 0);
        AddPiece(whiteKnight, white, 1, 0);
        AddPiece(whiteBishop, white, 2, 0);
        AddPiece(whiteKing, white, 3, 0);
        AddPiece(whiteQueen, white, 4, 0);
        AddPiece(whiteBishop, white, 5, 0);
        AddPiece(whiteKnight, white, 6, 0);
        AddPiece(whiteRook, white, 7, 0);

        for (int i = 0; i < 8; i++)
        {
            AddPiece(whitePawn, white, i, 1);
        }

        AddPiece(blackRook, black, 0, 7);
        AddPiece(blackKnight, black, 1, 7);
        AddPiece(blackBishop, black, 2, 7);
        AddPiece(blackKing, black, 3, 7);
        AddPiece(blackQueen, black, 4, 7);
        AddPiece(blackBishop, black, 5, 7);
        AddPiece(blackKnight, black, 6, 7);
        AddPiece(blackRook, black, 7, 7);

        for (int i = 0; i < 8; i++)
        {
            AddPiece(blackPawn, black, i, 6);
        }
    }

    public void AddPiece(GameObject prefab, Player player, int col, int row)
    {
        GameObject pieceObject = board.AddPiece(prefab, col, row);
        if (player == white)
        {
            pieceObject.GetComponent<Piece>().isAI = false;
        }
        else
        {
            pieceObject.GetComponent<Piece>().isAI = true;
        }
        player.pieces.Add(pieceObject);
        pieces[col, row] = pieceObject;
```

```
        scoreUpdate(2, Vector2Int.zero, Vector2Int.zero, null, Vector2Int.zero,
pieceObject, null);
        //Debug.Log("Score: " + score + " " + pieceObject.name);
    }

    public void SelectPieceAtGrid(Vector2Int gridPoint)
    {
        GameObject selectedPiece = pieces[gridPoint.x, gridPoint.y];
        if (selectedPiece)
        {
            board.SelectPiece(selectedPiece);
        }
    }

    public List<Vector2Int> MovesForPiece(GameObject pieceObject)
    {
        Piece piece = pieceObject.GetComponent<Piece>();
        Vector2Int gridPoint = GridForPiece(pieceObject);
        List<Vector2Int> locations = piece.MoveLocations(gridPoint);

        // filter out offboard locations
        locations.RemoveAll(gp => gp.x < 0 || gp.x > 7 || gp.y < 0 || gp.y > 7);

        // filter out locations with friendly piece
        locations.RemoveAll(gp => FriendlyPieceAt(gp));

        return locations;
    }

    public void Move(GameObject piece, Vector2Int gridPoint)
    {
        Piece pieceComponent = piece.GetComponent<Piece>();
        if (pieceComponent.type == PieceType.Pawn && !HasPawnMoved(piece))
        {
            movedPawns.Add(piece);
        }
        Vector2Int startGridPoint = GridForPiece(piece);
        //Debug.Log("Old X: " + startGridPoint.x + " Old Y: " +
startGridPoint.y);
        pieces[startGridPoint.x, startGridPoint.y] = null;
        pieces[gridPoint.x, gridPoint.y] = piece;
        //Debug.Log("New X: " + gridPoint.x + " New Y: " + gridPoint.y);
        scoreUpdate(0, startGridPoint, gridPoint, piece, Vector2Int.zero, null,
null);
        //Debug.Log("Score: " + score + " " + piece.name);
        board.MovePiece(piece, gridPoint);
    }

    public void PawnMoved(GameObject pawn)
    {
        movedPawns.Add(pawn);
    }

    public bool HasPawnMoved(GameObject pawn)
    {
        return movedPawns.Contains(pawn);
    }

    public void CapturePieceAt(Vector2Int gridPoint)//Взяття фігури суперника
    {
        GameObject pieceToCapture = PieceAtGrid(gridPoint);
        if (pieceToCapture.GetComponent<Piece>().type == PieceType.King)
        {
            Debug.Log(currentPlayer.name + " wins!");
            Destroy(board.GetComponent<TileSelector>());
            Destroy(board.GetComponent<MoveSelector>());
        }
        currentPlayer.capturedPieces.Add(pieceToCapture);
```

```
        scoreUpdate(1, Vector2Int.zero, Vector2Int.zero, null,
GridForPiece(pieceToCapture), null, pieceToCapture.GetComponent<Piece>());
        pieces[gridPoint.x, gridPoint.y] = null;
        Destroy(pieceToCapture);
    }

    public void SelectPiece(GameObject piece)//Вибір фігури
    {
        board.SelectPiece(piece);
    }

    public void DeselectPiece(GameObject piece)//Відмна вибору фігури
    {
        board.DeselectPiece(piece);
    }

    public bool DoesPieceBelongToCurrentPlayer(GameObject piece)//Перевірка чи
належить фігури гравцю
    {
        return currentPlayer.pieces.Contains(piece);
    }

    public GameObject PieceAtGrid(Vector2Int gridPoint)
    {
        if (gridPoint.x > 7 || gridPoint.y > 7 || gridPoint.x < 0 || gridPoint.y
< 0)
        {
            return null;
        }
        return pieces[gridPoint.x, gridPoint.y];
    }

    public Vector2Int GridForPiece(GameObject piece)
    {
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
            {
                if (pieces[i, j] == piece)
                {
                    return new Vector2Int(i, j);
                }
            }
        }

        return new Vector2Int(-1, -1);
    }

    public bool FriendlyPieceAt(Vector2Int gridPoint)
    {
        GameObject piece = PieceAtGrid(gridPoint);

        if (piece == null) {
            return false;
        }

        if (otherAIPlayer.pieces.Contains(piece))
        {
            return false;
        }

        return true;
    }

    public void NextPlayer()
    {
        Player tempPlayer = currentPlayer;
        currentPlayer = otherAIPlayer;
        otherAIPlayer = tempPlayer;
```

```
        if (currentPlayer.name == "black")
        {
            //alphaBetaPruning();
        }
    }


    public void scoreUpdate(int eventType, Vector2Int movedPieceStartPosition,
Vector2Int movedPieceEndPosition, GameObject movedPiece, Vector2Int takenPiece,
GameObject instancePiece, Piece takenPieceData)
    {
        int reverse = 1;
        if (eventType == 0)//MovePiece
        {
            {
                Piece pieceComponent = movedPiece.GetComponent<Piece>();
                if (white.pieces.Contains(movedPiece))
                {
                    reverse = 1;
                }
                else if (black.pieces.Contains(movedPiece))
                {
                    reverse = -1;
                }
                if (pieceComponent.type == PieceType.Pawn)
                {
                    if (reverse == 1)
                    {
                        score -= reverse *
pawnEvalWhite[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
pawnEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
pawnEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
pawnEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
                else if (pieceComponent.type == PieceType.Knight)
                {
                    score -= reverse * knightEval[movedPieceStartPosition.y,
movedPieceStartPosition.x];
                    score += reverse * knightEval[movedPieceEndPosition.y,
movedPieceEndPosition.x];
                }
                else if (pieceComponent.type == PieceType.Bishop)
                {
                    if (reverse == 1)
                    {
                        score -= reverse *
bishopEvalWhite[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
bishopEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
bishopEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
bishopEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
                else if (pieceComponent.type == PieceType.Rook)
                {
                    if (reverse == 1)
                    {
```

```
                        score += reverse *
rookEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
rookEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
rookEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
                else if (pieceComponent.type == PieceType.Queen)
                {
                    score -= reverse * queenEval[movedPieceStartPosition.y,
movedPieceStartPosition.x];
                    score += reverse * queenEval[movedPieceEndPosition.y,
movedPieceEndPosition.x];
                }
                else if (pieceComponent.type == PieceType.King)
                {
                    if (reverse == 1)
                    {
                        score -= reverse *
kingEvalWhite[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
kingEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
kingEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
kingEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
            }
        }
        else if (eventType == 1)//DestroyPiece
        {
            if (takenPieceData.isAI == true)
            {
                reverse = -1;
            }
            else if (takenPieceData.isAI == false)
            {
                reverse = 1;
            }
            if (takenPieceData.type == PieceType.Pawn)
            {
                score += (-10 * reverse);

                if (reverse == -1)
                {
                    score += reverse * pawnEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * pawnEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
            else if (takenPieceData.type == PieceType.Knight)
            {
                score += -30 * reverse;
                score += reverse * knightEval[takenPiece.y, takenPiece.x];
            }
            else if (takenPieceData.type == PieceType.Bishop)
            {
```

```
                score += -30 * reverse;
                if (reverse == -1)
                {
                    score += reverse * bishopEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * bishopEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
            else if (takenPieceData.type == PieceType.Rook)
            {
                score += -50 * reverse;
                if (reverse == -1)
                {
                    score += reverse * rookEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * rookEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
            else if (takenPieceData.type == PieceType.Queen)
            {
                score += -90 * reverse;
                score += reverse * queenEval[takenPiece.y, takenPiece.x];
            }
            else if (takenPieceData.type == PieceType.King)
            {
                score += -900 * reverse;
                if (reverse == -1)
                {
                    score += reverse * kingEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * kingEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
        }
        else if (eventType == 2)//InitialeScoreCalculating
        {
            Piece pieceComponent = instancePiece.GetComponent<Piece>();
            if (white.pieces.Contains(instancePiece))
            {
                reverse = 1;
            }
            else if (black.pieces.Contains(instancePiece))
            {
                reverse = -1;
            }
            if (pieceComponent.type == PieceType.Pawn)
            {
                score += (10 * reverse);
                if (reverse == 1)
                {
                    score += reverse *
pawnEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
pawnEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
```

```
                }
            }
            else if (pieceComponent.type == PieceType.Knight)
            {
                score += 30 * reverse;
                score += reverse * knightEval[GridForPiece(instancePiece).y,
GridForPiece(instancePiece).x];
            }
            else if (pieceComponent.type == PieceType.Bishop)
            {
                score += 30 * reverse;
                if (reverse == 1)
                {
                    score += reverse *
bishopEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
bishopEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
            }
            else if (pieceComponent.type == PieceType.Rook)
            {
                score += 50 * reverse;
                if (reverse == 1)
                {
                    score += reverse *
rookEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
rookEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
            }
            else if (pieceComponent.type == PieceType.Queen)
            {
                score += 90 * reverse;
                score += reverse * queenEval[GridForPiece(instancePiece).y,
GridForPiece(instancePiece).x];
            }
            else if (pieceComponent.type == PieceType.King)
            {
                score += 900 * reverse;
                if (reverse == 1)
                {
                    score += reverse *
kingEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
kingEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
            }
        }
        Debug.Log("Score: " + score);
    }
    public void alphaBetaPruning(int depth, int nodeIndex,
                bool maximizingPlayer,
                int[] values, int alpha,
                int beta)
    {
        // Terminating condition. i.e
        // leaf node is reached
        if (depth == 3)
            return values[nodeIndex];
```

```
    if (maximizingPlayer)
    {
        int best = MIN;
        for (int i = 0; i < 2; i++)
        {
            int val = minimax(depth + 1, nodeIndex * 2 + i,
                              false, values, alpha, beta);
            best = Math.Max(best, val);
            alpha = Math.Max(alpha, best);

            // Alpha Beta Pruning
            if (beta <= alpha)
                break;
        }
        return best;
    }
    else
    {
        int best = MAX;
        for (int i = 0; i < 2; i++)
        {

            int val = minimax(depth + 1, nodeIndex * 2 + i,
                              true, values, alpha, beta);
            best = Math.Min(best, val);
            beta = Math.Min(beta, best);

            // Alpha Beta Pruning
            if (beta <= alpha)
                break;
        }
        return best;
    }
    function alphabeta(node, depth, α, β, maximizingPlayer) is
if depth = 0 or node is a terminal node then
    return the heuristic value of node
if maximizingPlayer then
    value := −∞
    for each child of node do
        value:= max(value, alphabeta(child, depth − 1, α, β, FALSE))
        α := max(α, value)
        if value ≥ β then
            break (*β cutoff *)
    return value
else
    value:= +∞
    for each child of node do
        value:= min(value, alphabeta(child, depth − 1, α, β, TRUE))
        β := min(β, value)
        if value ≤ α then
            break (*α cutoff *)
    return value
}
}
```

Файл UnitManager.cs

```
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoBehaviour
{
    public static GameManager instance;
    public GameObject Camera;

    public Board board;

    public GameObject whiteKing;
    public GameObject whiteQueen;
    public GameObject whiteBishop;
    public GameObject whiteKnight;
    public GameObject whiteRook;
    public GameObject whitePawn;

    public GameObject blackKing;
    public GameObject blackQueen;
    public GameObject blackBishop;
    public GameObject blackKnight;
    public GameObject blackRook;
    public GameObject blackPawn;

    public GameObject[,] pieces;
    private List<GameObject> movedPawns;

    private Player white;
    private Player black;
    public Player currentPlayer;
    public Player otherAIPlayer;

    public static int DEPTH = 4;

    public float score = 0.0f;


//////////////////////////////////////////////////////////////////////////////
<summary>
    public static float[,] reverseArray(float[,] arr)
    {
        float[,] arr1 = new float[8, 8];
        for (int i = 0; i < 8; i++)
        {
            int start = 0;
            int end = 7;

            while (start < end)
            {
                float temp = arr[start, i];
                arr1[start, i] = arr[end, i];
                arr1[end, i] = temp;
                start++;
                end--;
            }
        }
        return arr1;
    }

    public static float[,] rookEvalBlack = {
        { 0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f },
        { 0.5f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
        { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
```

```
    { -0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -0.5f },
    { 0.0f,   0.0f, 0.0f,  0.5f,  0.5f,  0.0f,  0.0f,  0.0f }
};
public static float[,] rookEvalWhite = reverseArray(rookEvalBlack);

public static float[,] queenEval =
{
    { -2.0f, -1.0f, -1.0f, -0.5f, -0.5f, -1.0f, -1.0f, -2.0f},
    { -1.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -1.0f},
    { -1.0f,  0.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -1.0f},
    { -0.5f,  0.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -0.5f},
    {  0.0f,  0.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -0.5f},
    { -1.0f,  0.5f,  0.5f,  0.5f,  0.5f,  0.5f,  0.0f, -1.0f},
    { -1.0f,  0.0f,  0.5f,  0.0f,  0.0f,  0.0f,  0.0f, -1.0f},
    { -2.0f, -1.0f, -1.0f, -0.5f, -0.5f, -1.0f, -1.0f, -2.0f}
};

public static float[,] kingEvalBlack =
{
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -3.0f, -4.0f, -4.0f, -5.0f, -5.0f, -4.0f, -4.0f, -3.0f},
    { -2.0f, -3.0f, -3.0f, -4.0f, -4.0f, -3.0f, -3.0f, -2.0f},
    { -1.0f, -2.0f, -2.0f, -2.0f, -2.0f, -2.0f, -2.0f, -1.0f},
    {  2.0f,  2.0f,  0.0f,  0.0f,  0.0f,  0.0f,  2.0f,  2.0f },
    {  2.0f,  3.0f,  1.0f,  0.0f,  0.0f,  1.0f,  3.0f,  2.0f }
};
public static float[,] kingEvalWhite = reverseArray(kingEvalBlack);

public static float[,] bishopEvalBlack =
    {
        { -2.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -2.0f},
        { -1.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f,  0.0f, -1.0f},
        { -1.0f,  0.0f,  0.5f,  1.0f,  1.0f,  0.5f,  0.0f, -1.0f},
        { -1.0f,  0.5f,  0.5f,  1.0f,  1.0f,  0.5f,  0.5f, -1.0f},
        { -1.0f,  0.0f,  1.0f,  1.0f,  1.0f,  1.0f,  0.0f, -1.0f},
        { -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f, -1.0f},
        { -1.0f,  0.5f,  0.0f,  0.0f,  0.0f,  0.0f,  0.5f, -1.0f},
        { -2.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -2.0f}
    };
public static float[,] bishopEvalWhite = reverseArray(bishopEvalBlack);

public static float[,] knightEval =
{
    { -5.0f, -4.0f, -3.0f, -3.0f, -3.0f, -3.0f, -4.0f, -5.0f},
    {-4.0f, -2.0f,  0.0f,  0.0f,  0.0f,  0.0f, -2.0f, -4.0f},
    {-3.0f,  0.0f,  1.0f,  1.5f,  1.5f,  1.0f,  0.0f, -3.0f},
    {-3.0f,  0.5f,  1.5f,  2.0f,  2.0f,  1.5f,  0.5f, -3.0f},
    {-3.0f,  0.0f,  1.5f,  2.0f,  2.0f,  1.5f,  0.0f, -3.0f},
    {-3.0f,  0.5f,  1.0f,  1.5f,  1.5f,  1.0f,  0.5f, -3.0f},
    {-4.0f, -2.0f,  0.0f,  0.5f,  0.5f,  0.0f, -2.0f, -4.0f},
    {-5.0f, -4.0f, -3.0f, -3.0f, -3.0f, -3.0f, -4.0f, -5.0f}
};

public static float[,] pawnEvalBlack = {
    { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f },
    { 5.0f, 5.0f, 5.0f, 5.0f, 5.0f, 5.0f, 5.0f, 5.0f },
    { 1.0f, 1.0f, 2.0f, 3.0f, 3.0f, 2.0f, 1.0f, 1.0f },
    { 0.5f, 0.5f, 1.0f, 2.5f, 2.5f, 1.0f, 0.5f, 0.5f },
    { 0.0f, 0.0f, 0.0f, 2.0f, 2.0f, 0.0f, 0.0f, 0.0f },
    { 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, -1.0f, -0.5f, 0.5f },
    { 0.5f, 1.0f, 1.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.5f },
    { 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f }
};
public static float[,] pawnEvalWhite = reverseArray(pawnEvalBlack);
```

```
//////////////////////////////////////////////////////////////////////////////
</summary>


    void Awake()
    {
        instance = this;
    }

    void Start ()
    {
        pieces = new GameObject[8, 8];
        movedPawns = new List<GameObject>();

        white = new Player("white", true);
        black = new Player("black", false);

        currentPlayer = white;
        otherAIPlayer = black;

        InitialSetup();
    }

    private void InitialSetup()
    {
        AddPiece(whiteRook, white, 0, 0);
        AddPiece(whiteKnight, white, 1, 0);
        AddPiece(whiteBishop, white, 2, 0);
        AddPiece(whiteKing, white, 3, 0);
        AddPiece(whiteQueen, white, 4, 0);
        AddPiece(whiteBishop, white, 5, 0);
        AddPiece(whiteKnight, white, 6, 0);
        AddPiece(whiteRook, white, 7, 0);

        for (int i = 0; i < 8; i++)
        {
            AddPiece(whitePawn, white, i, 1);
        }

        AddPiece(blackRook, black, 0, 7);
        AddPiece(blackKnight, black, 1, 7);
        AddPiece(blackBishop, black, 2, 7);
        AddPiece(blackKing, black, 3, 7);
        AddPiece(blackQueen, black, 4, 7);
        AddPiece(blackBishop, black, 5, 7);
        AddPiece(blackKnight, black, 6, 7);
        AddPiece(blackRook, black, 7, 7);

        for (int i = 0; i < 8; i++)
        {
            AddPiece(blackPawn, black, i, 6);
        }
    }

    public void AddPiece(GameObject prefab, Player player, int col, int row)
    {
        GameObject pieceObject = board.AddPiece(prefab, col, row);
        if (player == white)
        {
            pieceObject.GetComponent<Piece>().isAI = false;
        }
        else
        {
            pieceObject.GetComponent<Piece>().isAI = true;
        }
        player.pieces.Add(pieceObject);
        pieces[col, row] = pieceObject;
```

```
        scoreUpdate(2, Vector2Int.zero, Vector2Int.zero, null, Vector2Int.zero,
pieceObject, null);
        //Debug.Log("Score: " + score + " " + pieceObject.name);
    }

    public void SelectPieceAtGrid(Vector2Int gridPoint)
    {
        GameObject selectedPiece = pieces[gridPoint.x, gridPoint.y];
        if (selectedPiece)
        {
            board.SelectPiece(selectedPiece);
        }
    }

    public List<Vector2Int> MovesForPiece(GameObject pieceObject)
    {
        Piece piece = pieceObject.GetComponent<Piece>();
        Vector2Int gridPoint = GridForPiece(pieceObject);
        List<Vector2Int> locations = piece.MoveLocations(gridPoint);

        // filter out offboard locations
        locations.RemoveAll(gp => gp.x < 0 || gp.x > 7 || gp.y < 0 || gp.y > 7);

        // filter out locations with friendly piece
        locations.RemoveAll(gp => FriendlyPieceAt(gp));

        return locations;
    }

    public void Move(GameObject piece, Vector2Int gridPoint)
    {
        Piece pieceComponent = piece.GetComponent<Piece>();
        if (pieceComponent.type == PieceType.Pawn && !HasPawnMoved(piece))
        {
            movedPawns.Add(piece);
        }
        Vector2Int startGridPoint = GridForPiece(piece);
        //Debug.Log("Old X: " + startGridPoint.x + " Old Y: " +
startGridPoint.y);
        pieces[startGridPoint.x, startGridPoint.y] = null;
        pieces[gridPoint.x, gridPoint.y] = piece;
        //Debug.Log("New X: " + gridPoint.x + " New Y: " + gridPoint.y);
        scoreUpdate(0, startGridPoint, gridPoint, piece, Vector2Int.zero, null,
null);
        //Debug.Log("Score: " + score + " " + piece.name);
        board.MovePiece(piece, gridPoint);
    }

    public void PawnMoved(GameObject pawn)
    {
        movedPawns.Add(pawn);
    }

    public bool HasPawnMoved(GameObject pawn)
    {
        return movedPawns.Contains(pawn);
    }

    public void CapturePieceAt(Vector2Int gridPoint)//Взяття фігури суперника
    {
        GameObject pieceToCapture = PieceAtGrid(gridPoint);
        if (pieceToCapture.GetComponent<Piece>().type == PieceType.King)
        {
            Debug.Log(currentPlayer.name + " wins!");
            Destroy(board.GetComponent<TileSelector>());
            Destroy(board.GetComponent<MoveSelector>());
        }
        currentPlayer.capturedPieces.Add(pieceToCapture);
```

```
        scoreUpdate(1, Vector2Int.zero, Vector2Int.zero, null,
GridForPiece(pieceToCapture), null, pieceToCapture.GetComponent<Piece>());
        pieces[gridPoint.x, gridPoint.y] = null;
        Destroy(pieceToCapture);
    }

    public void SelectPiece(GameObject piece)//Вибір фігури
    {
        board.SelectPiece(piece);
    }

    public void DeselectPiece(GameObject piece)//Відмна вибору фігури
    {
        board.DeselectPiece(piece);
    }

    public bool DoesPieceBelongToCurrentPlayer(GameObject piece)//Перевірка чи
належить фігури гравцю
    {
        return currentPlayer.pieces.Contains(piece);
    }

    public GameObject PieceAtGrid(Vector2Int gridPoint)
    {
        if (gridPoint.x > 7 || gridPoint.y > 7 || gridPoint.x < 0 || gridPoint.y
< 0)
        {
            return null;
        }
        return pieces[gridPoint.x, gridPoint.y];
    }

    public Vector2Int GridForPiece(GameObject piece)
    {
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
            {
                if (pieces[i, j] == piece)
                {
                    return new Vector2Int(i, j);
                }
            }
        }

        return new Vector2Int(-1, -1);
    }

    public bool FriendlyPieceAt(Vector2Int gridPoint)
    {
        GameObject piece = PieceAtGrid(gridPoint);

        if (piece == null) {
            return false;
        }

        if (otherAIPlayer.pieces.Contains(piece))
        {
            return false;
        }

        return true;
    }

    public void NextPlayer()
    {
        Player tempPlayer = currentPlayer;
        currentPlayer = otherAIPlayer;
        otherAIPlayer = tempPlayer;
```

```
        if (currentPlayer.name == "black")
        {
            alphaBetaPruning();
        }
    }


    public void scoreUpdate(int eventType, Vector2Int movedPieceStartPosition,
Vector2Int movedPieceEndPosition, GameObject movedPiece, Vector2Int takenPiece,
GameObject instancePiece, Piece takenPieceData)
    {
        int reverse = 1;
        if (eventType == 0)//MovePiece
        {
            {
                Piece pieceComponent = movedPiece.GetComponent<Piece>();
                if (white.pieces.Contains(movedPiece))
                {
                    reverse = 1;
                }
                else if (black.pieces.Contains(movedPiece))
                {
                    reverse = -1;
                }
                if (pieceComponent.type == PieceType.Pawn)
                {
                    if (reverse == 1)
                    {
                        score -= reverse *
pawnEvalWhite[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
pawnEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
pawnEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
pawnEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
                else if (pieceComponent.type == PieceType.Knight)
                {
                    score -= reverse * knightEval[movedPieceStartPosition.y,
movedPieceStartPosition.x];
                    score += reverse * knightEval[movedPieceEndPosition.y,
movedPieceEndPosition.x];
                }
                else if (pieceComponent.type == PieceType.Bishop)
                {
                    if (reverse == 1)
                    {
                        score -= reverse *
bishopEvalWhite[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
bishopEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
bishopEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
bishopEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
                else if (pieceComponent.type == PieceType.Rook)
                {
                    if (reverse == 1)
                    {
```

```
                        score += reverse *
rookEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
rookEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
rookEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
                else if (pieceComponent.type == PieceType.Queen)
                {
                    score -= reverse * queenEval[movedPieceStartPosition.y,
movedPieceStartPosition.x];
                    score += reverse * queenEval[movedPieceEndPosition.y,
movedPieceEndPosition.x];
                }
                else if (pieceComponent.type == PieceType.King)
                {
                    if (reverse == 1)
                    {
                        score -= reverse *
kingEvalWhite[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
kingEvalWhite[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                    else if (reverse == -1)
                    {
                        score -= reverse *
kingEvalBlack[movedPieceStartPosition.y, movedPieceStartPosition.x];
                        score += reverse *
kingEvalBlack[movedPieceEndPosition.y, movedPieceEndPosition.x];
                    }
                }
            }
        }
        else if (eventType == 1)//DestroyPiece
        {
            if (takenPieceData.isAI == true)
            {
                reverse = -1;
            }
            else if (takenPieceData.isAI == false)
            {
                reverse = 1;
            }
            if (takenPieceData.type == PieceType.Pawn)
            {
                score += (-10 * reverse);

                if (reverse == -1)
                {
                    score += reverse * pawnEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * pawnEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
            else if (takenPieceData.type == PieceType.Knight)
            {
                score += -30 * reverse;
                score += reverse * knightEval[takenPiece.y, takenPiece.x];
            }
            else if (takenPieceData.type == PieceType.Bishop)
            {
```

```
                score += -30 * reverse;
                if (reverse == -1)
                {
                    score += reverse * bishopEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * bishopEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
            else if (takenPieceData.type == PieceType.Rook)
            {
                score += -50 * reverse;
                if (reverse == -1)
                {
                    score += reverse * rookEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * rookEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
            else if (takenPieceData.type == PieceType.Queen)
            {
                score += -90 * reverse;
                score += reverse * queenEval[takenPiece.y, takenPiece.x];
            }
            else if (takenPieceData.type == PieceType.King)
            {
                score += -900 * reverse;
                if (reverse == -1)
                {
                    score += reverse * kingEvalWhite[takenPiece.y,
takenPiece.x];
                }
                else if (reverse == 1)
                {
                    score += reverse * kingEvalBlack[takenPiece.y,
takenPiece.x];
                }
            }
        }
        else if (eventType == 2)//InitialeScoreCalculating
        {
            Piece pieceComponent = instancePiece.GetComponent<Piece>();
            if (white.pieces.Contains(instancePiece))
            {
                reverse = 1;
            }
            else if (black.pieces.Contains(instancePiece))
            {
                reverse = -1;
            }
            if (pieceComponent.type == PieceType.Pawn)
            {
                score += (10 * reverse);
                if (reverse == 1)
                {
                    score += reverse *
pawnEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
pawnEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
```

```
                }
            }
            else if (pieceComponent.type == PieceType.Knight)
            {
                score += 30 * reverse;
                score += reverse * knightEval[GridForPiece(instancePiece).y,
GridForPiece(instancePiece).x];
            }
            else if (pieceComponent.type == PieceType.Bishop)
            {
                score += 30 * reverse;
                if (reverse == 1)
                {
                    score += reverse *
bishopEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
bishopEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
            }
            else if (pieceComponent.type == PieceType.Rook)
            {
                score += 50 * reverse;
                if (reverse == 1)
                {
                    score += reverse *
rookEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
rookEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
            }
            else if (pieceComponent.type == PieceType.Queen)
            {
                score += 90 * reverse;
                score += reverse * queenEval[GridForPiece(instancePiece).y,
GridForPiece(instancePiece).x];
            }
            else if (pieceComponent.type == PieceType.King)
            {
                score += 900 * reverse;
                if (reverse == 1)
                {
                    score += reverse *
kingEvalWhite[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
                else if (reverse == -1)
                {
                    score += reverse *
kingEvalBlack[GridForPiece(instancePiece).y, GridForPiece(instancePiece).x];
                }
            }
        }
        Debug.Log("Score: " + score);
    }
    public void alphaBetaPruning(int depth, int nodeIndex,
                bool maximizingPlayer,
                int[] values, int alpha,
                int beta)
    {
        // Terminating condition. i.e
        // leaf node is reached
        if (depth == 3)
            return values[nodeIndex];
```

```
if (maximizingPlayer)
{
    int best = MIN;
    for (int i = 0; i < 2; i++)
    {
        int val = minimax(depth + 1, nodeIndex * 2 + i,
                        false, values, alpha, beta);
        best = Math.Max(best, val);
        alpha = Math.Max(alpha, best);

        // Alpha Beta Pruning
        if (beta <= alpha)
            break;
    }
    return best;
}
else
{
    int best = MAX;
    for (int i = 0; i < 2; i++)
    {

        int val = minimax(depth + 1, nodeIndex * 2 + i,
                        true, values, alpha, beta);
        best = Math.Min(best, val);
        beta = Math.Min(beta, best);

        // Alpha Beta Pruning
        if (beta <= alpha)
            break;
    }
    return best;
}
```

Файл Board.cs

```
using UnityEngine;

public class Board : MonoBehaviour
{
    public Material defaultMaterial;
    public Material selectedMaterial;

    public GameObject AddPiece(GameObject piece, int col, int row)
    {
        Vector2Int gridPoint = Geometry.GridPoint(col, row);
        GameObject newPiece = Instantiate(piece,
Geometry.PointFromGrid(gridPoint), Quaternion.identity, gameObject.transform);
        return newPiece;
    }

    public void RemovePiece(GameObject piece)
    {
        Destroy(piece);
    }

    public void MovePiece(GameObject piece, Vector2Int gridPoint)
    {

        piece.transform.position = Geometry.PointFromGrid(gridPoint);
    }

    public void SelectPiece(GameObject piece)
    {
        MeshRenderer renderers = piece.GetComponentInChildren<MeshRenderer>();
        renderers.material = selectedMaterial;
    }

    public void DeselectPiece(GameObject piece)
    {
        MeshRenderer renderers = piece.GetComponentInChildren<MeshRenderer>();
        renderers.material = defaultMaterial;
    }
}
```

Файл Player.cs

```
using System.Collections.Generic;
using UnityEngine;

public class Player
{
    public List<GameObject> pieces;
    public List<GameObject> capturedPieces;

    public string name;
    public int forward;

    public Player(string name, bool positiveZMovement)
    {
        this.name = name;
        pieces = new List<GameObject>();
        capturedPieces = new List<GameObject>();

        if (positiveZMovement == true)
        {
            this.forward = 1;
        }
        else
        {
            this.forward = -1;
        }
    }
}
```

Файл Piece.cs

```
using System.Collections.Generic;
using UnityEngine;

public enum PieceType {King, Queen, Bishop, Knight, Rook, Pawn};
public enum Coordinates { A, B, C, D, E, F, G, H };

public abstract class Piece : MonoBehaviour
{
    public PieceType type;
    public bool isAI;

    protected Vector2Int[] RookDirections = {new Vector2Int(0,1), new
Vector2Int(1, 0),
        new Vector2Int(0, -1), new Vector2Int(-1, 0)};
    protected Vector2Int[] BishopDirections = {new Vector2Int(1,1), new
Vector2Int(1, -1),
        new Vector2Int(-1, -1), new Vector2Int(-1, 1)};

    public abstract List<Vector2Int> MoveLocations(Vector2Int gridPoint);


}
```

Файл Bishop.cs

```
using System.Collections.Generic;
using UnityEngine;

public class Bishop : Piece
{

    public override List<Vector2Int> MoveLocations(Vector2Int gridPoint)
    {
        List<Vector2Int> locations = new List<Vector2Int>();

        foreach (Vector2Int dir in BishopDirections)
        {
            for (int i = 1; i < 8; i++)
            {
                Vector2Int nextGridPoint = new Vector2Int(gridPoint.x + i *
dir.x, gridPoint.y + i * dir.y);
                locations.Add(nextGridPoint);
                if (GameManager.instance.PieceAtGrid(nextGridPoint))
                {
                    break;
                }
            }
        }

        return locations;
    }
}
```

Файл Pawn.cs

```
using System.Collections.Generic;
using UnityEngine;

public class Pawn : Piece
{
    public override List<Vector2Int> MoveLocations(Vector2Int gridPoint)
    {
        List<Vector2Int> locations = new List<Vector2Int>();

        int forwardDirection = GameManager.instance.currentPlayer.forward;
        Vector2Int forwardOne = new Vector2Int(gridPoint.x, gridPoint.y +
forwardDirection);
        if (GameManager.instance.PieceAtGrid(forwardOne) == false)
        {
            locations.Add(forwardOne);
        }

        Vector2Int forwardTwo = new Vector2Int(gridPoint.x, gridPoint.y + 2 *
forwardDirection);
        if (GameManager.instance.HasPawnMoved(gameObject) == false &&
GameManager.instance.PieceAtGrid(forwardTwo) == false)
        {
            locations.Add(forwardTwo);
        }

        Vector2Int forwardRight = new Vector2Int(gridPoint.x + 1, gridPoint.y +
forwardDirection);
        if (GameManager.instance.PieceAtGrid(forwardRight))
        {
            locations.Add(forwardRight);
        }

        Vector2Int forwardLeft = new Vector2Int(gridPoint.x - 1, gridPoint.y +
forwardDirection);
        if (GameManager.instance.PieceAtGrid(forwardLeft))
        {
            locations.Add(forwardLeft);
        }

        return locations;
    }
}
```

Файл Knight.cs

```
using System.Collections.Generic;
using UnityEngine;

public class Knight : Piece
{
    public override List<Vector2Int> MoveLocations(Vector2Int gridPoint)
    {
        List<Vector2Int> locations = new List<Vector2Int>();

        locations.Add(new Vector2Int(gridPoint.x - 1, gridPoint.y + 2));
        locations.Add(new Vector2Int(gridPoint.x + 1, gridPoint.y + 2));

        locations.Add(new Vector2Int(gridPoint.x + 2, gridPoint.y + 1));
        locations.Add(new Vector2Int(gridPoint.x - 2, gridPoint.y + 1));

        locations.Add(new Vector2Int(gridPoint.x + 2, gridPoint.y - 1));
        locations.Add(new Vector2Int(gridPoint.x - 2, gridPoint.y - 1));

        locations.Add(new Vector2Int(gridPoint.x + 1, gridPoint.y - 2));
        locations.Add(new Vector2Int(gridPoint.x - 1, gridPoint.y - 2));

        return locations;
    }
}
```

Файл King.cs

```
using System.Collections.Generic;
using UnityEngine;

public class King : Piece
{
    public override List<Vector2Int> MoveLocations(Vector2Int gridPoint)
    {
        List<Vector2Int> locations = new List<Vector2Int>();
        List<Vector2Int> directions = new List<Vector2Int>(BishopDirections);
        directions.AddRange(RookDirections);

        foreach (Vector2Int dir in directions)
        {
            Vector2Int nextGridPoint = new Vector2Int(gridPoint.x + dir.x,
gridPoint.y + dir.y);
            locations.Add(nextGridPoint);
        }

        return locations;
    }
}
```

Файл Queen.cs

```
using System.Collections.Generic;
using UnityEngine;

public class Queen : Piece
{
    public override List<Vector2Int> MoveLocations(Vector2Int gridPoint)
    {
        List<Vector2Int> locations = new List<Vector2Int>();
        List<Vector2Int> directions = new
List<Vector2Int>(BishopDirections);
        directions.AddRange(RookDirections);

        foreach (Vector2Int dir in directions)
        {
            for (int i = 1; i < 8; i++)
            {
                Vector2Int nextGridPoint = new Vector2Int(gridPoint.x + i
* dir.x, gridPoint.y + i * dir.y);
                locations.Add(nextGridPoint);
                if (GameManager.instance.PieceAtGrid(nextGridPoint))
                {
                    break;
                }
            }
        }

        return locations;
    }
}
```

Файл Rook.cs

```
using System.Collections.Generic;
using UnityEngine;

public class Rook : Piece
{
    public override List<Vector2Int> MoveLocations(Vector2Int gridPoint)
    {
        List<Vector2Int> locations = new List<Vector2Int>();

        foreach (Vector2Int dir in RookDirections)
        {
            for (int i = 1; i < 8; i++)
            {
                Vector2Int nextGridPoint = new Vector2Int(gridPoint.x + i *
dir.x, gridPoint.y + i * dir.y);
                locations.Add(nextGridPoint);
                if (GameManager.instance.PieceAtGrid(nextGridPoint))
                {
                    break;
                }
            }
        }

        return locations;
    }
}
```

Файл Minimax.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

namespace ChessAIForms
{
    public class AI
    {

        private const int pawnValue = 100;
        private const int knightValue = 320;
        private const int bishopValue = 330;
        private const int rookValue = 500;
        private const int queenValue = 900;
        private const int kingValue = 20000;
        private static readonly int[] bestPawnPositions = {
              0,  0,  0,  0,  0,  0,  0,  0,
             50, 50, 50, 50, 50, 50, 50, 50,
             10, 10, 20, 30, 30, 20, 10, 10,
              5,  5, 10, 25, 25, 10,  5,  5,
              0,  0,  0, 20, 20,  0,  0,  0,
              5, -5,-10,  0,  0,-10, -5,  5,
              5, 10, 10,-20,-20, 10, 10,  5,
              0,  0,  0,  0,  0,  0,  0,  0
        };

        private static readonly int[] bestKnightPositions = {
            -50,-40,-30,-30,-30,-30,-40,-50,
            -40,-20,  0,  0,  0,  0,-20,-40,
            -30,  0, 10, 15, 15, 10,  0,-30,
            -30,  5, 15, 20, 20, 15,  5,-30,
            -30,  0, 15, 20, 20, 15,  0,-30,
            -30,  5, 10, 15, 15, 10,  5,-30,
            -40,-20,  0,  5,  5,  0,-20,-40,
            -50,-40,-30,-30,-30,-30,-40,-50,
        };

        private static readonly int[] bestBishopPositions = {
            -20,-10,-10,-10,-10,-10,-10,-20,
            -10,  0,  0,  0,  0,  0,  0,-10,
            -10,  0,  5, 10, 10,  5,  0,-10,
            -10,  5,  5, 10, 10,  5,  5,-10,
            -10,  0, 10, 10, 10, 10,  0,-10,
            -10, 10, 10, 10, 10, 10, 10,-10,
            -10,  5,  0,  0,  0,  0,  5,-10,
            -20,-10,-10,-10,-10,-10,-10,-20,
        };

        private static readonly int[] bestRookPositions = {
              0,  0,  0,  0,  0,  0,  0,  0,
              5, 10, 10, 10, 10, 10, 10,  5,
             -5,  0,  0,  0,  0,  0,  0, -5,
             -5,  0,  0,  0,  0,  0,  0, -5,
             -5,  0,  0,  0,  0,  0,  0, -5,
             -5,  0,  0,  0,  0,  0,  0, -5,
             -5,  0,  0,  0,  0,  0,  0, -5,
              0,  0,  0,  5,  5,  0,  0,  0
        };

        private static readonly int[] bestQueenPositions = {
             -20,-10,-10, -5, -5,-10,-10,-20,
             -10,  0,  0,  0,  0,  0,  0,-10,
             -10,  0,  5,  5,  5,  5,  0,-10,
              -5,  0,  5,  5,  5,  5,  0, -5,
```

```
              0,  0,  5,  5,  5,  5,  0, -5,
            -10,  5,  5,  5,  5,  5,  0,-10,
            -10,  0,  5,  0,  0,  0,  0,-10,
            -20,-10,-10, -5, -5,-10,-10,-20
        };

        private static readonly int[] bestKingPositions = {
            -30,-40,-40,-50,-50,-40,-40,-30,
            -30,-40,-40,-50,-50,-40,-40,-30,
            -30,-40,-40,-50,-50,-40,-40,-30,
            -30,-40,-40,-50,-50,-40,-40,-30,
            -20,-30,-30,-40,-40,-30,-30,-20,
            -10,-20,-20,-20,-20,-20,-20,-10,
             20, 20,  0,  0,  0,  0, 20, 20,
             20, 30, 10,  0,  0, 10, 30, 20
        };

        public AI(int _depth)
        {
            depth = _depth;
        }

        public int CalculatePoint(Board board)
        {
            int scoreWhite = 0;
            int scoreBlack = 0;
            scoreWhite += GetScoreFromExistingPieces(Player.White, board);
            scoreBlack += GetScoreFromExistingPieces(Player.Black, board);

            int evaluation = scoreBlack - scoreWhite;

            int prespective = (board.Turn == Player.White) ? -1 : 1;
            return evaluation * prespective;
        }

        private static int GetScoreFromExistingPieces(Player player, Board
board)
        {
            int material = 0;

            for (int i = 0; i < 64; i++)
            {
                if (board.Pieces[i] != null)
                {
                    if (board.Pieces[i].GetType() == typeof(Pawn) &&
board.Pieces[i].Player == player)
                    {
                        material += (pawnValue + bestPawnPositions[i]); // plus
"+ bestPawnPositions[i]" if you want, but it doesn't work well
                    }
                    if (board.Pieces[i].GetType() == typeof(Knight) &&
board.Pieces[i].Player == player)
                    {
                        material += (knightValue); // plus "+
bestKnightPositions[i]" if you want, but it doesn't work well
                    }
                    if (board.Pieces[i].GetType() == typeof(Bishop) &&
board.Pieces[i].Player == player)
                    {
                        material += (bishopValue); // plus "+
bestBishopPositions[i]" if you want, but it doesn't work well
                    }
                    if (board.Pieces[i].GetType() == typeof(Rook) &&
board.Pieces[i].Player == player)
                    {
                        material += (rookValue); // plus "+
bestRookPositions[i]" if you want, but it doesn't work well
                    }
```

```
                    if (board.Pieces[i].GetType() == typeof(Queen) &&
board.Pieces[i].Player == player)
                    {
                        material += (queenValue); // plus "+
bestQueenPositions[i]" if you want, but it doesn't work well
                    }
                    if (board.Pieces[i].GetType() == typeof(King) &&
board.Pieces[i].Player == player)
                    {
                        material += (kingValue); // plus "+
bestKingPositions[i]" if you want, but it doesn't work well
                    }
                }
            }
            return material;
        }



        //minimax algorithm

        private Board GenerateMovedBoard(Board oldBoard, Move move)
        {
            Board newBoard = new Board();
            newBoard = ObjectExtensions.Copy(oldBoard);
            Board.MovePiece(newBoard, move.Tile, move.Next);
            return newBoard;
        }

        private int GetPieceValue(Board board, int index)
        {
            if (board.Pieces[index].GetType() == typeof(Pawn))
            {
                return pawnValue;
            }
            else if (board.Pieces[index].GetType() == typeof(Rook))
            {
                return rookValue;
            }
            else if (board.Pieces[index].GetType() == typeof(Knight))
            {
                return knightValue;
            }
            else if (board.Pieces[index].GetType() == typeof(Bishop))
            {
                return bishopValue;
            }
            else if (board.Pieces[index].GetType() == typeof(Queen))
            {
                return queenValue;
            }
            else if (board.Pieces[index].GetType() == typeof(King))
            {
                return kingValue;
            }

            return 0;
        }

        private void OrderMoves(List<Move> moveList, Board board)
        {
            int[] moveScore = new int[moveList.Count];

            for (int i = 0; i < moveList.Count; i++)
            {
                moveScore[i] = 0;

                if (board.Pieces[moveList[i].Next] != null )
```

```
                {
                    moveScore[i] += 10 * GetPieceValue(board, moveList[i].Next)
- GetPieceValue(board, moveList[i].Tile);
                }

                if (Board.PawnPromoted(board.Pieces, moveList[i].Tile))
                {
                    moveScore[i] += queenValue;
                }

            }

            for (int sorted = 0; sorted < moveList.Count; sorted++)
            {
                int bestScore = int.MinValue;
                int bestScoreIndex = 0;

                for (int i = sorted; i < moveList.Count; i++)
                {
                    if (moveScore[i] > bestScore)
                    {
                        bestScore = moveScore[i];
                        bestScoreIndex = i;
                    }
                }

                // swap

                Move bestMove = moveList[bestScoreIndex];
                moveList[bestScoreIndex] = moveList[sorted];
                moveList[sorted] = bestMove;
            }
        }

        private int Minimax(Board board, int depth, int alpha, int beta, bool
isMaximizingPlayer)
        {
            if (depth == 0)
                return CalculatePoint(board);

            if (isMaximizingPlayer)
            {
                int bestValue = int.MinValue;

                List<Move> possibleMoves = Board.GetAllLegalMoves(Player.Black,
board);

                OrderMoves(possibleMoves, board);
                foreach (var move in possibleMoves)
                {
                    Board newBoard = GenerateMovedBoard(board, move);

                    int value = Minimax(newBoard, depth - 1, alpha, beta,
false);

                    bestValue = Math.Max(value, bestValue);

                    alpha = Math.Max(alpha, value);

                    if (beta <= alpha)
                    {
                        break;
                    }
                }

                return bestValue;
            }
            else
            {
```

```
                int bestValue = int.MaxValue;

                List<Move> possibleMoves = Board.GetAllLegalMoves(Player.White,
board);

                OrderMoves(possibleMoves, board);
                foreach (var move in possibleMoves)
                {
                    Board newBoard = GenerateMovedBoard(board, move);

                    int value = Minimax(board, depth - 1, alpha, beta, true);

                    bestValue = Math.Min(value, bestValue);

                    beta = Math.Min(beta, value);

                    if (beta <= alpha)
                    {
                        break;
                    }
                }

                return bestValue;
            }
        }

        public Move GetBestMove(Board board)
        {
            int bestValue = int.MinValue;
            Move bestMove = null;
            bool turn;
            if (board.Turn == Player.Black)
            {
                turn = false;
            }
            else
            {
                turn = true;
            }

            List<Move> possibleMoves = Board.GetAllLegalMoves(board.Turn,
board);

            OrderMoves(possibleMoves, board);
            foreach (var move in possibleMoves)
            {
                Board newBoard = GenerateMovedBoard(board, move);

                int value = Minimax(newBoard, depth, int.MinValue, int.MaxValue,
turn);

                if (value >= bestValue)
                {
                    bestValue = value;
                    bestMove = move;
                }
            }

            return bestMove;
        }


    }
}
```

Файл Geometry.cs

```
using UnityEngine;

public class Geometry
{
    static public Vector3 PointFromGrid(Vector2Int gridPoint)
    {
        float x = -3.5f + 1.0f * gridPoint.x;
        float z = -3.5f + 1 MoveSelector.0f * gridPoint.y;
        return new Vector3(x, 0, z);
    }

    static public Vector2Int GridPoint(int col, int row)
    {
        return new Vector2Int(col, row);
    }

    static public Vector2Int GridFromPoint(Vector3 point)
    {
        int col = Mathf.FloorToInt(4.0f + point.x);
        int row = Mathf.FloorToInt(4.0f + point.z);
        return new Vector2Int(col, row);
    }
}
```

Файл MoveSelector.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveSelector : MonoBehaviour
{
    public GameObject moveLocationPrefab;
    public GameObject tileHighlightPrefab;
    public GameObject attackLocationPrefab;

    private GameObject tileHighlight;
    private GameObject movingPiece;
    private List<Vector2Int> moveLocations;
    private List<GameObject> locationHighlights;

    void Start ()
    {
        this.enabled = false;
        tileHighlight = Instantiate(tileHighlightPrefab,
Geometry.PointFromGrid(new Vector2Int(0, 0)),
            Quaternion.identity, gameObject.transform);
        tileHighlight.SetActive(false);
    }

    void Update ()
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        if (Input.GetMouseButtonDown(1))
        {
            CancelMove();
            Debug.Log("Cancel Move");
        }
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            //Debug.Log("Hit");
            Vector3 point = hit.point;
            Vector2Int gridPoint = Geometry.GridFromPoint(point);

            tileHighlight.SetActive(true);
            tileHighlight.transform.position =
Geometry.PointFromGrid(gridPoint);
            if (Input.GetMouseButtonDown(0))
            {
                if (!moveLocations.Contains(gridPoint))
                {
                    return;
                }
                if (GameManager.instance.PieceAtGrid(gridPoint) == null)
                {
                    GameManager.instance.Move(movingPiece, gridPoint);
                }
                else
                {
                    GameManager.instance.CapturePieceAt(gridPoint);
                    GameManager.instance.Move(movingPiece, gridPoint);
                }
                ExitState();
            }
        }
        else
        {
            tileHighlight.SetActive(false);
        }
    }
```

```
    private void CancelMove()
    {
        this.enabled = false;

        foreach (GameObject highlight in locationHighlights)
        {
            Destroy(highlight);
        }

        GameManager.instance.DeselectPiece(movingPiece);
        TileSelector selector = GetComponent<TileSelector>();
        selector.EnterState();
    }

    public void EnterState(GameObject piece)
    {
        movingPiece = piece;
        this.enabled = true;

        moveLocations = GameManager.instance.MovesForPiece(movingPiece);
        locationHighlights = new List<GameObject>();

        if (moveLocations.Count == 0)
        {
            CancelMove();
        }

        foreach (Vector2Int loc in moveLocations)
        {
            GameObject highlight;
            if (GameManager.instance.PieceAtGrid(loc))
            {
                highlight = Instantiate(attackLocationPrefab,
Geometry.PointFromGrid(loc), Quaternion.identity, gameObject.transform);
            }
            else
            {
                highlight = Instantiate(moveLocationPrefab,
Geometry.PointFromGrid(loc), Quaternion.identity, gameObject.transform);
            }
            locationHighlights.Add(highlight);
        }
    }

    private void ExitState()
    {
        this.enabled = false;
        TileSelector selector = GetComponent<TileSelector>();
        tileHighlight.SetActive(false);
        GameManager.instance.DeselectPiece(movingPiece);
        movingPiece = null;
        GameManager.instance.NextPlayer();
        selector.EnterState();
        foreach (GameObject highlight in locationHighlights)
        {
            Destroy(highlight);
        }
    }
}
```

Файл NeuralNetwork.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace NeuralNetwork
{
	public class NeuralNet
	{
		public double LearnRate { get; set; }
		public double Momentum { get; set; }
		public List<Neuron> InputLayer { get; set; }
		public List<List<Neuron>> HiddenLayers { get; set; }
		public List<Neuron> OutputLayer { get; set; }

		private static readonly System.Random Random = new System.Random();

		public NeuralNet(int inputSize, int hiddenSize, int outputSize, int
numHiddenLayers = 1, double? learnRate = null, double? momentum = null)
		{
			LearnRate = learnRate ?? .4;
			Momentum = momentum ?? .9;
			InputLayer = new List<Neuron>();
			HiddenLayers = new List<List<Neuron>>();
			OutputLayer = new List<Neuron>();

			for (var i = 0; i < inputSize; i++)
				InputLayer.Add(new Neuron());

		for (int i = 0; i < numHiddenLayers; i++)
		{
			HiddenLayers.Add(new List<Neuron>());
			for (var j = 0; j < hiddenSize; j++)
				HiddenLayers[i].Add(new
Neuron(i==0?InputLayer:HiddenLayers[i-1]));
		}

			for (var i = 0; i < outputSize; i++)
				OutputLayer.Add(new Neuron(HiddenLayers[numHiddenLayers-
1]));
		}

		public void Train(List<DataSet> dataSets, int numEpochs)
		{
			for (var i = 0; i < numEpochs; i++)
			{
				foreach (var dataSet in dataSets)
				{
					ForwardPropagate(dataSet.Values);
					BackPropagate(dataSet.Targets);
				}
			}
		}

		public void Train(List<DataSet> dataSets, double minimumError)
		{
			var error = 1.0;
			var numEpochs = 0;

			while (error > minimumError && numEpochs < int.MaxValue)
			{
				var errors = new List<double>();
				foreach (var dataSet in dataSets)
				{
					ForwardPropagate(dataSet.Values);
```

```
                        BackPropagate(dataSet.Targets);
                        errors.Add(CalculateError(dataSet.Targets));
                    }
                    error = errors.Average();
                    numEpochs++;
                }
            }

            private void ForwardPropagate(params double[] inputs)
            {
                var i = 0;
                InputLayer.ForEach(a => a.Value = inputs[i++]);
                foreach (var layer in HiddenLayers)
                    layer.ForEach(a => a.CalculateValue());
                OutputLayer.ForEach(a => a.CalculateValue());
            }

            private void BackPropagate(params double[] targets)
            {
                var i = 0;
                OutputLayer.ForEach(a => a.CalculateGradient(targets[i++]));
                foreach(var layer in
HiddenLayers.AsEnumerable<List<Neuron>>().Reverse() )
                {
                    layer.ForEach(a => a.CalculateGradient());
                    layer.ForEach(a => a.UpdateWeights(LearnRate,
Momentum));
                }
                OutputLayer.ForEach(a => a.UpdateWeights(LearnRate,
Momentum));
            }

            public double[] Compute(params double[] inputs)
            {
                ForwardPropagate(inputs);
                return OutputLayer.Select(a => a.Value).ToArray();
            }

            private double CalculateError(params double[] targets)
            {
                var i = 0;
                return OutputLayer.Sum(a =>
Mathf.Abs((float)a.CalculateError(targets[i++])));
            }

            public static double GetRandom()
            {
                return 2 * Random.NextDouble() - 1;
            }
        }

        public enum TrainingType
        {
            Epoch,
            MinimumError
        }

}
```

Файл Neuron.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace NeuralNetwork
{
	public class Neuron
	{
		public List<Synapse> InputSynapses { get; set; }
		public List<Synapse> OutputSynapses { get; set; }
		public double Bias { get; set; }
		public double BiasDelta { get; set; }
		public double Gradient { get; set; }
		public double Value { get; set; }

		public Neuron()
			{
				InputSynapses = new List<Synapse>();
				OutputSynapses = new List<Synapse>();
				Bias = NeuralNet.GetRandom();
			}

			public Neuron(IEnumerable<Neuron> inputNeurons) : this()
			{
				foreach (var inputNeuron in inputNeurons)
				{
					var synapse = new Synapse(inputNeuron, this);
					inputNeuron.OutputSynapses.Add(synapse);
					InputSynapses.Add(synapse);
				}
			}

			public virtual double CalculateValue()
			{
				return Value = Sigmoid.Output(InputSynapses.Sum(a =>
a.Weight * a.InputNeuron.Value) + Bias);
			}

			public double CalculateError(double target)
			{
				return target - Value;
			}

			public double CalculateGradient(double? target = null)
			{
				if(target == null)
					return Gradient = OutputSynapses.Sum(a =>
a.OutputNeuron.Gradient * a.Weight) * Sigmoid.Derivative(Value);

				return Gradient = CalculateError(target.Value) *
Sigmoid.Derivative(Value);
			}

			public void UpdateWeights(double learnRate, double momentum)
			{
				var prevDelta = BiasDelta;
				BiasDelta = learnRate * Gradient;
				Bias += BiasDelta + momentum * prevDelta;

				foreach (var synapse in InputSynapses)
				{
					prevDelta = synapse.WeightDelta;
					synapse.WeightDelta = learnRate * Gradient *
synapse.InputNeuron.Value;
```

```
                                    synapse.Weight += synapse.WeightDelta + momentum *
prevDelta;
                              }
                        }

      }

      public class Synapse
      {
            public Neuron InputNeuron { get; set; }
            public Neuron OutputNeuron { get; set; }
            public double Weight { get; set; }
            public double WeightDelta { get; set; }

            public Synapse(Neuron inputNeuron, Neuron outputNeuron)
            {
                  InputNeuron = inputNeuron;
                  OutputNeuron = outputNeuron;
                  Weight = NeuralNet.GetRandom();
            }
      }

      public static class Sigmoid
      {
            public static double Output(double x)
            {
                  return x < -45.0 ? 0.0 : x > 45.0 ? 1.0 : 1.0 / (1.0 +
Mathf.Exp((float)-x));
            }

            public static double Derivative(double x)
            {
                  return x * (1 - x);
            }
      }

      public class DataSet
      {
            public double[] Values { get; set; }
            public double[] Targets { get; set; }

            public DataSet(double[] values, double[] targets)
            {
                  Values = values;
                  Targets = targets;
            }
      }

}
```