МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ПРОГРАМНА РЕАЛІЗАЦІЯ МАТЕМАТИЧНИХ АЛГОРИТМІВ ШАХОВОЇ ГРИ»**

Виконав: студент 2 курсу, групи 8.1132
спеціальності 113 Прикладна математика
(шифр і назва спеціальності)
освітньої програми Прикладна математика
(назва освітньої програми)
М. В. Бончев
(ініціали та прізвище)
Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти магістр

Спеціальність 113 Прикладна математика
(шифр і назва)

Освітня програма Прикладна математика

ЗАТВЕРДЖУЮ

Завідувач кафедри
фундаментальної та прикладної
математики, професор, д.т.н.

_____ Гребенюк С.М.
(підпис)

« ____ » _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бончеву Максиму Володимировичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Програмна реалізація математичних алгоритмів шахової гри

керівник роботи (проекту) Гребенюк Сергій Миколайович, професор, д.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «____» _____ 2023 року № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Основні поняття та властивості математичних алгоритмів розробки шахової гри
2. Реалізація штучного інтелекту

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.05.2023	
2.	Збір вихідних даних.	26.05.2023	
3.	Обробка методичних та теоретичних	16.06.2023	
	джерел.		
4.	Розробка програмного забезпечення	29.09.2023	
5.	Опис алгоритмів та реалізації програмного забезпечення	03.10.2023	
6.	Оформлення та нормоконтроль	10.11.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.		

Студент _____ М. В. Бончев
(підпис) (ініціали та прізвище)

Керівник роботи _____ С. М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г. Спиця

РЕФЕРАТ

Кваліфікаційна робота магістра «Програмна реалізація математичних алгоритмів шахової гри»: 54 с., 9 рис., 11 джерел.

АЛГОРИТМИ, АРХІТЕКТУРА ПРОЕКТУ, ВІДСІЧЕННЯ АЛЬФА-БЕТА, ОПТИМІЗАЦІЯ, ПАТЕРНИ ПРОЕКТУВАННЯ, РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ, РОЗРОБКА ШАХОВОЇ ГРИ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єкт дослідження – Програмна реалізація шахової гри, її алгоритмів та алгоритмів штучного інтелекту

Мета роботи: Розробити програмне забезпечення, з допомогою якого користувач зможе грати в шахи з іншим користувачем на одному пристрої, чи з шаховим штучним інтелектом

Метод дослідження – аналітичний, чисельний, порівняльний.

У даній дипломній роботі виконано комплексне дослідження та реалізацію ігрового додатка для гри в шахи з використанням штучного інтелекту. Робота охоплює наступні ключові аспекти: досліджено та аналізовано основні особливості розробки шахової гри з впровадженим штучним інтелектом. Визначено складність врахування всіх можливих ходів та стратегій гри, що вимагає розробки ефективних алгоритмів. Розглянуто усі можливі способи реалізації даного програмного забезпечення, включаючи різні мови програмування, середовища розробки тощо. Розроблено графічний інтерфейс гри на базі бібліотеки SDL, що дозволяє користувачу зручно взаємодіяти з ігровим полем, вибирати фігури та виконувати ходи. Проведено детальний аналіз і реалізацію алгоритму Minimax з оптимізацією Alpha-Beta Pruning для реалізації штучного інтелекту гравця. Розроблений алгоритм здатний визначати оптимальні ходи в режимі реального часу. Описано взаємодію ігрового додатка з користувачем та штучним інтелектом.

SUMMARY

Master's Qualification Thesis "Software implementation of mathematical algorithms of the chess game": 54 pages, 9 figures, 11 sources.

ALGORITHMS, PROJECT ARCHITECTURE, ALPHA-BETA CUTOFF, OPTIMIZATION, DESIGN PATTERNS, GUI DEVELOPMENT, CHESS GAME DEVELOPMENT, ARTIFICIAL INTELLIGENCE.

The object of research is the software implementation of the chess game, its algorithms and algorithms of artificial intelligence

The purpose of the work: To develop software that will allow a user to play chess with another user on the same device, or with a chess artificial intelligence

The research method is analytical, numerical, comparative.

In this work, a comprehensive study and implementation of a game application for playing chess using artificial intelligence was performed. The work covers the following key aspects: the main features of the development of a chess game with artificial intelligence have been studied and analyzed. The complexity of taking into account all possible moves and strategies of the game is determined, which requires the development of effective algorithms. All possible ways of implementing this software were considered, including various programming languages, development environments, etc. The graphic interface of the game was developed based on the SDL library, which allows the user to conveniently interact with the playing field, select pieces and make moves. A detailed analysis and implementation of the Minimax algorithm with Alpha-Beta Pruning optimization was carried out to implement the player's artificial intelligence. The developed algorithm is capable of determining optimal moves in real time. The interaction of the game application with the user and artificial intelligence is described.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

UI – графічний інтерфейс користувача.

AI або ШІ – Штучний інтелект

ООП – Об'єктно-орієнтоване програмування

DLL – бібліотека динамічної компоновки.

API – Прикладний програмний інтерфейс додатку

MVP – шаблон проектування.

MVC – шаблон проектування.

SDL – Мультимедійна бібліотека

Фреймворк – зовнішній програмний продукт

ВСТУП

Шахи є однією з найстаріших та найпопулярніших настільних ігор, які привертають увагу тисяч ігрових ентузіастів з усього світу. Ця гра вважається символом стратегії та логіки, а також сприяє розвитку креативності та аналітичних здібностей. Історія шахів сягає далеко в минуле і пов'язана з різними культурами та цивілізаціями.

В давнину шахи вважалися привілеєм правителів та інтелектуалів, але з плином часу вони стали доступні всім верствам суспільства, ставши важливою частиною глобальної ігрової культури. Ця гра захоплює і зацікавлює гравців усіх вікових категорій, що створює величезну спільноту шахістів, яка активно спілкується та змагається у своїх змаганнях.

Метою цієї дипломної роботи є розробка шахового програмного додатку з допомогою математичних алгоритмів, який дозволить користувачам грати у шахи як із комп'ютером-супротивником, так і самостійно, забезпечуючи при цьому зручний та інтуїтивний графічний інтерфейс.

Актуальність теми полягає в тому, що шахи залишаються популярними та захоплюючими для багатьох людей, і їх розвиток та впровадження у формі програмного додатку дає можливість розширити та покращити ігровий досвід для користувачів. Крім того такий проєкт є цікавим, тому що для розробки такого програмного забезпечення потрібно використовувати багато математичних алгоритмів, патернів проєктування та принципів ООП, щоб забезпечити зручну роботу з кодом.

Робота складається з трьох основних розділів. Перший розділ містить основні задачі та вимоги до розроблюваного програмного додатку. Зокрема, описані особливості розробки шахової гри, які включають правила гри, хід фігур, збереження стану дошки тощо. Також розглянута задача реалізації шахового

штучного інтелекту, який взаємодіє з користувачем та здатний приймати рішення на основі обраних алгоритмів. Другий розділ присвячений вибору оптимальних методів та засобів для розробки шахового програмного комплексу. Розглянуті різні варіанти архітектури програмного комплексу. Крім того, розглянуті різні інструменти розробки. Після аналізу задачі були виявлені основні задачі, серед яких реалізація графічного інтерфейсу, розробка алгоритму та взаємодія с користувачем. Для таких задач були виокремлені мова програмування C++, зручне програмне середовище Visual Studio, а також бібліотека SDL для реалізації графічного інтерфейсу. І у третьому розділі детально описана архітектура програми та структура проєкту, включаючи класи та методи, які використовуються для реалізації шахового програмного додатку

Програмний додаток розроблявся з використанням мови програмування C++, бібліотеки SDL та середовище розробки Visual Studio 2019.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ШАХОВОЇ ГРИ З ШТУЧНИМ ІНТЕЛЕКТОМ

Метою проекту є розробка повноцінного шахового програмного додатку, який надасть можливість гравцеві грати в шахи як з штучним інтелектом, так і проти себе самого, за допомогою зручного та інтуїтивно зрозумілого графічного інтерфейсу. Програмний засіб розробляється для персональних комп’ютерів на базі операційної системи Windows XP / 7 / 10.

Як вхідні дані програма отримує вибір користувача, який може обрати режим гри з ботом або противника на одному комп'ютері. Для гри проти комп'ютерного супротивника користувач також має можливість вибрати рівень складності штучного інтелекту.

Основні можливості програмного додатку включають в себе реалізацію правил гри в шахах, взаємодію з користувачем через графічний інтерфейс, відслідковування правильності ходів та штучний інтелект.

1.1 Особливості розробки шахової гри

Розробка шахової гри з штучним інтелектом є складним та цікавим завданням, оскільки вона вимагає вирішення декількох технічних та алгоритмічних проблем.

Однією з ключових особливостей розробки шахової гри є необхідність враховувати складні правила гри, які включають рухи різних фігур, можливість рокіровки, проходження пішака та інші особливості. Для забезпечення реалістичності гри, важливо коректно обробляти всі можливі комбінації ходів і заборонити некоректні ходи для кожної фігури. Усі можливі ходи показано на рис. 1.1.

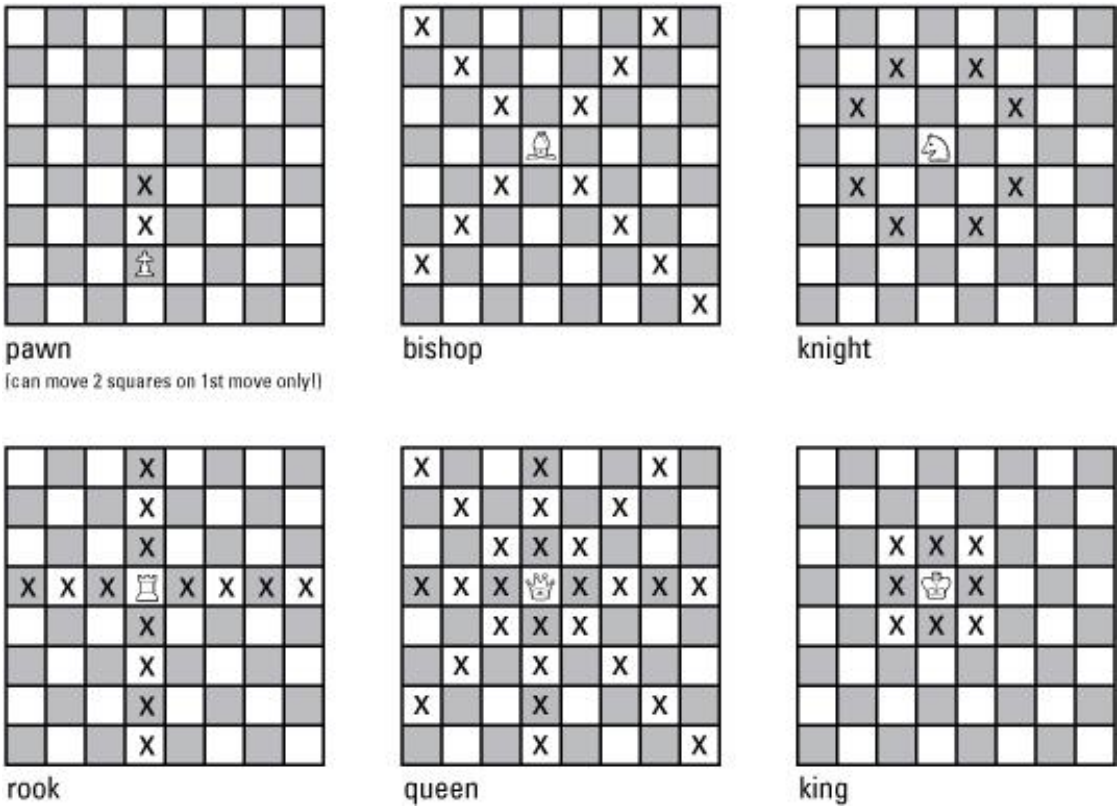

Рисунок 1.1 – Можливі ходи усіх фігур [2]

Шахова гра може мати велику кількість обчислень та перевірок стану дошки на кожен хід, що може призвести до значних навантажень на процесор. Оптимізація алгоритмів та структур даних є необхідним для забезпечення плавної роботи гри та запобігання затримок. Тому важливо перевіряти можливі ходи лише один раз, коли це необхідно.

Тестування: Велика увага повинна бути приділена тестуванню гри на правильність роботи, перевірці правильності ходів, а також виявленню можливих помилок та ситуацій, що виникають під час гри.

1.2 Користувацький інтерфейс системи

Розробка зручного та інтуїтивно зрозумілого користувацького інтерфейсу (UI[1]) є важливою частиною проєкту. Гра повинна мати привабливий та

естетичний дизайн, логічно розміщені елементи управління та дружній інтерфейс, щоб гравці могли легко взаємодіяти з грою. Користувацький інтерфейс також має надавати інформацію про стан гри, можливості ходів і дозволяти здійснювати ходи гравця.

Головна ідея стилістики додатку – максимальна простота інтерфейсу. Разом з тим дизайн спроєктовано в стилі Flat UI, що зараз є провідним стилем багатьох застосунків, в тому числі desktop-сервісів. Він базується на принципах швейцарського стилю, в основі якого лежить типографіка як спосіб виключити зайві деталі інтерфейсу. Простота та мінімалізм є основою візуального оформлення такого проєкту та забезпечує легкість сприйняття інформації, а також інтуїтивно зрозумілий інтерфейс для користувачів. [9]

1.3 Розробка шахового штучного інтелекту

Основною метою розробки штучного інтелекту є створення програмного агента, який здатен грати в шахи.

Завдання AI в шахах вимагає застосування різноманітних алгоритмів та методів штучного інтелекту. Одним із найпопулярніших алгоритмів для розробки AI є алгоритм Minimax. Цей алгоритм дозволяє оцінити різні ходи гри і вибрати оптимальний хід, передбачаючи ходи як гравця, так і AI.

Для реалізації AI в шахах можуть бути використані такі методи:

- перебір всіх можливих ходів: Цей метод передбачає перебір всіх можливих ходів гри та оцінку кожного ходу за допомогою функції оцінки. Однак такий підхід може бути дуже обчислювально витратним і має обмеження на глибину дерева пошуку;
- алгоритм Minimax з оптимізацією Alpha-Beta Pruning: Цей метод дозволяє зменшити кількість гілок дерева пошуку, що дозволяє значно

підвищити продуктивність AI;

- використання нейронних мереж: Нейронні мережі можуть бути використані для навчання AI на основі історичних даних гри та вироблення оптимальних стратегій. Але в цьому проєкті це було б нелогічно, оскільки розробка та тренування нейронної мережі зайняла би занадто багато часу;
- використання генетичних алгоритмів: Генетичні алгоритми дозволяють еволюціонувати стратегії AI та покращувати їх в процесі гри.

Для досягнення мети у цьому проєкті будуть використані алгоритми штучного інтелекту Minimax і альфа-бета відсічення. Ці алгоритми дозволяють здійснювати оптимальні ходи на основі пошуку найкращого можливого рішення в грі.

Розробка AI в шахах є складною задачею, оскільки вона вимагає поєднання комп'ютерних наук, математики та стратегічного мислення. Проте вона дозволяє створити гру зі значною ступенем складності та забезпечити захоплюючий та цікавий досвід для гравців.

1.4 Огляд існуючих програмних рішень

Історія створення першої гри в шахи на комп'ютері налічує вже понад півстоліття. Одним із перших успішних ігор був шаховий симулятор на мейнфреймах IBM у 1950-х роках. Протягом наступних десятиліть ігри в шахи стали все більш поширеними на різних платформах, і в даний час існує велика кількість програмних рішень для гри в шахи на різних пристроях.

Перші штучні інтелекти у шахах базувались на простих правилах та переборі всіх можливих ходів. Перші шахівні штучні інтелекти з'явилися вже в 1950-х роках. У 1951 році був розроблений один із перших шахових штучних

інтелектів – програма NSS (National Security Service), яку створив Клауд Шеннон. Вона використовувала обмежені можливості перебору ходів та мала низьку ефективність, але була першим кроком до створення більш складних шахових штучних інтелектів.

Нарешті, у 1957 році інженер IBM на ім’я Алекс Бернштейн створив перший у світі повністю автоматизований шаховий механізм. Механізм був створений для мейнфрейму IBM 704 і займав приблизно вісім хвилин на хід (рис. 1.2).

Рисунок 1.2 – Шахова програма IBM, 1957 рік, працює на комп'ютері IBM 704

У наступні десятиліття із зростанням потужності комп'ютерів та розвитком алгоритмів шахових штучних інтелектів, програми ставали все сильнішими та більш вдосконаленими. В 1970-х роках з'явилися перші програми, що використовували алгоритм Мінімакс для обчислення оптимальних ходів. В 1980-х роках з'явилися програми, які використовували більші обмежені методи штучного інтелекту, такі як Alpha-Beta Pruning, для покращення продуктивності обчислень.

Сучасні шахівні штучні інтелекти використовують комбінацію різних методів та алгоритмів для досягнення найкращих результатів. Вони використовують техніки глибокого навчання, розроблені на основі штучних нейронних мереж, в поєднанні з традиційними алгоритмами, такими як Мінімакс та Alpha-Beta Pruning, для ефективного пошуку оптимальних ходів.

Серед найпопулярніших і сильних шахових штучних інтелектів в наш час можна виділити програму Stockfish. Вона базується на комбінації традиційних алгоритмів та глибоких нейронних мереж, що дозволяє їй досягати високої продуктивності та сильної гри в шахах.

Сучасними популярними платформами для гри в шахи є онлайн-шахівні сайти та мобільні додатки. Деякі з найпопулярніших сайтів, які надають можливість грати в шахи онлайн, включають Chess.com, lichess.org, chess24.com, і ICC (Internet Chess Club).

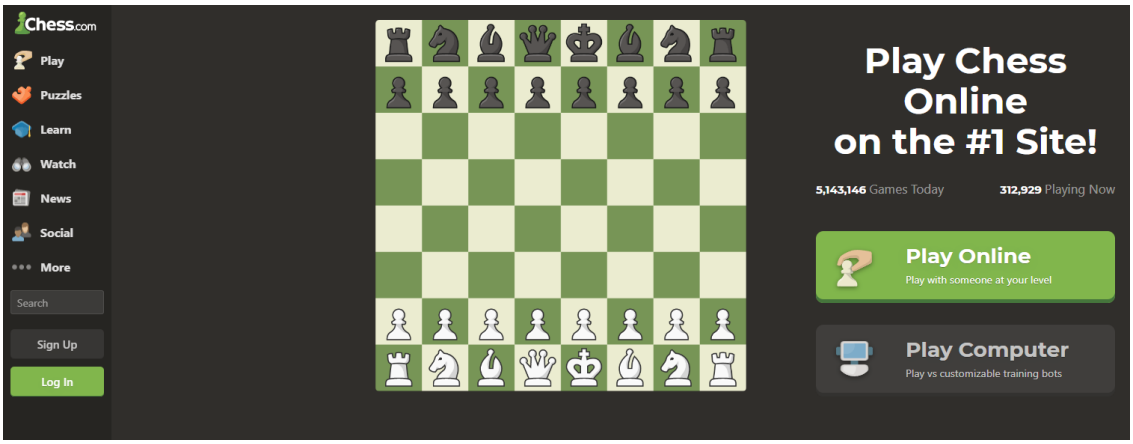

Рисунок 1.3 – Інтерфейс сайту для гри в шахи Chess.Com

2 МЕТОДИ ТА ЗАСОБИ РОЗРОБКИ

2.1 Вибір архітектури програмного комплексу

Вибір архітектури програмного комплексу є критично важливим етапом розробки шахового додатку з штучним інтелектом. Особливості шахової гри, зокрема різні типи фігур та їх рухи, вимагають структурованого підходу до розробки, щоб забезпечити зручність утримання коду та підтримку можливості розширення у майбутньому.

Для досягнення належної структурованості програмної архітектури в цій роботі використовується об'єктно-орієнтований підхід (ООП). Об'єктно-орієнтоване програмування базується на трьох основних принципах, які допомагають у створенні ефективної та організованої архітектури:

Інкапсуляція – це об’єднання в межах класу певних даних і методів для роботи з ними.

Наслідування – це можливість створення нового класу (нащадка) на базі існуючого (базового).

Поліморфізм – це різна поведінка методу в різних класах (рис. 2.1). Методи з однаковим ім’ям, описані в різних класах, можуть мати різну реалізацію. [1]

За такого підходу кожна фігура шахової гри може бути представлена в окремому класі, який містить всі необхідні дані та методи для обробки рухів цієї фігури. Наприклад, можуть бути класи для пішаків, коней, слонів та інших фігур. Кожен з цих класів містить методи для обчислення можливих ходів, перевірки легальності ходів та інші функції, пов'язані з цією фігурою.

Після розробки окремих класів фігур, в скрипті Game об'єднати ці фігури в об'єкт, що представляє шахову дошку. Цей об'єкт містить інформацію про розміщення фігур на дошці та методи для переміщення фігур з одного місця на

інше. Реалізація самої гри представлена в скрипті MainLoop, в якому ініціалізується скрипт SDL_Handler для графічного інтерфейсу, и кожен раз запрошує у гравця хід, потім передає інформацію до Game, да обновляє графічний інтерфейс.

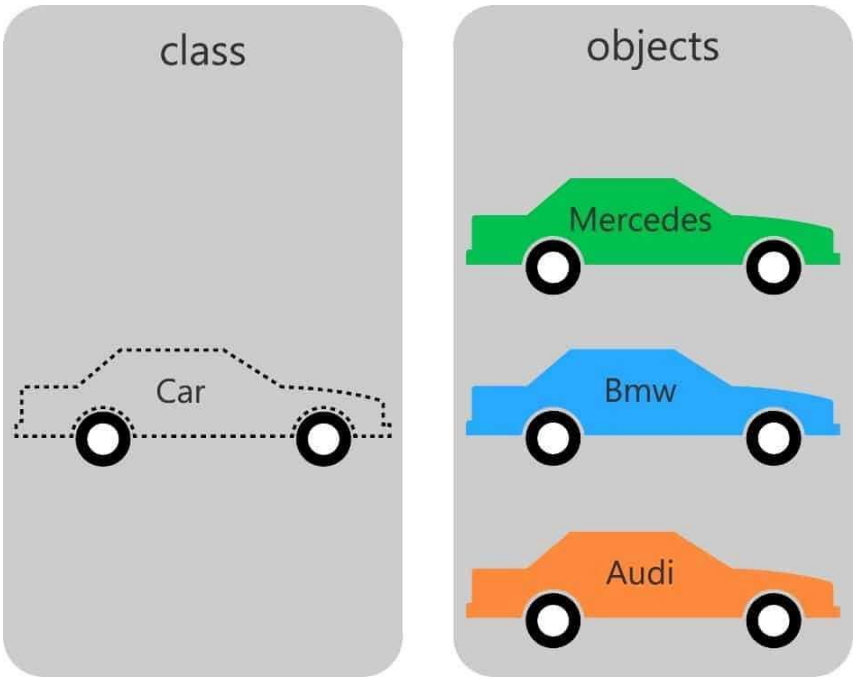

Рисунок 2.1 – Класи та об'єкти в ООП [5]

Крім того, архітектура передбачає окремий клас для штучного інтелекту, який використовує алгоритми Minimax і альфа-бета відсічення для прийняття оптимальних рішень під час гри. Цей клас здійснює аналіз можливих ходів, оцінку позиції на дошці та обчислення кращого ходу для комп'ютерного суперника. Цей клас розташований в папці ChessAI.

Такий підхід до архітектури дозволяє забезпечити збалансовану та легко розширювальну систему, яка буде зручно утримуватись та розвиватись у майбутньому. Кожен клас відповідатиме за конкретний аспект гри, а зв'язки між класами дозволять координувати рухи фігур та оцінювати стан гри, щоб забезпечити ефективний та реалістичний шаховий досвід для користувачів.

2.2 Опис архітектури користувацького інтерфейсу системи

Основною частиною архітектури користувацького інтерфейсу є клас SDL_Handler, який відповідає за взаємодію з бібліотекою SDL та управління відображенням графічного інтерфейсу на екрані.

Клас SDL_Handler містить методи для рендерингу графічних елементів, таких як шахова дошка, фігури, підсвічення можливих ходів та інші елементи інтерфейсу. Він забезпечує взаємодію з класами, відповідальними за логіку гри, та передає їм інформацію про дії користувача.

Архітектура користувацького інтерфейсу передбачає використання паттерну проєктування "Модель-Вид-Контролер" (MVC) (рис. 2.2).

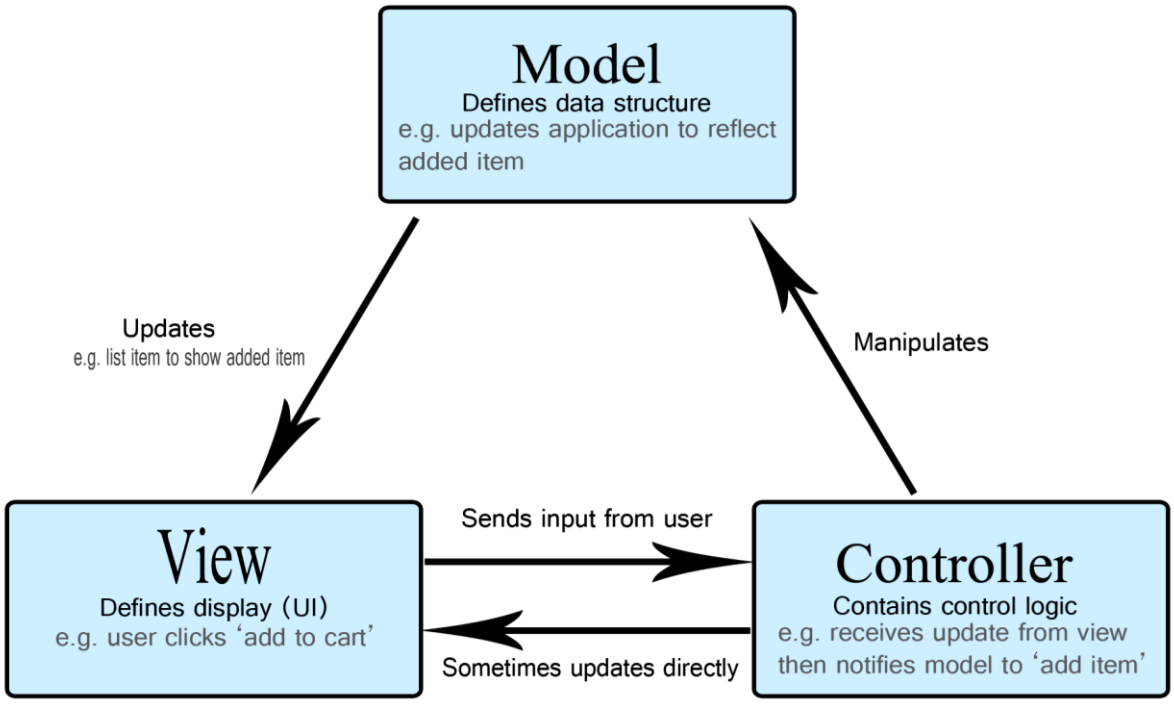

Рисунок 2.2 – MVC паттерн [9]

Відповідно до цього підходу, клас SDL_Handler виконує роль "Виду", який відповідає за відображення інформації та реагує на дії користувача. Він не несе

на собі логіку гри, а лише передає дії користувача відповідному "Контролеру" або "Моделі".

Клас "Контролер" відповідає за обробку дій користувача, таких як вибір фігури, обробка кліків на дошці тощо. Він взаємодіє з "Моделлю", яка містить логіку гри, правила руху фігур, а також взаємодію з алгоритмом штучного інтелекту для визначення відповіді комп'ютерного суперника.

Така архітектура дозволяє розділити взаємодію з графічним інтерфейсом та логіку гри, що робить код більш зрозумілим, модульним та підтримуваним. Крім того, за допомогою класу SDL_Handler можна реалізувати відображення анімації, ефектів та інших елементів, що робить гру більш привабливою для користувача.

2.3 Вибір алгоритму для штучного інтелекту

Вибір алгоритму для штучного інтелекту є важливим етапом в розробці шахової гри з штучним інтелектом. У даній роботі обрано алгоритм Minimax з оптимізацією Alpha-Beta pruning.

Мінімакс (MinMax) алгоритм.

Мінімакс – це свого роду алгоритм відстеження, який використовується в процесі прийняття рішень і теорії ігор, щоб знайти оптимальний хід для гравця, припускаючи, що ваш опонент також грає оптимально. Він широко використовується в покрокових іграх для двох гравців, таких як Tic-Tac-Toe, Backgammon, Mancala, Chess тощо.

У Minimax два гравці називаються максимайзером і мінімізатором. Максимізатор намагається отримати найвищу можливу оцінку, а мінімізатор намагається зробити навпаки та отримати найнижчу можливу оцінку.

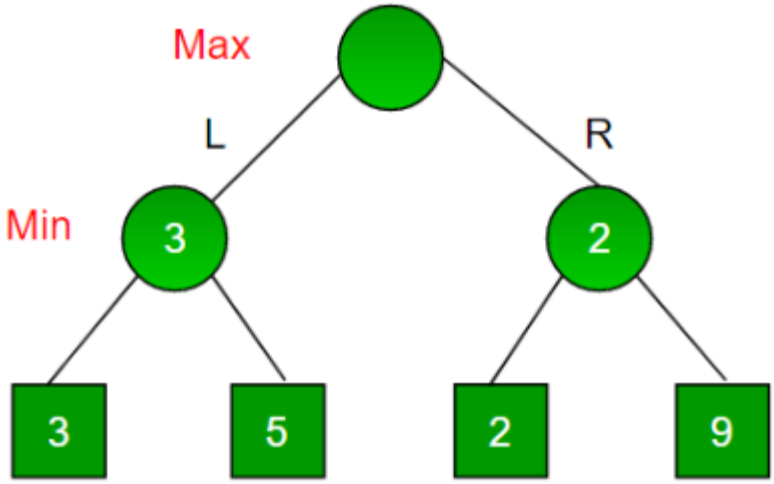

Рисунок 2.3 – Приклад роботи Minimax алгоритму

Кожен стан дошки має значення, пов’язане з ним. У даному стані, якщо максимайзер має перевагу, тоді рахунок на дошці матиме деяке додатне значення. Якщо мінімізатор має перевагу в такому стані дошки, тоді воно матиме деяке від’ємне значення. Значення дошки розраховуються за допомогою деяких евристик, які є унікальними для кожного типу гри, й на ідеї мінімізації можливої втрати та максимізації можливого прибутку для кожного ходу. Цінність кожної фігури показано на рисунку 2.4. [11]

Оптимізація Alpha-Beta pruning

Alpha-Beta pruning – це метод оптимізації Minimax алгоритму, який дозволяє значно зменшити кількість гілок дерева гри, що обробляються. Альфа-бета-відсікання насправді не є новим алгоритмом, а радше технікою оптимізації мінімаксного алгоритму. Це значно скорочує час обчислень. Це дозволяє шукати набагато швидше і навіть переходити на глибші рівні в дереві гри. Він обрізає гілки в дереві гри, які не потрібно шукати, оскільки вже існує кращий доступний хід. Це називається скороченням альфа-бета, оскільки воно передає 2 додаткові параметри у функції minimax, а саме альфа та бета.

На рисунку 2.5 показує приклад Альфа-Бета відсікання. Тут піддерева, виділені сірим кольором, не потрібно досліджувати (коли ходи оцінюються зліва

направо), оскільки відомо, що група піддерев у цілому дає значення еквівалентного піддерева або гірше, і тому не може впливати на кінцевий результат. Максимальний і мінімальний рівні представляють хід гравця та супротивника відповідно.

Рисунок 2.4 – Цінність фігур

Альтернативи вибору

Існують інші алгоритми для штучного інтелекту, такі як Monte Carlo Tree Search (MCTS), Expectimax і інші. Кожен з цих алгоритмів має свої переваги і недоліки, і вибір залежить від специфіки гри і ресурсів, доступних для обчислень.

Отже, вибір алгоритму Minimax з оптимізацією Alpha-Beta pruning є вдалою стратегією для реалізації штучного інтелекту шахової гри, оскільки він забезпечує ефективне прийняття рішень та досягнення заданого рівня гри для користувача.

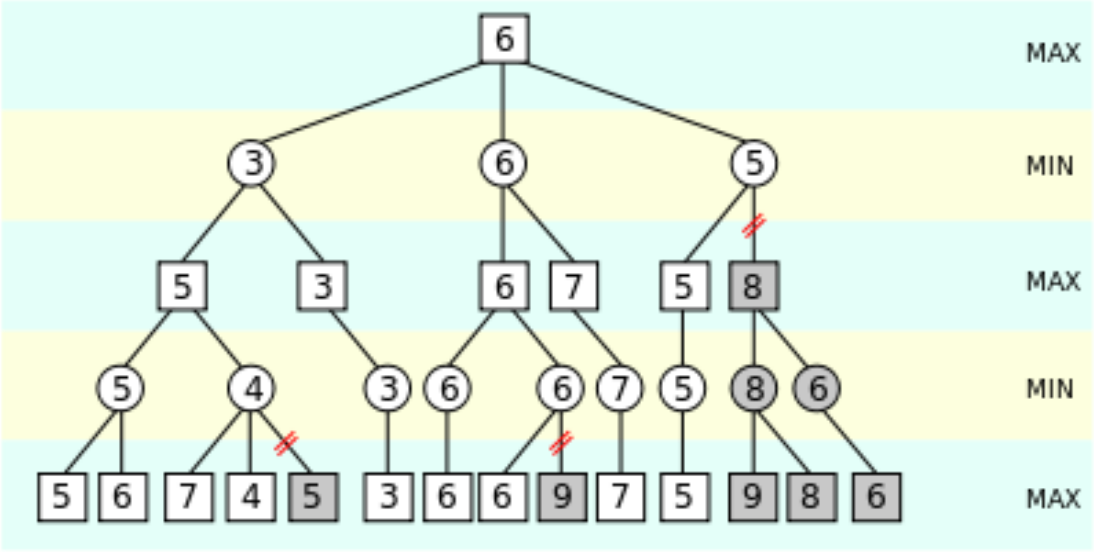

Рисунок 2.5 – Приклад Альфа-Бета відсікання [10]

2.4 Опис інструментів розробки

2.4.1 Мова програмування C++

Вибір мови програмування є важливим аспектом при виконанні кваліфікаційної роботи. Мова програмування C++ була обрана з декількох причин.

Мова програмування C++ є потужною і високорівневою мовою програмування, що спрощує розробку складних програм і надає розширені можливості для ефективної оптимізації коду. Розробка C++ була започаткована у 1979 році Бьорном Страуструпом як розширення мови C, і вона швидко здобула популярність серед програмістів та розробників. [3]

Однією з ключових переваг мови C++ є її об'єктно-орієнтований підхід, що дозволяє розробникам створювати класи і об'єкти, що забезпечують кращу структуризацію коду і забезпечують більшу повторюваність використання коду.

C++ також включає підтримку поліморфізму, інкапсуляції, наслідування та абстракції – ключових понять об'єктно-орієнтованого програмування. Це дозволяє створювати ефективний і структурований код, що сприяє легшій розширюваності та модифікації програм. Крім того, мова підтримує інші парадигми програмування, такі як процедурне програмування і забезпечує багато різних інструментів для розробки різноманітних застосунків.

Ще однією перевагою C++ є його висока продуктивність та швидкодія, що робить його ідеальним вибором для розробки ігрових програм, таких як шахи з штучним інтелектом. Із застосуванням оптимізацій та розумних алгоритмів, реалізація шахів з штучним інтелектом на C++ може працювати швидко та ефективно, забезпечуючи високу продуктивність гри.

Серед інших переваг мови C++ можна відзначити його широку підтримку і велику спільноту розробників. Існує багато ресурсів, документації та бібліотек, таких як SDL які допомагають розробникам створювати потужні програми на цій мові. Також в C++ є багата стандартна бібліотека (STL), яка надає багато готових інструментів для роботи зі змінними, рядками, контейнерами, алгоритмами, потоками та іншими основними складовими програм. [4]

Обираючи мову C++ для розробки шахів, можна зазначити її потужність, гнучкість і ефективність, що дозволить реалізувати проєкт шахової гри з штучним інтелектом на високому рівні, забезпечуючи приємний користувацький досвід та швидку гру. Мова C++ вже успішно використовується в багатьох ігрових розробках, і вона ідеально підходить для реалізації шахової гри з використанням штучного інтелекту.

2.4.2 Огляд програмного середовища Visual Studio та можливі альтернативи

Microsoft Visual Studio [6] – один з продуктів компанії Майкрософт, який представляє з себе інтегровану середу розробки програмного забезпечення і ряд інших інструментальних засобів. Даний продукт дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в не керованому, так і керованому кодах для всіх платформ, підтримуваних Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework і Microsoft Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторингу коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і як відладчик машинного рівня. Решта вбудованих інструментів включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду, додавання нових наборів інструментів, наприклад, для редагування і візуального проєктування коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів циклу розробки програмного забезпечення. Загальний вигляд вікна, яке відображається при завантаженні Visual Studio, початок роботи та створення нового проєкту (рис. 2.6) [6]

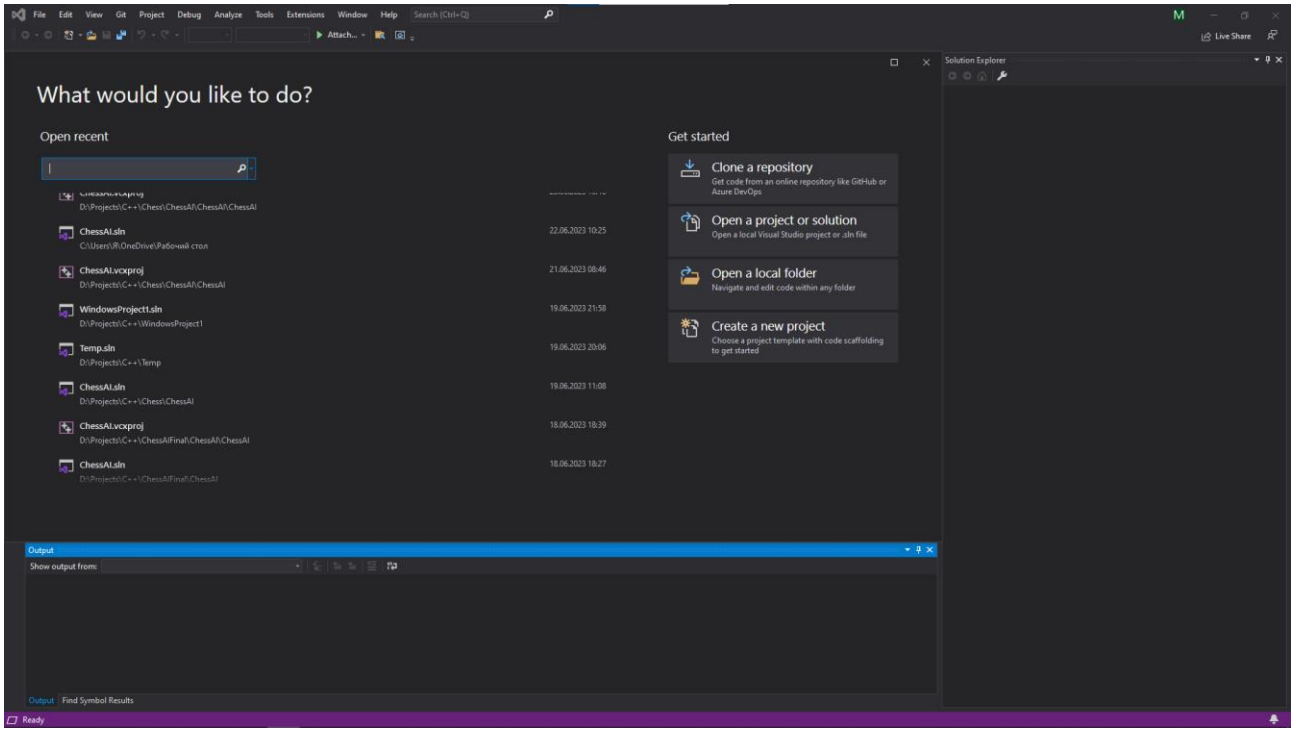

Рисунок 2.6 – Початковий екран Visual Studio

2.4.3 Огляд бібліотеки SDL та її можливостей для реалізації графічного інтерфейсу

SDL (Simple DirectMedia Layer) – це крос-платформна бібліотека, яка була розроблена для спрощення створення мультимедійних додатків, зокрема ігор, на різних платформах. Вона була розроблена Джейсоном МакЛіном у 1997 році, а в даний час активно підтримується і розвивається спільнотою розробників.

Основні можливості бібліотеки SDL:

- відео та аудіо вивід: SDL дозволяє легко виводити графіку та звук на екрані та аудіо в простий та ефективний спосіб, незалежно від платформи;
- управління вікном: Бібліотека SDL надає можливості для створення та керування вікном програми, що є важливим елементом графічних

інтерфейсів;

- робота з клавіатурою та мишею: SDL надає можливість легко взаємодіяти з клавіатурою та мишею, що дозволяє реалізувати обробку введення користувача;
- підтримка графічних форматів: Бібліотека підтримує різні графічні формати, такі як BMP, PNG, JPEG та інші;
- портабельність: Однією з основних переваг SDL є те, що вона є крос платформною. Це дозволяє писати код один раз і запускати його на різних операційних системах, таких як Windows, Linux, MacOS та інші.

Можливі альтернативи бібліотеці SDL включають SFML (Simple and Fast Multimedia Library), Allegro та інші. Кожна з них має свої переваги та особливості, і вибір бібліотеки залежить від потреб проєкту та власних вподобань розробника. Однак SDL є популярною та добре документованою бібліотекою, що робить її зручним вибором для реалізації графічного інтерфейсу в шаховій грі на мові C++ з використанням SDL.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Архітектура програми та структура проєкту

У розробленому шахівному проєкті використовується ООП-підхід, що дозволяє організувати код у логічні та окремі блоки, що зберігають свою функціональність. Програмний код проєкту структурований таким чином, щоб забезпечити легке розуміння та розширення.

Головним класом проєкту є MainLoop, який відповідає за контроль основних процесів шахової гри. Цей клас обробляє введення користувача, здійснює взаємодію з шахівним інтелектом та керує логікою гри.

Кожна шахова фігура має свій власний клас (наприклад, клас Pawn, клас Rook, клас Bishop тощо), який наслідує загальний клас Piece. Клас Piece містить загальні атрибути та методи, які характеризують кожну фігуру у грі. Кожен окремий клас фігури реалізує свої власні алгоритми для обчислення можливих ходів та взаємодії з дошкою. Базовий клас Piece має наступні методи:

```
enum Team { BLACK, WHITE, NONE };
enum PieceType { PAWN, ROOK, KNIGHT, BISHOP, KING, QUEEN, EMPTY };
```

Налаштування типу фігури та команди, за яку грає ця фігура

```
Piece(Team team, Point pos, SDL_Handler* handler, PieceType type);
```

Конструктор класу, в якому ініціалізується команда, за яку грає фігура та тип фігури

```
virtual void calcPossibleMoves(Piece* field[8][8], bool checkCheck) = 0;
```

Це головний метод, який прораховує усі можливі ходи, приймаючи стан дошки

std::vector<PossibleMove> getPossibleMoves() { return m_possibleMoves; };
Цей метод повертає усі можливі ходи данної фігури. Використовується, коли користувач натискає на фігуру

Шаховий інтелект в проєкті реалізовано в окремій папці ChessAI, що містить класи для кожної фігури, minimax алгоритму та mainAI скрипту. Ці класи дозволяють обчислювати оптимальний хід для комп'ютерного гравця на основі оцінки стану гри та пошуку найкращого ходу.

Загальна структура проєкту виглядає наступним чином:

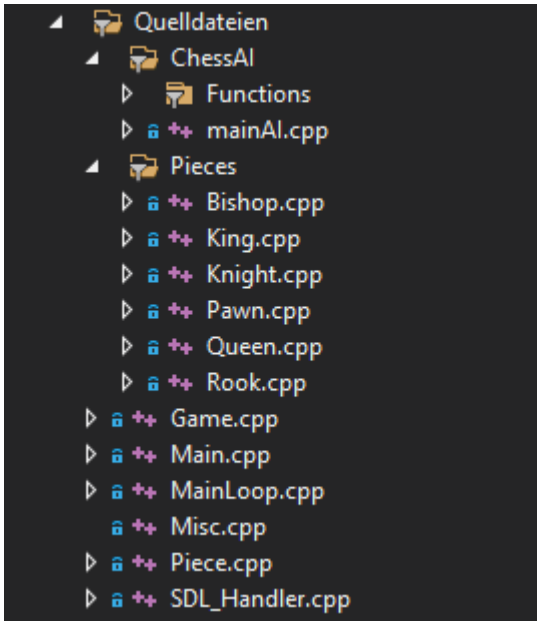

Рисунок 3.1 – Структура проєкту

Крім того, проєкт має розділений користувацький інтерфейс, який реалізований за допомогою класу SDL_Handler, що займається відображенням графіки та взаємодіє з користувачем. Завдяки цій структурі, код логіки гри і графічного інтерфейсу зберігається в окремих компонентах, що полегшує розробку, тестування та підтримку програми.

Такий підхід до архітектури та структури проєкту дозволяє зберігати код чистим, добре організованим та легко зрозумілим. Така структура також дозволяє розширювати функціональність гри та вдосконалювати шаховий інтелект без великих змін у коді.

3.1.1 Реалізація графічного інтерфейсу за допомогою SDL

Бібліотека SDL (Simple DirectMedia Layer) є потужним інструментом для роботи з графікою, звуком, інтерфейсом та взаємодією з користувачем у програмах. У даному проєкті графічний інтерфейс реалізовано через власний клас SDL_Handler, який відповідає за керування графічними операціями.

SDL_Handler є враппером навколо функцій бібліотеки SDL та забезпечує зручний інтерфейс для малювання, обробки подій та взаємодії з користувачем. Завдяки цьому класу, робота з графікою стає більш простою та зрозумілою.

Для реалізації графічного інтерфейсу створено MVC патерн. Цей шаблон ділить систему на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Таким чином відокремлюєються дані (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. [7]

Контролер, яким в цьому проєкті є клас Game, відстежує взаємодію користувача із інтерфейсом, що виникають в процесі гри. Контролер дозволяє структурувати код в одному циклі. А виглядом в проєкті є клас SDL Handler. Таким чином, Модель в циклі після отримання руху гравця оновлює інтерфейс, звертаючись до SDL.

Мета такого підходу – гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того

використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

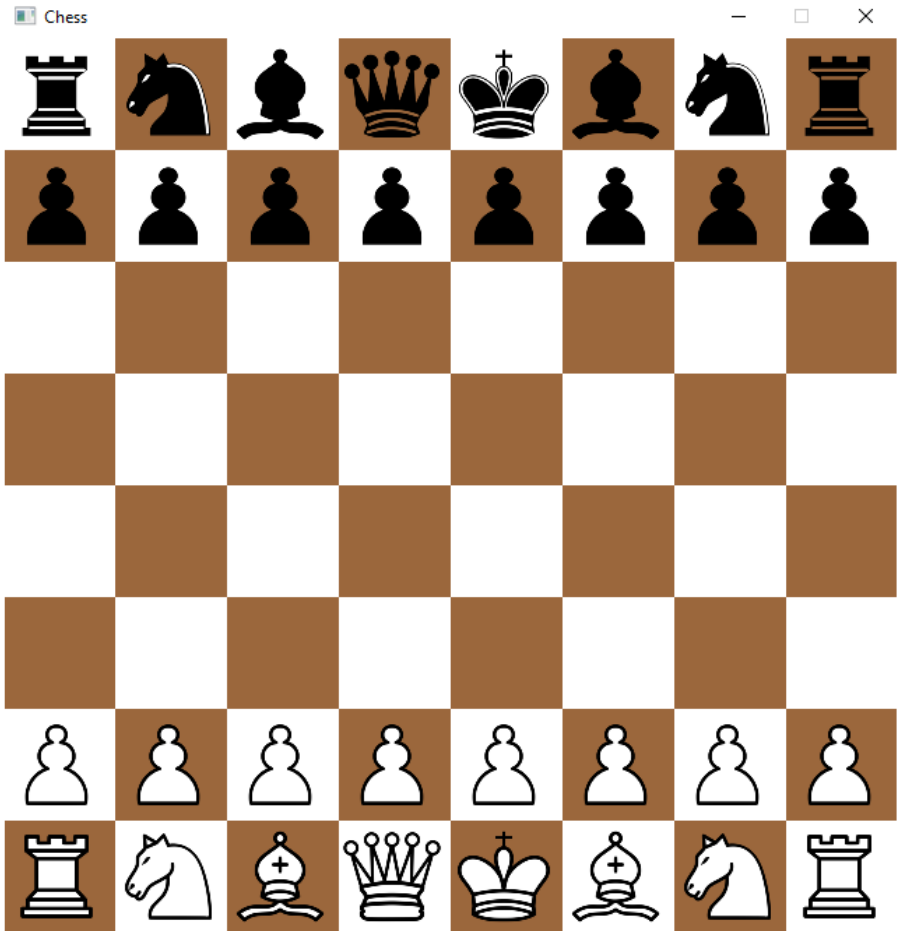

Рисунок 3.2 – Графічний інтерфейс програми

Одним із головних елементів графічного інтерфейсу є функція SDL_WaitEvent(&handler.m_event), яка відслідковує та обробляє події, такі як кліки миші або натискання клавіші. За допомогою цієї функції, програма взаємодіє з користувачем, отримуючи вхідні дії та реагуючи на них.

Для відображення шахової дошки та фігур на екрані використовується рендеринг – процес перетворення графічних об'єктів у пікселі, які можуть бути відображені на екрані. Під час рендерингу, клас SDL_Handler малює шахову дошку, фігури та інші елементи гри на екрані.

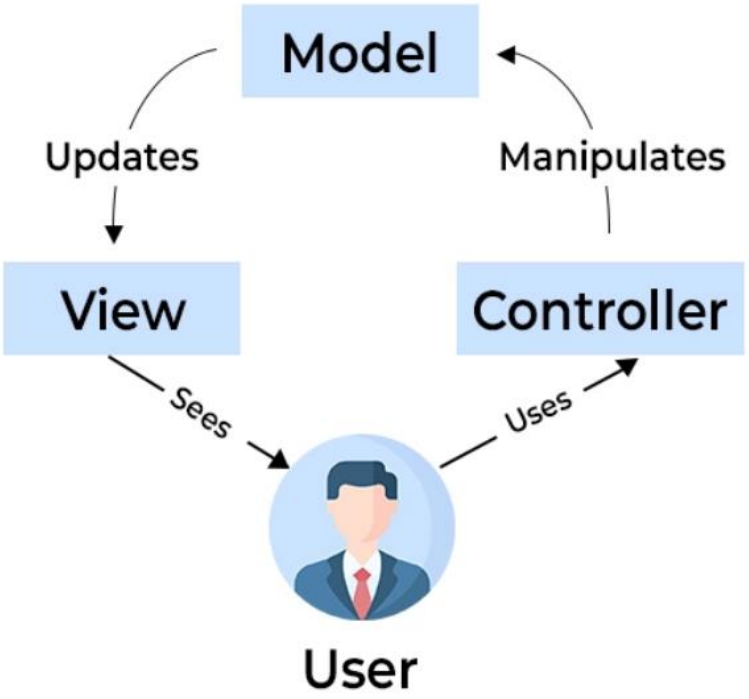

Рисунок 3.3 – Модель-вид-контролер патерн [9]

Кожна фігура має своє графічне представлення, яке відповідає за відображення фігури на дошці. Під час рендерингу, клас SDL_Handler зчитує дані про стан дошки та фігур з головного класу MainLoop і малює відповідні графічні елементи на екрані.

Загальна структура рендерингу виглядає наступним чином:

- ініціалізація графічного інтерфейсу та створення вікна;
- ініціалізація текстур та інших елементів для рендерингу;
- очікування вхідних подій від користувача за допомогою SDL_WaitEvent(&handler.m_event);
- обробка вхідних подій та оновлення стану гри;
- рендеринг шахової дошки та фігур на екрані;
- повторення кроків 3-5 до закриття програми або завершення гри.

Таким чином, за допомогою патерну MVC, клас SDL_Handler, який являє собою вид, забезпечує зручний інтерфейс для роботи з графічним інтерфейсом та

взаємодіє з бібліотекою SDL для створення візуально привабливого та функціонального шахового додатку. А через модель є можливість звертатись до цього класу, та завжди оновлювати стан графічного інтерфейсу.

3.1.2 Обробка та відображення користувачем різних шахових ситуацій

Обробка та відображення користувачем різних шахових ситуацій є важливим етапом в реалізації шахової гри. Додаток повинен правильно розпізнавати та відображати різні шахові ситуації, такі як шах королю, можливі ходи фігур, особливі правила, такі як EnPassant або рокіровка короля.

Шах королю: Додаток відслідковує можливі ходи всіх фігур та перевіряє, чи знаходиться король у зоні атаки іншої фігури. Якщо король перебуває під загрозою, гравець буде попереджений про шах і має можливість зробити відповідний хід для уникнення мату.

Можливі ходи фігур: Додаток коректно обробляє можливі ходи для кожної фігури відповідно до правил шахів. Наприклад, пішак може рухатись лише вперед, атакувати по діагоналі для здійснення взяття фігури супротивника. Король може рухатись на одне поле в будь-якому напрямку.

EnPassant: Це спеціальне правило, коли пішак може здійснити взяття іншого пішака, який зробив двохкроковий хід на одному з певних рядків до нього. Додаток повинен відслідковувати цю можливість та коректно виконувати це правило при необхідності.

Рокірування короля: Це спеціальний хід, коли король здійснює двохкроковий рух у бік своїх ладей, а лад'ї пересуваються навколо короля. Цей хід можна здійснити тільки за певних умов: король та лад'ї не повинні мати рухів до цього моменту, та поле між королем та лад'ями повинно бути вільним.

Таким чином, коли гравець натискає на клітину поля, отримуємо фігуру, яка стоїть на цій клітині:

```
xStart = handler.m_event.button.x / handler.CELL_WIDTH;
yStart = handler.m_event.button.y / handler.CELL_WIDTH;
clickedOn = game->GetFieldPos(xStart, yStart);
```

Після цього звертаємося до класу game, щоб отримати усі можливі ходи:

```
game->renderPossibleMoves(clickedOn);
```

А в цій функції викликаються функцію фігури, яка рахує і видає усі можливі ходи гравця:

```
piece->calcPossibleMoves(m_field, true);
std::vector<PossibleMove> possible = piece->getPossibleMoves();
```

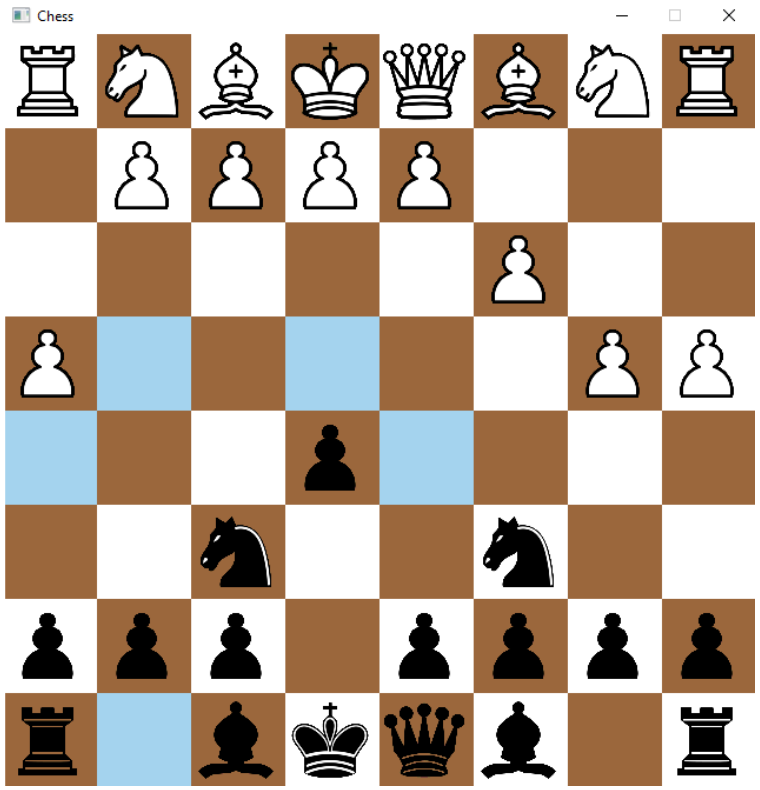

Рисунок 3.4 – Можливі ходи коня

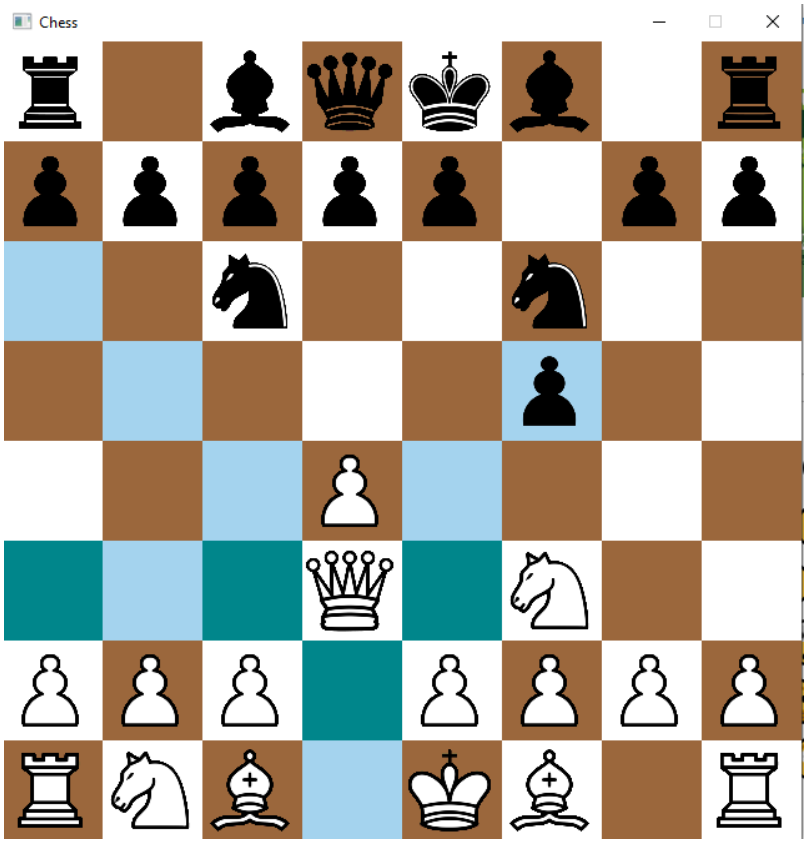

Рисунок 3.5 – Можливі ходи ферзя

При реалізації обробки та відображення різних шахових ситуацій, програма взаємодіє з класами фігур, реалізує їх правила руху та взаємодію, а також враховувати спеціальні правила шахів, що забезпечують правильну та логічну гру. Потім, як можна побачити на рисунку 3.4, SDL відображує усі можливі ходи фігури, на яку натиснув користувач. Це дозволяє наглядно побачити, які ходи гравець може здійснити.

3.2 Реалізація штучного інтелекту

3.2.1 Опис реалізації алгоритму Minimax та його оптимізацій

Алгоритм Minimax є основою для реалізації штучного інтелекту в шаховій грі. Його ідея полягає у пошуку оптимального ходу для гравця шахів, спираючись

на глибинний рекурсивний аналіз можливих ходів та оцінку позиції досягнутої після кожного ходу.

Основна ідея Minimax полягає в тому, що максимізуючий гравець намагається максимізувати свій виграш, тобто знаходить хід, який принесе йому найбільший виграш. З іншого боку, мінімізуючий гравець намагається мінімізувати збитки максимізуючого гравця, тобто знаходить хід, який призведе до найменших виграшів для опонента.

У розробленому програмному комплексі алгоритм Minimax був реалізований у рекурсивному вигляді. Головною структурою в цьому проєкті є структура Node, яка включає в себе наступні поля:

char board[12][12] – Двовимірний масив з поточним станом гри, де ‘ ‘ – це пусте поле, ‘x’ – границі поля, ‘перша літера фігури з маленької літери‘ – чорна фігура, і з великої літери – біла фігура.

map<char, int> materials; – Структура, яка зберігає інформацію про цінність фігур.
bool cur_side; – білі фігури = 0 чорні = 1
vector<Node*> next; – Посилання на усі наступні ходи, які можуть бути здійсненні з даної позиції

Головна функція виглядає так:

double minimax_alpha_beta(Node& root, Node& best, int depth, bool state, double alpha, double beta, bool IsAIPlayingWhite)

де Node – це структура з поточним станом гри, Depth – глибина алгоритму.

На кожному кроці ця функція аналізує всі можливі ходи, які може зробити гравець, та їх відповіді. Для оцінки кожного можливого ходу використовується

функція оцінки позиції гри static_eval, яка враховує стан дошки, розташування фігур та інші параметри. При static_eval оцінюється стан гри, по загальній кількості балів фігур, які є на полі. А також до цієї кількості додаються бали, які залежать від положення фігури на дошці, щоб позиція була найкраща для ШІ. Ось як виглядає ця таблиця для білих фігур (рис. 3.6)

```
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
[ -1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
[  2.0,  2.0,  0.0,  0.0,  0.0,  0.0,  2.0,  2.0 ],
[  2.0,  3.0,  1.0,  0.0,  0.0,  1.0,  3.0,  2.0 ]
```

```
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -0.5,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[  0.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[ -1.0,  0.5,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]
```

```
[  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
[  0.5,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[ -0.5,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -0.5],
[  0.0,   0.0, 0.0,  0.5,  0.5,  0.0,  0.0,  0.0]
```

```
[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  1.0,  1.0,  0.5,  0.0, -1.0],
[ -1.0,  0.5,  0.5,  1.0,  1.0,  0.5,  0.5, -1.0],
[ -1.0,  0.0,  1.0,  1.0,  1.0,  1.0,  0.0, -1.0],
[ -1.0,  1.0,  1.0,  1.0,  1.0,  1.0,  1.0, -1.0],
[ -1.0,  0.5,  0.0,  0.0,  0.0,  0.0,  0.5, -1.0],
[ -2.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -2.0]
```

```
[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0],
[-4.0, -2.0,  0.0,  0.0,  0.0,  0.0, -2.0, -4.0],
[-3.0,  0.0,  1.0,  1.5,  1.5,  1.0,  0.0, -3.0],
[-3.0,  0.5,  1.5,  2.0,  2.0,  1.5,  0.5, -3.0],
[-3.0,  0.0,  1.5,  2.0,  2.0,  1.5,  0.0, -3.0],
[-3.0,  0.5,  1.0,  1.5,  1.5,  1.0,  0.5, -3.0],
[-4.0, -2.0,  0.0,  0.5,  0.5,  0.0, -2.0, -4.0],
[-5.0, -4.0, -3.0, -3.0, -3.0, -3.0, -4.0, -5.0]
```

```
[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0],
[5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0,  5.0],
[1.0,  1.0,  2.0,  3.0,  3.0,  2.0,  1.0,  1.0],
[0.5,  0.5,  1.0,  2.5,  2.5,  1.0,  0.5,  0.5],
[0.0,  0.0,  0.0,  2.0,  2.0,  0.0,  0.0,  0.0],
[0.5, -0.5, -1.0,  0.0,  0.0, -1.0, -0.5,  0.5],
[0.5,  1.0, 1.0,  -2.0, -2.0,  1.0,  1.0,  0.5],
[0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0]
```

Рисунок 3.6 – Таблиця балів для обчислення стану гри [2]

Відповідно до цієї оцінки, алгоритм визначає оптимальний хід для штучного інтелекту.

Оптимізація Alpha-Beta Pruning дозволяє виключити непотрібні розглядання гілок дерева ходів, що значно зменшує кількість обчислень та покращує продуктивність алгоритму. Це досягається за рахунок додаткових параметрів Alpha і Beta, які обмежують діапазон оцінок. Якщо на якому-небудь рівні рекурсії виявляється, що можливий хід максимізуючого гравця не може перевищити вже знайдений найкращий виграш, ця гілка рекурсії відкидається (проріджується) і не розглядається далі.

В даному проєкту на кожному кроці рахується значення minimax в наступній глибині, після поточної. Мінімальне і максимальне значення встановлюється в межі α і β. Після цього перевіряється, якщо alpha >= beta, тоді заходимо у наступну глибину циклу, оскільки в цьому не буде сенсу.

```
if (alpha >= beta)
        break;
```

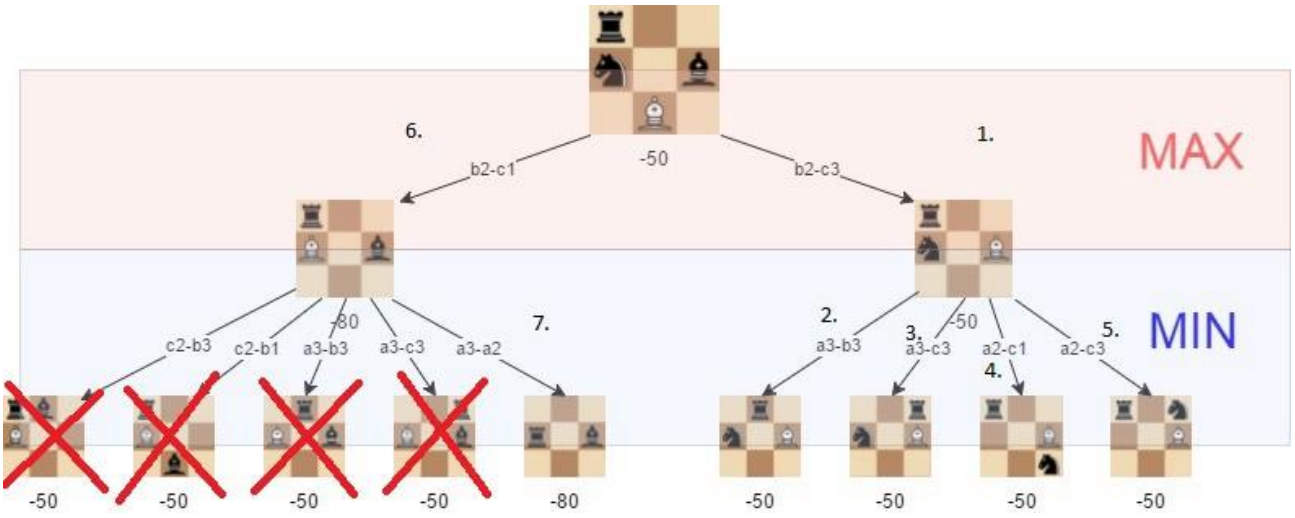

Рисунок 3.7 – Альфа-бета відсічення

3.2.2 Взаємодія користувача із штучним інтелектом

В рамках цього проєкту, оскільки штучний інтелект був реалізований окремим класом, та мав свої функції, і мав передавати інформацію до загального стану гри, потрібно було спланувати те, як користувач і штучний інтелект будуть взаємодіяти.

Вся взаємодія відбувається в класі MainLoop(). Під час, коли очікується якийсь event в циклі, програма перевіряє, чий зараз хід:

```
if( game->getTurn() == playerTeam)
{
// ход ігрока
}
else
{
// Ход штучного інтелекту
}
```

Під час ходу гравця, програма очікує на введення користувачем ходу на ігровій дошці. Після завершення ходу користувача, введений хід передається до спеціального класу ChessAI, де він аналізується та використовується для оцінки поточної гри та прийняття рішення штучним інтелектом:

```
ChessAI->GetUserInput(UserMoveForAI);
```

У випадку ходу штучного інтелекту, програма очікує завершення обчислень інтелектуальної частини. Після отримання оптимального ходу від

штучного інтелекту, він вставляється в глобальний менеджер гри, що призводить до оновлення ігрової дошки та відображення змін на екрані.

```
game->InsertAIMove(chessAIMoveStr);
```

Такий підхід створює плавну, динамічну та інтуїтивно зрозумілу взаємодію між гравцем та штучним інтелектом під час гри в шахи. Кожен хід, зроблений гравцем або штучним інтелектом, успішно передається та обробляється відповідними компонентами програми.

4 РОБОТА КОРИСТУВАЧА З ПРОГРАММОЮ

4.1 Системні вимоги та інсталяція

Для успішної інсталяції та запуску розробленого шахового додатка необхідно дотриматися наступних системних вимог та інструкцій:

- наявність встановленого C++ компілятора, наприклад, Visual Studio (версії 2017 або вище) з підтримкою C++ 17;
- SDL бібліотека. Її можна скачати з офіційного сайту: https://www.libsdl.org/;
- SDL Image: https://wiki.libsdl.org/SDL2_image/FrontPage (рис. 4.1).

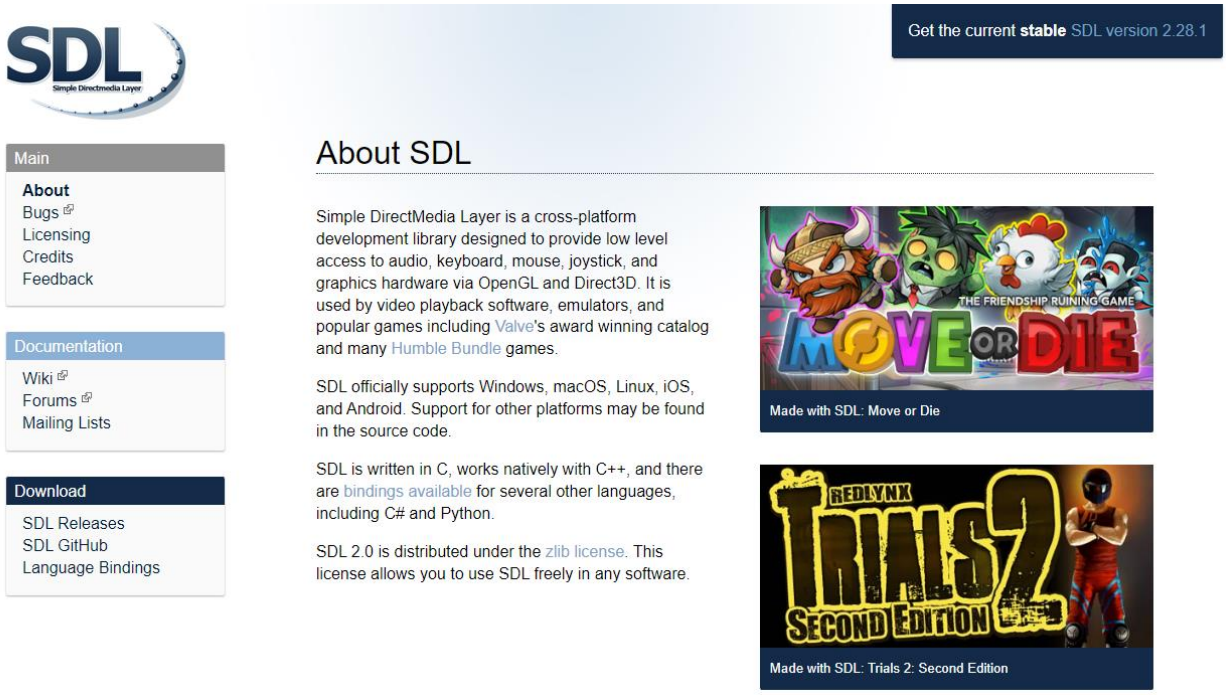

Рисунок 4.1 – Вид головного сайту SDL [7]

4.2 Опис інтерфейсу та можливостей програми для користувача

Запуск програмного додатку виконується запуском виконуючого файлу, який знаходиться в системному каталозі, у якому зберігаються усі модулі програмного комплексу, або після компіляції проєкту. При запуску відображається консольне вікно програми (рис. 4.2).

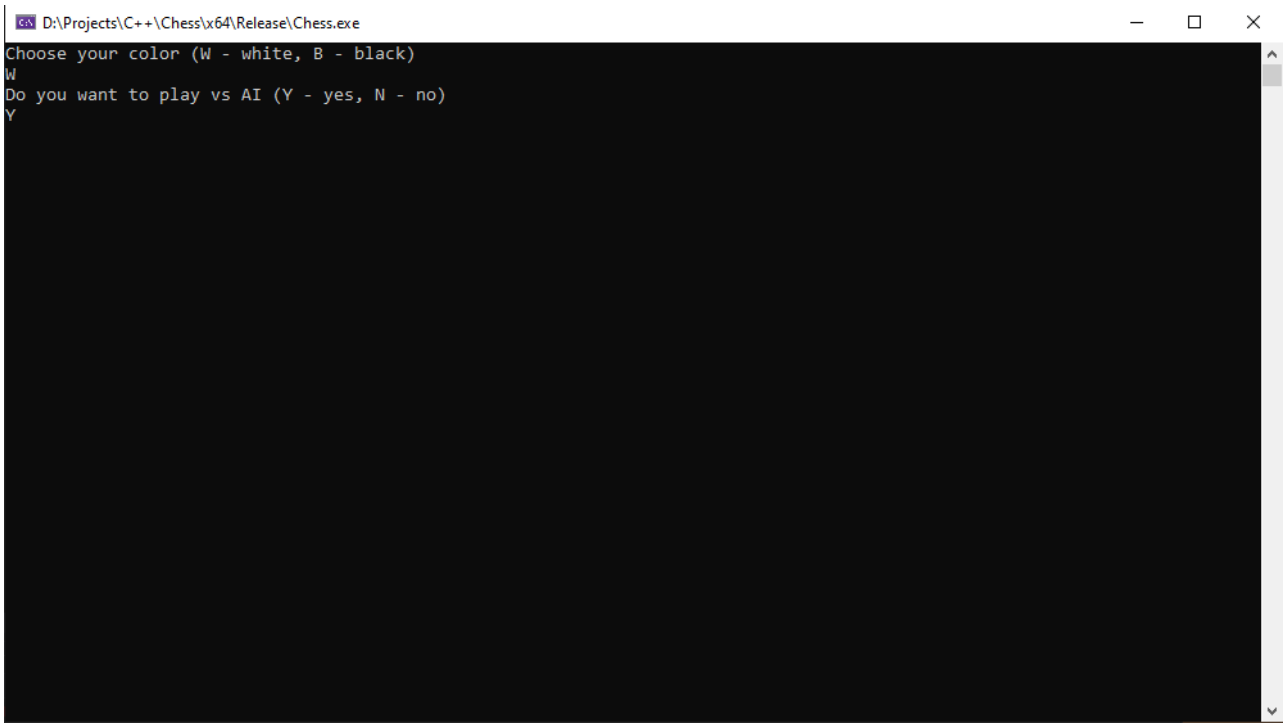

Рисунок 4.2 – Головне вікно програми

В цьому вікні користувачу потрібно вибрати сторону. Треба написати ‘W’, якщо користувач хоче грати за білі фігури, та ‘B’ – за чорні. І потім вибрати, чи хоче він грати проти штучного інтелекту – ‘Y’, якщо хоче, та ‘N’, якщо ні. Після цього відкриється головна програма (рис. 4.3), якщо користувач вибрав білі фігури, і програма (рис. 4.4) – якщо чорні.

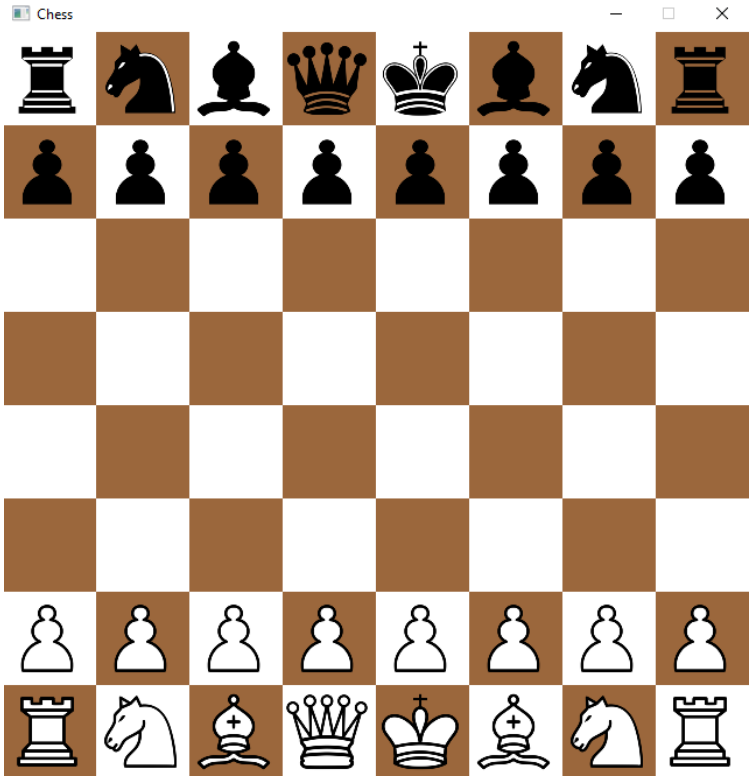

Рисунок 4.3 – Інтерфейс, якщо користувач вибрав білі фігури

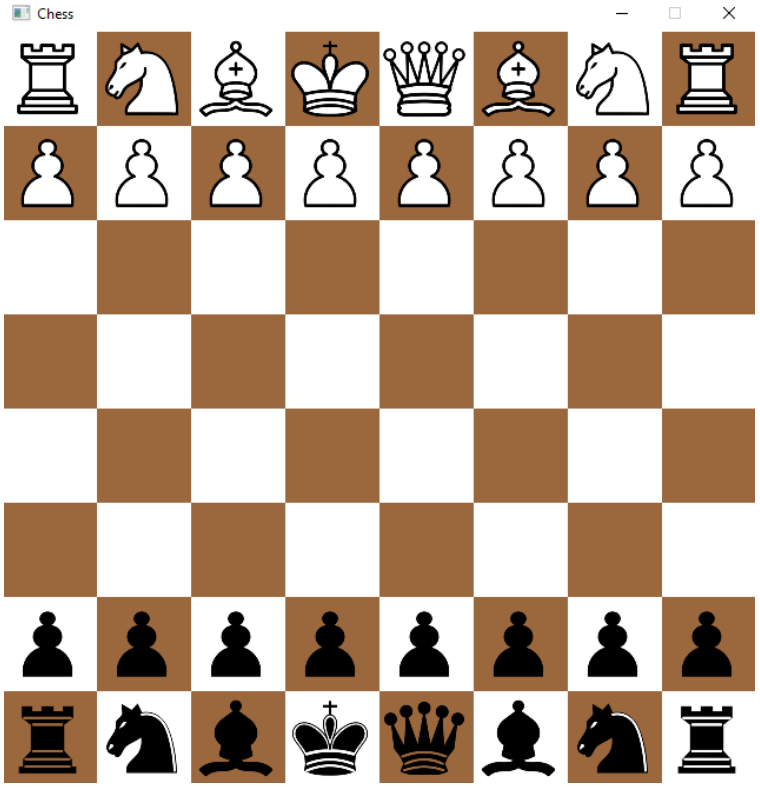

Рисунок 4.4 – Інтерфейс, якщо користувач вибрав чорні фігури

Наприкінці гри, коли одна зі сторін перемогла, – про це висвітиться повідомлення в консолі (рис. 4.5)

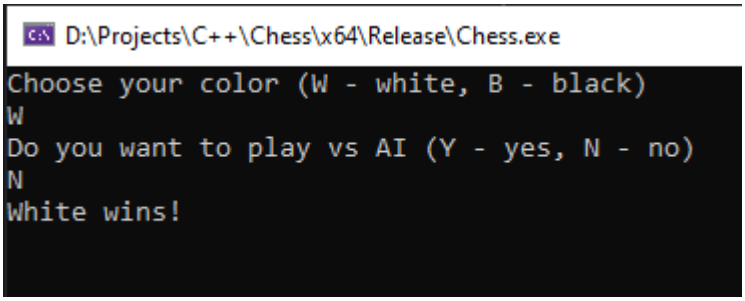

Рисунок 4.5 – Повідомлення про перемогу білих фігур

ВИСНОВКИ

У даній кваліфікаційній роботі була проведена розробка та реалізація шахового додатка з штучним інтелектом, який є гідним суперником для користувача. Описуючи кожен крок від постановки задачі та вибору програмного забезпечення і інструментів розробки до реалізації конкретних аспектів, проведено глибокий аналіз особливостей розробки шахової гри, вибір архітектури програмного комплексу та деталі вибору мови програмування та усіх необхідних бібліотек , особливості графічного інтерфейсу та взаємодії з користувачем, а також алгоритми штучного інтелекту та його оптимізації.

Були вивчені та застосовані алгоритми штучного інтелекту, зокрема алгоритм Minimax з оптимізацією Alpha-Beta Pruning, що дозволило ефективно оцінювати можливі ходи та вибирати найкращий варіант для гри, кожен раз оцінюючи стан гри та вибираючи найкращі ходи на кожному кроці.

Розроблений додаток не лише надає можливість користувачам грати у шахи, а й розширює їх досвід завдяки штучному інтелекту. В проєкті використовуються багато алгоритмів, які необхідні для коректної роботи програми, такі як прорахунок усіх можливих ходів кожної фігури, перевірка на шах і мат для короля, алгоритми для відображення всього на екрані і взаємодії користувача із додатком, штучний інтелект для шахів із використанням алгоритму MiniMax.

Під час розробки було праналізовано багато можливих бібліотек та мов програмування, але для написання швидко працюючого алгоритму була вибрана саме мова C++ та була використана бібліотека SDL для графічного інтерфейсу, що додало важливу візуальну складову до гри. Були розглянуті особливості побудови UI для додатку, та модель MVC для зручної взаємодії графічного інтерфейсу та системи. Системні вимоги та інструкції з інсталяції були ретельно

розглянуті, забезпечуючи зручний доступ користувачів до гри та проєкту.

Отже, дана робота представляє собою важливий етап у розробці та впровадженні ігрових додатків із штучним інтелектом. Технічні аспекти, реалізація математичних алгоритмів та використання об'єктно-орієнтованого програмування узгоджуються для створення захоплюючого ігрового середовища, яке сприяє підвищенню рівня взаємодії користувача з грою та розвитку його аналітичних навичок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бондаренко О. О., Ластовецький В. В., Пилипчук О. П., Шестопалов Є. А. Інформатика : підруч. для 7 кл. закл. загал. серед. Освіти. Харків : Вид-во "Ранок", 2020. 160 с.

2. Eade J.V. Chess For Dummies: tutorial. Hoboken, New Jersey: Publisher "For Dummies", 2011. 384 с. URL : https://www.amazon.com/Chess-Dummies-James-Eade/dp/1118016955 (дата звернення : 25.06.2023).

3. Straustrup B. A history of C++: 1979–1991. Murray Hill, New Jersey: Publisher "AT&T Bell Laboratories", 1993. 35 с.

4. Straustrup B. The C++ Programming Language (Third Edition and Special Edition) Murray Hill, New Jersey: Publisher "Addison-Wesley Professional", 1996. 1030 с.

5. PHP об'єктно-орієнтованого програмування. URL : https://www.w3big.com/ru/php/php-oop.html#gsc.tab=0 (дата звернення : 12.05.2023).

6. Microsoft Visual Studio. Офіційна документація. URL : https://visualstudio.microsoft.com/ (дата звернення : 17.05.2023).

7. SDL – кросплатформна мультимедійна бібліотека. URL : https://www.libsdl.org/ (дата звернення : 01.06.2023).

8. Russell S. J., Norvig P . Artificial Intelligence: A Modern Approach 3rd Edition. London: Publisher "Pearson", 2020. 1136 с.

9. Шаблони проектування MVC, MVP та MVVM. URL : https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad (дата звернення : 08.06.2023).

10. Cormen T. H., Leiserson C. E., Rivest R. L., Clifford S. Introduction to Algorithms, 3rd Edition (Mit Press) Cambridge, Massachusetts: Publisher "MIT

Press", 2009. 1292 с.

11. Maschler M. B., Solan E., Shmuel Z. Game Theory. Cambridge, Massachusetts: Publisher "Cambridge University Press", 2013. 1003 с.

ДОДАТОК А

Програмний додаток для гри в шахи. Лістинг програми

MainLoop:

```
#include "MainLoop.h"

#include "Misc.h"
#include "Game.h"

#include <iostream>

#include <SDL.h>
#include "SDL_Handler.h"

#include <stdio.h>
#include <memory>

#include "mainAI.h"
#include "Piece.h"

void MainLoop::Run(char playerSideChar, bool isPlayingVsBot)
{
  SDL_Handler handler;
  handler.renderBackground();

  Piece::Team playerTeam;
  if (playerSideChar == 'B' || playerSideChar == 'b') playerTeam = Piece::Team::BLACK;
  else playerTeam = Piece::Team::WHITE;

  std::unique_ptr<Game> game = std::make_unique<Game>(&handler, playerTeam);
  std::unique_ptr<mainAI> ChessAI = std::make_unique<mainAI>(playerTeam);

  bool quit = false;

  int xStart = -1;
  int yStart = -1;
  int xEnd = -1;
  int yEnd = -1;
  Piece* clickedOn = nullptr;
```

```
  while (!quit)
  {
        while (SDL_WaitEvent(&handler.m_event))
        {
              if (handler.m_event.type == SDL_QUIT)
              {
                    quit = true;
                    break;
              }
              if (!isPlayingVsBot || game->getTurn() == playerTeam) //We give player ability to
move
              {
                    if (handler.m_event.type == SDL_MOUSEBUTTONDOWN)
                    {
                          xStart = handler.m_event.button.x / handler.CELL_WIDTH;
                          yStart = handler.m_event.button.y / handler.CELL_WIDTH;
                          clickedOn = game->GetFieldPos(xStart, yStart);
                          if (clickedOn != nullptr)
                          {
                                if (clickedOn->getTeam() == game->getTurn())
                                {
                                      game->renderPossibleMoves(clickedOn);
                                }
                          }
                    }
                    else if (handler.m_event.type == SDL_MOUSEBUTTONUP)
                    {
                          if (clickedOn != nullptr)
                          {
                                if (clickedOn->getTeam() == game->getTurn())
                                {
                                      game->UndoRenderPossibleMove(clickedOn);
                                }
                          }
                          xEnd = handler.m_event.button.x / 80;
                          yEnd = handler.m_event.button.y / 80;
                          if (clickedOn != nullptr)
                          {
                                if ((xStart != -1 && yStart != -1 && xEnd != -1 && yEnd != -
1)
                                      && (clickedOn->getTeam() == game->getTurn())
                                      && (game->isValidMove(xEnd, yEnd, clickedOn)))
                                {
                                std::vector<PossibleMove> list = game->GetFieldPos(xStart, yStart)
>getPossibleMoves();
                                      for (const auto& value : list)
                                      {
                                        if (value.MovePos.xCoord == xEnd &&
value.MovePos.yCoord == yEnd)
```

```
                                                {
                                                    std::pair<std::string, MoveType>
UserMoveForAI = game->move(clickedOn, PossibleMove{ {xEnd, yEnd}, value.Move_Type }); //
Here we should return a string of a move
                                                    //Here we should give player's move to
AI
                                                    ChessAI-
>GetUserInput(UserMoveForAI);
                                                }
                                            }
                                            xStart = -1;
                                            yStart = -1;
                                            yEnd = -1;
                                            game->calcAllMoves();
                                            clickedOn = nullptr;
                                        }
                                    }
                                }
                            }
                            else if(!ChessAI->isBotThinking())
                            {
                             std::string chessAIMoveStr = ChessAI->CalculateAIMove();
                             game->InsertAIMove(chessAIMoveStr);
                             game->calcAllMoves();
                             clickedOn = nullptr;

                             xStart = -1;
                             yStart = -1;
                             yEnd = -1;
                            }
                    }
}
}
```

Функція minimax_aplha_beta:

```
double minimax_alpha_beta(Node& root, Node& best, int depth, bool state, double alpha, double
beta, bool IsAIPlayingWhite) {

   // cout << "==========" << " DEPTH LEVEL " << depth << " ==========" << endl;

   if (depth >= MAX_DEPTH || 0 == root->next.size()) {
      static_evals++;
      return static_eval(root, IsAIPlayingWhite);
   }

   // MAXIMIZING
   else if (state) {
```

```
        vector<Node>::iterator it;
        for (it = root->next.begin(); it != root->next.end(); it++) {

            double val = minimax_alpha_beta(*it, best, depth + 1, 0, alpha, beta, IsAIPlayingWhite);
            if (val > alpha) {
                if (depth == 0)
                    best = *it;
                alpha = val;
            }
            if (alpha >= beta)
                break;
        }
        return alpha;
    }

    // MINIMIZING
    else {

        vector<Node>::iterator it;
        for (it = root->next.begin(); it != root->next.end(); it++) {

            double val = minimax_alpha_beta(*it, best, depth + 1, 1, alpha, beta, IsAIPlayingWhite);
            if (val < beta) {
                if (0 == depth)
                    best = *it;
                beta = val;
            }
            if (alpha >= beta)
                break;
        }
        return beta;
    }
}
```

TreeAI:

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include "mainAI.h"

using namespace std;

bool same_side(bool side, char target) {

    if (0 == side && ('Q' == target || 'R' == target || 'B' == target || 'N' == target || 'P' == target || 'K' ==
target))
```

```
        return true;
    if (1 == side && ('q' == target || 'r' == target || 'b' == target || 'n' == target || 'p' == target || 'k' ==
target))
        return true;
    return false;
}

bool is_repeat(Node& test) {

    string board_hash = "";
    char c;
    int i, j;

    for (i = 2; i < 10; i++) {
        for (j = 2; j < 10; j++) {
            c = test->board[i][j];
            if (' ' != c)
                board_hash += c;
            else
                board_hash += '0';
        }
    }

    if (0 == board_hash.size() || find(prev_moves.begin(), prev_moves.end(), board_hash) ==
prev_moves.end()) {
        prev_moves.push_back(board_hash);
        // cout << "NEW HASH :: " << board_hash << endl;
        return false;
    }
    // cout << "OLD HASH :: " << board_hash << endl;
    return true;
}


void tree_insert(Node& root, int x, int y, int p, int q) {

    Node next = new tNode;

    // Copying the current board state to the new board state
    int a, b;
    for (a = 0; a < 12; a++) {
        for (b = 0; b < 12; b++) {
            next->board[a][b] = root->board[a][b];
        }
    }

    // Moving the piece from (x,y) to (p,q), replacing if needed
    char source = next->board[x][y];
    char target = next->board[p][q];
```

```
    next->board[p][q] = source;
    next->board[x][y] = ' ';

    // Checks if this board state has already been examined; skips it if it has been examined
    if (!is_repeat(next)) {

        next->materials = root->materials;
        if (' ' != target)
            next->materials[target] -= 1;

        // Copying attributes from old node to new node
        next->w_king_pos = root->w_king_pos;
        next->b_king_pos = root->b_king_pos;
        next->cur_side = !(root->cur_side);

        string conv_array[12] = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11" };
        next->mv = "(" + conv_array[x] + "," + conv_array[y] + ")" + "->" + "(" + conv_array[p] + ","
+ conv_array[q] + ")";

        root->next.push_back(next);
    }
    else {
        delete next;
    }
}

void tree_delete(Node& root) {
    if (root == NULL)
        return;

    for (int i = root->next.size() - 1; i >= 0; i--)
    {
        tree_delete(root->next[i]);
        Node old = root->next.back();
        root->next.pop_back();
        delete old;
    }
}
```