МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система-тренажер для імітації гри в шахи»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Берест О.Б.**

Студента групи ІН – 71 **Казначеєв В.І.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-71 спеціальності "Комп'ютерних наук" денної форми навчання Казначеєва Володимира Ігоровича.

Тема: "**Інформаційна система-тренажер для імітації гри в шахи** "

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) Огляд існуючих рішень; 2) аналіз методолгій програмування; 3) аналіз мови програмування; 4) аналіз необхідних бібліотек; 5) аналіз алгоритмів для створення шахового рушія; 6) практична частина.

Дата видачі завдання "_____"_____ 2021 р.

Керівник випускної роботи _____ Берест О.Б.

Завдання приняв до виконання _____ Казначеєв В.І.

РЕФЕРАТ

Записка: 63 стор., 23 рис.,1 додаток, 18 джерел.

Об’єкт дослідження — Алгоритми для створення шахового рушію.

Мета роботи — Створення інформаційної системи-тренажеру для імітації гри в шахи.

Методи дослідження — емпірично-науковий метод.

Результати — був проведений аналіз алгоритмів для створення шахового рушія. Також було проаналізовано та вибрана мова програмування разом з методологією програмування. В результаті було розроблено додаток який дає можливість грати проти шахового рушія.

ІНФОРМАЦІЙНА СИСТЕМА-ТРЕНАЖЕР, НЕЙРОННІ МЕРЕЖІ,
АЛЬФА-БЕТА ВІДСІКАННЯ, МІНІМАКС, ШАХИ, JAVA,
LOMBOK, ООП, SOLID.

ЗМІСТ

● ВСТУП

Серед усіх видів ігор які колись були придумані людством є одна яка близька як до спорту, так і до науки разом з мистецтвом. Ця гра називається шахами.

Шахи є організованим видом спорту з ієрархією звань, своєю системою яка регулює турніри, декількома лігами на різних рівнях (національні, міжнародні), конгресами, тощо.

На сьогоднішній день неможливо недооцінити популярність цієї гри, вона стає все більш популярною з кожним днем. Все більше людей зацікавилися в тому щоб грати в цю гру як на професійному рівні, так і на аматорському.

В свою чергу збільшення зацікавленості в цій грі призвело до необхідності мати можливість навчатися грати в шахи швидко та ефективно. Все більша кількість людей купує спеціалізовані курсе, або навіть платить людям задля того щоб вони навчили їх грати якомога швидше.

Багато хто з гравців воліє грати в спеціальних шахових клубах, в яких можуть виступати гросмейстери та тренери, проводяться лекції які допомагають покращити себе як гравця. Проте найголовнішим в шахах завжди була практика, тому навіть найбільш завзятим теоретикам необхідно була знайти супротивника свого рівня з яким вони могли б тренуватися.

Проте наразі люди зіткнулися з такою проблемою як пандемія, яка не дає можливості цим людям грати разом з іншими людьми. Саме через це популярність набирають веб – додатки з можливістю грати в шахи проти інших людей онлайн, або навіть програми які дозволяють грати проти штучного інтелекту.

Можливість грати проти комп’ютера дає нагоду спробувати себе проти супротивника складність якого при необхідності можна збільшити або зменшити. В свою чергу це дає можливість гравцю роздивитися якусь конкретну

позицію або підготувати стратегію на для конкретного випадку, що дозволяє гравцю розвиватися та ставати краще.

Наразі існує велика кількість програм які можуть дати людині нагоду грати проти штучного інтелекту, проте ці програми зазвичай несуть у собі мету лише дати користувачу мінімальний необхідний інтерфейс, що тягне за собою багато незручностей, адже і досі існують програми навіть без графічного інтерфейсу. Такий підхід не дає людині повністю зосередитися на навчанні цій безумовно тяжкій грі, та містить багату кількість недоліків.

Зазвичай у таких програмах використовуються шахові рушії який аналізує позицію та можливі ходи для того щоб повернути найкращі ходи. Такі алгоритми як правило є платними та їх не просто знайти. Окрім цього існують нейронні мережі які також виконують аналогічний функціонал, проте на даний момент є найбільш ефективними.

Тому в цій дипломній роботі буде розглянуто можливі варіанти для створення шахового рушія, проведено аналіз аналогів та вибрано конкретне рішення для того, щоб створити систему-тренажер для гравців у шахи яка буде містити вибраний підхід для шахового рушія в поєднанні з графічним інтерфейсом який буде давати можливість юзеру взаємодіяти з програмою.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд правил шахів

Перед тим як буде виконаний огляд існуючих рішень, необхідно ознайомитися з правилами шахів для того, щоб ми мали можливість оцінити якість виконання аналогів відносно реальної гри.

Шахи — це стратегічна гра яка поєднує в собі елементи уяви разом з наукою та спортом. Походить від індійської гри чатуранґа. Гра в шахи проходить на дошці яку називають шахівниця (дошка поділена на 64 темні або світлі клітини) на якій розташовані 16 світлих та 16 темних фігур(рисунок 1.1). Гра закінчується коли королю супротивника оголошений мат(ситуація в якій після нападу на короля супротивника своєю фігурою — шаху у іншого гравця не існує можливості походити так аби король не був під шахом) як на рисунку 1.2.

Рисунок 1.1 – Приклад шахової дошки

Рисунок 1.2 – Приклад мату

При початку партії в шахи на дошці розташовано 6 типів фігур для кожного гравця – один король та ферзь, дві тури, два слона та коня та вісім пішаків(рисунок 1.3 зліва на право відповідні типи фігур).

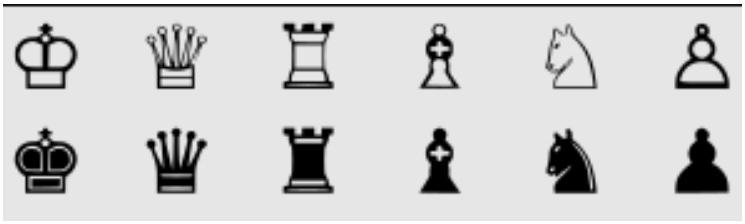

Рисунок 1.3 – Приклад шахових фігур

Кожна фігура має свої особливості та може пересуватися згідно за правилами, окрім цього фігури мають свою цінність яка залежить від її сили:

- Ферзь – сильна фігура в шахах, може ходити на будь яку клітинку по діагоналі, вертикалі або горизонталі відносно її положення. Зазвичай її оцінюють в 9 одиниць.
- Король- одна з найслабших фігур в шахах, проте найважливіша, метою гри є захист короля від мату. Ходить аналогічно до королеви, проте лише на одну клітину. Зазвичай оцінюється в 3.5 одиниць

відносно його сили, проте оскільки метою гри є захист цієї фігури, тому така оцінка не використовується при обчисленнях кроку.

- Тура – сильна фігура яка може походити на будь яку клітинки по вертикалі або горизонталі відносно її положення. Зазвичай оцінюється в 5 одиниць.
- Слон – слабка фігура яка може ходити на будь яку клітинки по діагоналі відносно її положення. Зазвичай оцінюється в 3 одиниці.
- Кінь – слабка фігура яка може ходити «літерою» відносно її положення. Зазвичай оцінюється в 3 одиниці.
- Пішак – найслабша фігура, може ходити на одну клітинку вперед, або но дві, якщо ця фігура ще не рухалася.

Окрім цього в шахах існує можливість взяття фігури супротивником, тобто якщо в місці куди походить фігура є фігура супротивника, то гравець ставить на її місце свою фігуру, а фігура супротивника знімається з дошки. Винятком з цих правил є пішаки, вони беруть фігури які знаходяться на одну клітинку вперед по діагоналі. Взяття фігур свого кольору неможливу.

Наостанок, необхідно враховувати те, що гравець має враховувати те, що після здійснення свого ходу його король не може бути під шахом, бо це хід буде визнаний нелегальним і відмінений, а при ще одному нелегальному ході гравець автоматично програє. Тобто, наприклад, якщо якась фігура захищає короля від шаху, то ти не маєш права ходити цією фігурою.

1.2 Аналіз аналогів

Після того як було проведено огляд правил гри в шахи, необхідно виконати аналіз аналогів, виявити їх переваги та недоліки та зробити висновки що необхідно реалізувати в своєму шаховому рушії.

Спочатку проведемо порівняння трьох відомих шахових рушіїв: Stockfish, AlphaZero та Leela Chess Zero. Потім проведемо аналіз деяких уже існуючих програм які використовую шахові рушії разом з графічним інтерфейсом такі як Scid та Arena.

1.2.1 Stockfish

Stockfish – безкоштовний та відкритий рушій який доступний для багатьох платформ. Він є найсильнішим серед рушіїв з відкритим програмним кодом. За станом на жовтень 2020 року Stockfish є рушієм з найвищим рейтингом відносно списку комп'ютерних шахових рейтингів. Він є єдиним комп'ютером з рейтингом більше ніж 3500, а саме 3558.

Програма походить від шахового рушія з відкритим кодом – Glaurung, яких булу створено та випущено Ромстадом в 2004 році. Названо рушій так було тому, що його було «Вироблено в Норвегії та приготовано в Італії», мається на увазі те, що розробник був італійцем, а розробник Glaurung – норвежцем. Вісімнадцятого червня 2008 року Марко Костальба (розробник Stockfish) відмовився від розробки проекту, та запропонував створити розвилку останньої версії його рушія для того, щоб продовжити роботу з нею. З того часом проектом керує група волонтерів які і надалі покращують та розробляють цей проект.

Особливостями даного рушія є те, що він може використовувати до 512 потоків , а його розмір таблиці транспозиції може бути до 128 гігабайтів. Сам рушій використовує відсічення альфа – бета разом з бітовими дошками. Stockfish порівнянні з іншими рушіями має більшу глибину пошуку. Окрім цього він підтримує

Розглянемо детальніше, що таке альфа – бета відсічення. Альфа-бета відсічення зазвичай використовується для оцінки задач в яких кожну конкретну позицію можливо показати у вигляді дерева варіантів. Цей алгоритм використовується для зменшення кількості вузлів мінімакса. В основі цього алгоритму покладена ідея того, що оцінка гілки дерева варіантів може бути

зупинена, якщо значення яке ми отримаємо після оцінки цієї гілки буде завжди гіршим ніж значення обчислене для попередньої гілки. Оскільки альфа-бета відсікання є оптимізацією, то воно не впливає на коректність праці алгоритму.

Мінімакс же в своє чергу це правило для того, щоб приймати рішення(в нашому випадку для теорії ігор) для зменшення можливих втрат які людина, яка буде приймати рішення може зіткнутися при найгіршому для особи сценарію. В основі лежить так звана теорема мінімаксу: «Для будь-якої гри між двома гравцями з нульовою сумою та скінченною кількістю стратегій можливо знайти таке значення X, що для будь-якої стратегії другого гравця, перший може собі гарантувати виграш X і навпаки, для будь-якої стратегії першого гравця, другий гравець може гарантувати собі виграш -X».

Плюсами цього рушія є:

- За ідеальних умов(комп’ютер який зможе рахувати усі можливі варіанту) цей рушій буде найсильнішим, адже зможе рахувати усі варіанти.
- На відміну від нейронних мереж алгоритм не потребує ігор для навчання і зможе видавати результати базуючись лише на потужності комп’ютера
- Є повністю безкоштовним та має відкритий код
    - Можливість налаштувати глибину обчислювання, що дозволяє налаштувати цей рушій для гри проти людей
    -

Недоліками:

- Алгоритм залежить від потужності комп’ютери, наразі немає комп’ютерів які б дозволяли повністю рахувати усі варіанти.
- Швидкість обчислення кроків

1.2.2 AlphaZero

AlphaZero – нейронна мережа яка розроблена британською компанією DeepMind використовуючи підхід AlphoGO Zero. За 24 години ця програма досягла такого рівня , що змогла перемогти чемпіона миру серед програм – Stockfish, проте рушій грав на гіршому обладнанні та без доступу до дебютних баз та ендшпільних таблиць.

AlphaZero використовує концепцію штучної нейронної мережи - обчислюваної системи яка використовує біологічні нейронні мережи з яких створений мозок тварин. Сутністю таких систем є навчання задачі при цьому розбирання задачі без програмування безпосередньо під сам задачу. Тобто якщо ціллю є навчитися грати в шахи, то програма буде «навчатися» гри аналізуючи гру інших людей, або проти самого себе.

Такі програми не будуть використовувати уже сформовані принципи гри в шахи, таких як відомі дебюти та інші. Натомість програма буде формувати свої принципи гри які сформуються після аналізу.

Ця обчислювальна система ґрунтується на вузлах які називаються штучні нейрони(за аналогією до нейронів у мозку тварин). Кожне з’єднання між нейронами передає сигнали одне іншому. Отримавший сигнал нейрон зможе опрацювати його та відправити сигнал про це іншим нейронам.

Головною метою нейронної мережи є вирішення задачи таким же способом, яким би це робив мозок людини.

Особливістю AlphaZero є те, що на відмінну від традиційних рушіїв цій програмі не потрібні дебютні та ендшпільні бази даних. Окрім цього навідміну від Stockfish та аналогічних рушіїв непотрібні складні алгоритми для обчислення оцінки. Оскільки ця мережа може грати проти самої себе, то вона змогла самостійно розробити для себе принципи гри.

Плюсами цієї мережи є:

- Швидкість обчислення кращих кроків
- Менша прив’язаність до потужності комп’ютера
- Працює без дебютних та ендшпільних баз

Недоліками:

- Необхідність навчати мережу за допомогою аналізу інших ігор або гри проти самого себе , або інших рушіїв
- На відміну ввід традиційних рушіїв при достатній потужності є можливість того, що алгоритм буде відкидати непотрібні на його думку варіанти
- Неможливо налаштувати цю мережу для гри з людьми, адже принципи які вона використовує відрізняються від тих, що використовують люди в своїх іграх

1.2.3 Leela Chess Zero

Leela Chess Zero – безкоштовний шаховий рушій, який оснований на нейронних мережах по аналогії з AlphaZero разом з системою розподілення обчислень. Аналогічно до AlphaZero в цього рушія закладені тільки правила гри і нічого окрім цього. Leela Chess Zero навчається за допомого системи розподілення обчислень, яка координується на його веб-сайті.

В 2015 році після створення AlphaGo компанією DeepMind та викладанням опису алгоритму бельгійським програмістом Жан-Карло Паскутто добавив цей алгоритм до свого рушія Leela та назвав його Leela Zero. Після цього 9 січня 2018 року розробник Stockfish об’явив що він почав працювати над проектом Leela Chess Zero алгоритм для якого був взятий з Leela Zero, а генерація ходів з Stockfish. Після цього білоруський розробник Александр Ляшук повністю

переписав код рушія, що значно збільшило його швидкість та потужність та прибрало запозичення з Leela Zero та Stockfish.

При навчанні використовується розподілені обчислення – спосіб вирішення складних задач за допомогою використання декількох комп’ютерів, зазвичай з’єднаних в одну паралельну обчислювану систему. Особливістю цієї систему на відміну від суперкомп’ютерів є те, що вона має можливість необмежено збільшувати продуктивність за рахунок масштабування.

Плюсами цього рушія є:

- Швидкість обчислення кращих кроків
- Працює без дебютних та ендшпільних баз
- Можливість пришвидшити швидкість навчання за допомогою збільшення кількості комп’ютерів які працюють над цим

Недоліками:

- Необхідність навчати мережу за допомогою аналізу інших ігор або гри проти самого себе , або інших рушіїв
- На відміну ввід традиційних рушіїв при достатній потужності є можливість того, що алгоритм буде відкидати непотрібні на його думку варіанти
- Неможливо налаштувати цю мережу для гри з людьми, адже принципи які вона використовує відрізняються від тих, що використовують люди в своїх іграх
- Для ефекктивного навчання краще використовувати декілька комп’ютерів які працюють за допомогою розподілених обчислень

1.2.4 Scid

Scid (Shane`s Chess Information Database) – це графічний інтерфейс розроблений під декілька платформ, за допомогою якого можливо розбирати свої партії, грати проти шахових рушіїв та багато іншого(рисунок 1.4).

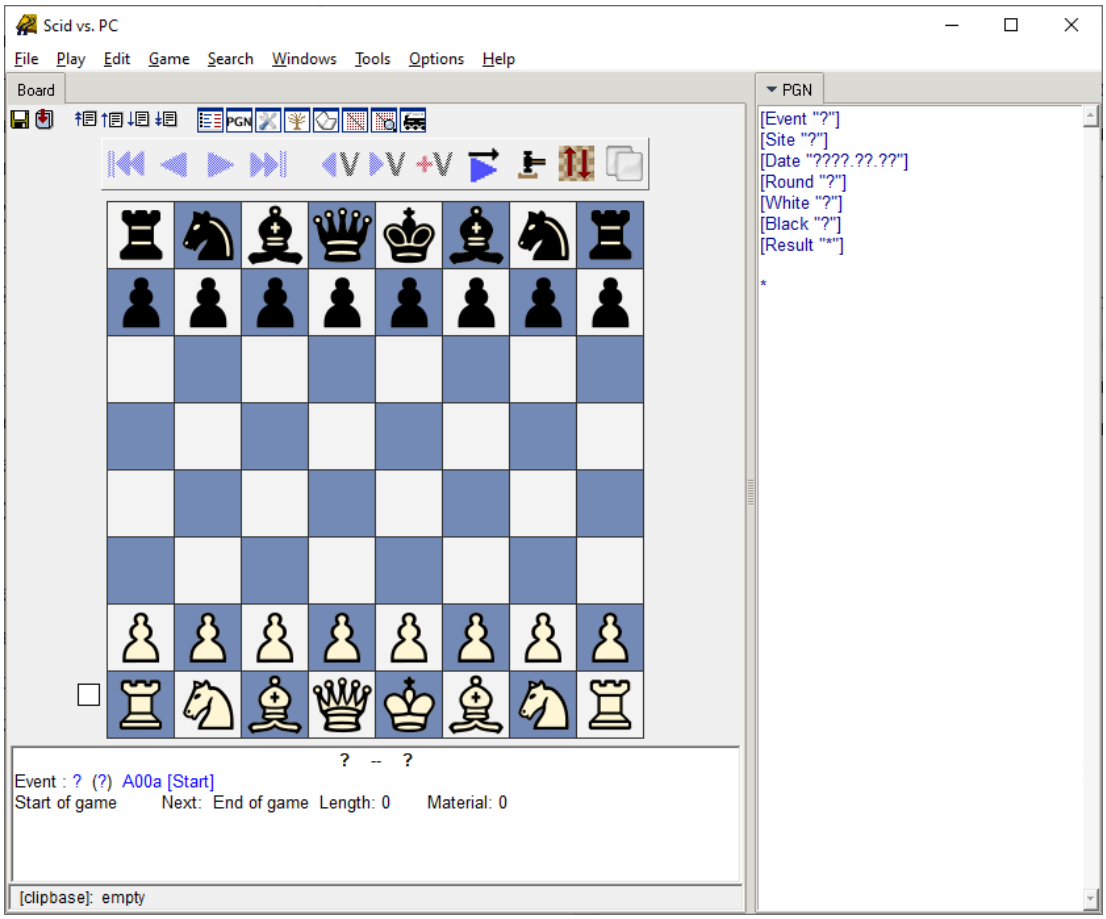

Рисунок 1.4 – Приклад графічного інтерфейсу

Можливість грати проти рушія реалізована за допомогою вибору рушія та отримання результатів з аналізом(рисунок 1.5). На вибір існує декілька рушіїв, та можливість конфігурувати складність рушія(рисунок 1.6). Даний базовий функціонал є необхідним і достатнім для програм такого типу. Дана програма реалізує доволі гнучкі можливості які зможуть задовільнити велику кількість користувачів.

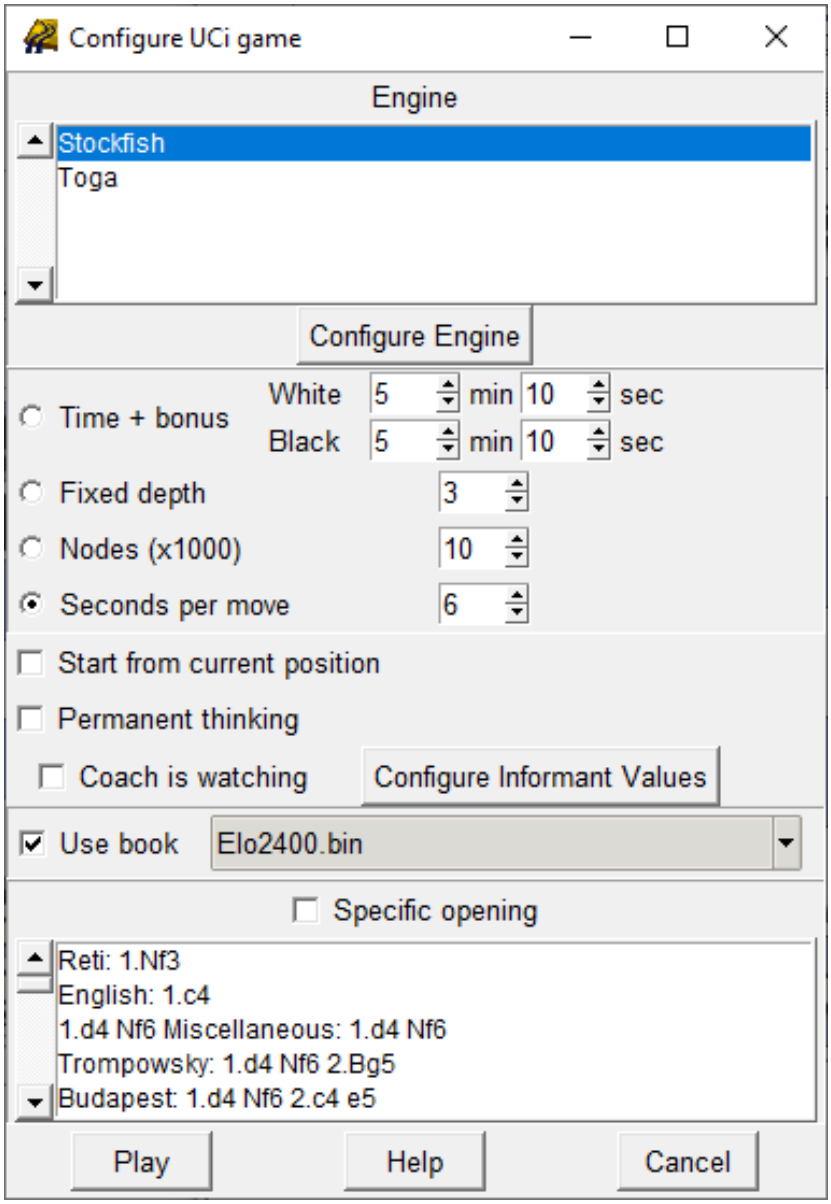

Рисунок 1.5 – Вибір рушія для гри проти нього

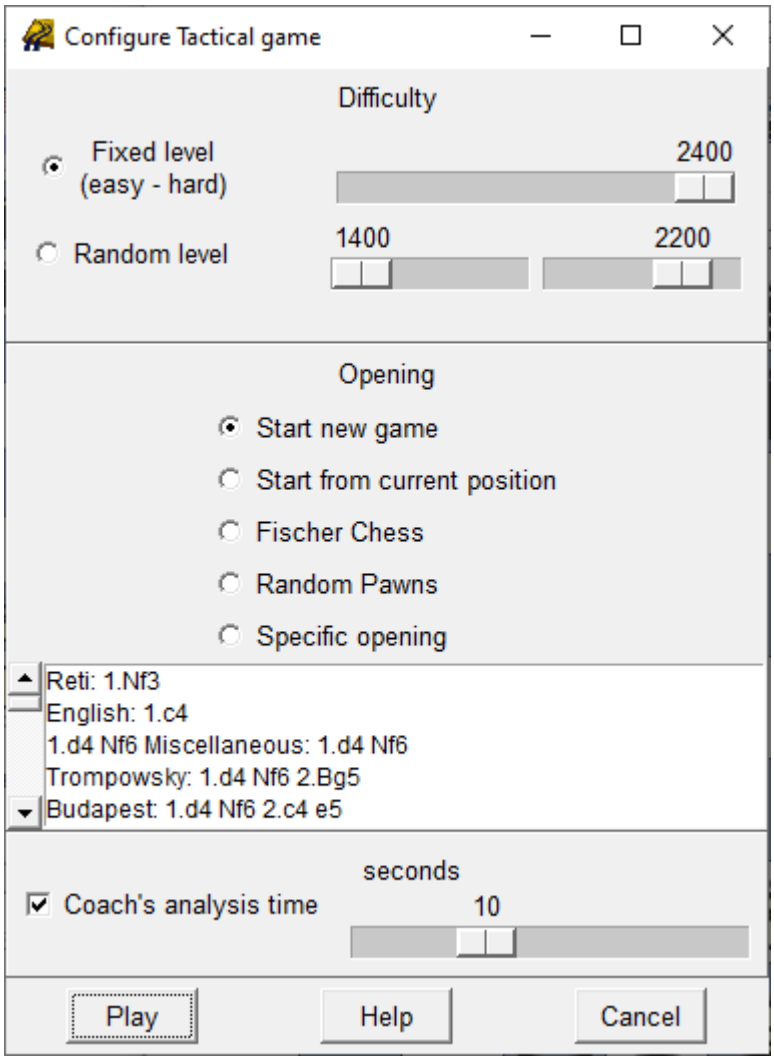

Рисунок 1.6 – Конфігурування складності рушія

Перше що можливо побачити, це старий інтерфейс, на теперішній момент ця програма виглядає застарілою, а новим юзерам неможливо швидко розібратися як все працює.

Після того, як партія завершується, існує можливість передивитися всю партію(рисунок 1.7) та подивитися з якою частотою гравці грають деякі ходи(рисунок 1.8) та отримати найкращі ходи з певної позиції(рисунок 1.9).

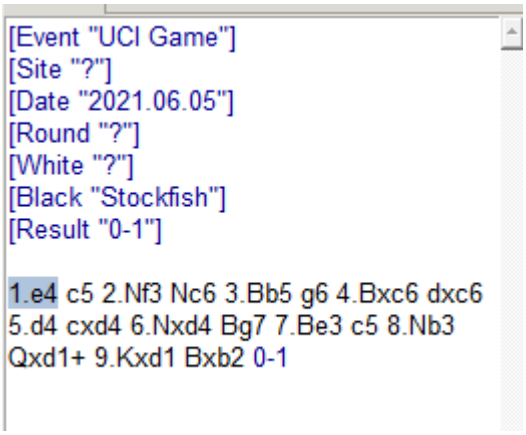

Рисунок 1.7 – Історія партії

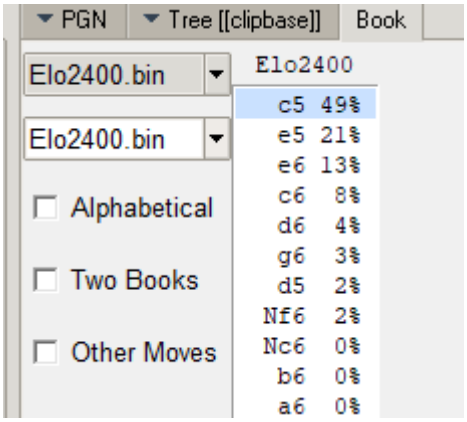

Рисунок 1.8 – Частота хода яку грають в позиції

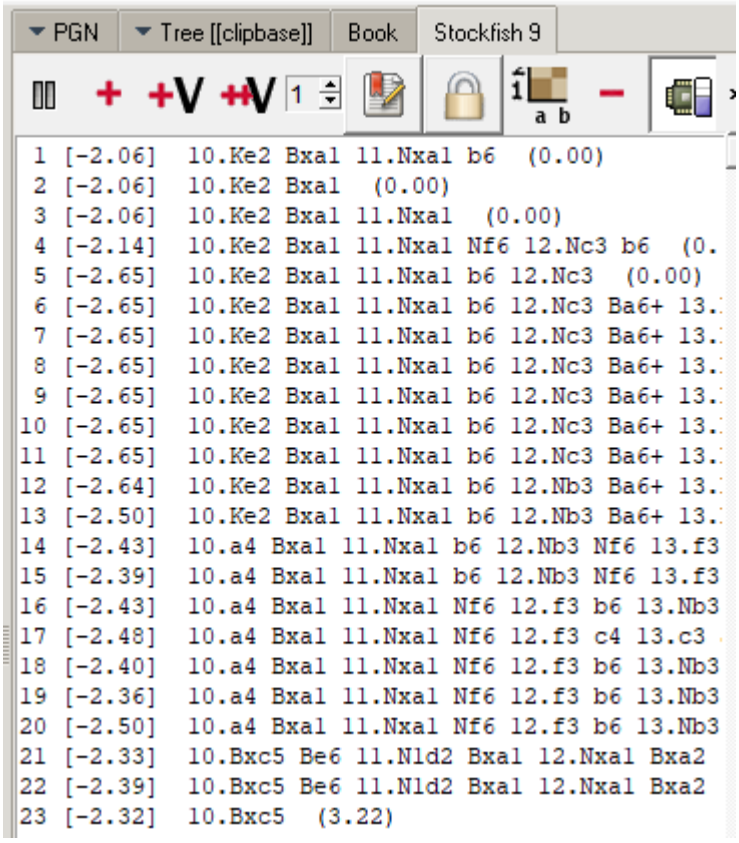

Рисунок 1.9 – Найкращі ходи для певної позиції

Після аналізу інтерфейсу можливо побачити, що в цій програмі є дуже великий недолік – дуже складно знайти потрібний функціонал, та зрозуміти чи він існує, окрім цього зі сторони юзера навіть якщо ти зміг знайти потрібну тобі функцію, дуже важко розібратися з тим результатом який програма повертає.

Підсумовуючи аналіз графічного інтерфейс та його базових можливостей можливо виділити такі переваги:

- Великий функціонал
- Компактність(програма важить менше 100 мегабайтів)

Та недоліки:

- Графічно застарілий інтерфейс
- Незрозумілий для юзера інтерфейс

1.2.5 Arena

Arena – це графічний інтерфейс для шахових рушіїв який розроблений для Windows та з 2016 року для Linux, розроблений Мартіном Блюмом. Команда розробників зробила свій форум та веб-сайт підтримка яких була потім зупинена, проте доступ до графічного інтерфейсу доступний і на теперішній час.

Після відкриття програми дається вибір язиків між німецьким та англійським, що є плюсом для таких програм(рисунок 1.10). Окрім цього перше що можливо зазначити це набагато кращий на вигляд інтерфейс разом з краще структурованим функціоналом, що дозволяє легше розібратися в ньому у порівнянні з минулим прикладом(рисунок 1.11).

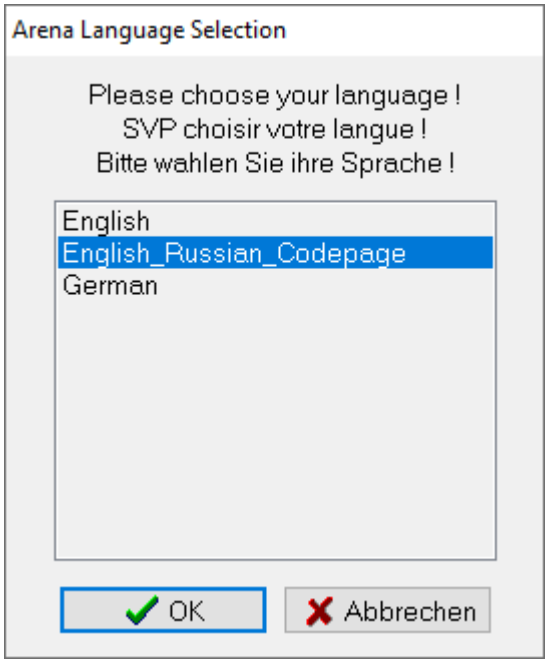

Рисунок 1.10 – Вибір мови

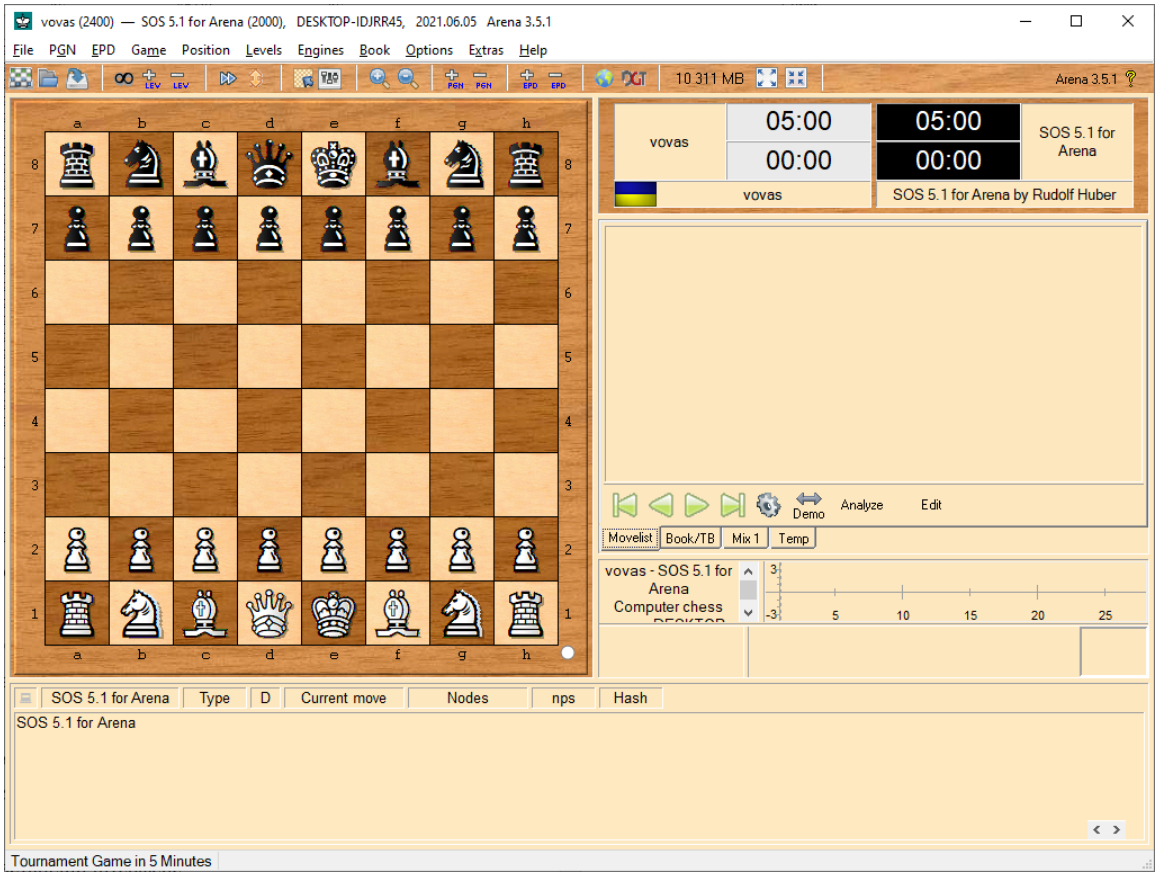

Рисунок 1.11 – Приклад графічного інтерфейсу

В даній програм також існує можливість грати проти самого себе, аналізувати позицію та грати проти вибраного шахового рушія(рисунок 1.12). Необхідно зазначити, шо в ця програма не має можливості вибрати власний рушій, це є мінусом, адже юзер не має можливості підключити більш новий рушій, окрім цього не має можливості змінити складність рушія. Під час гри можливо побачити історію гри та оцінку позиції(рисунок 1.13) разом з запропонованими комп’ютером ходами(рисунок 1.14).

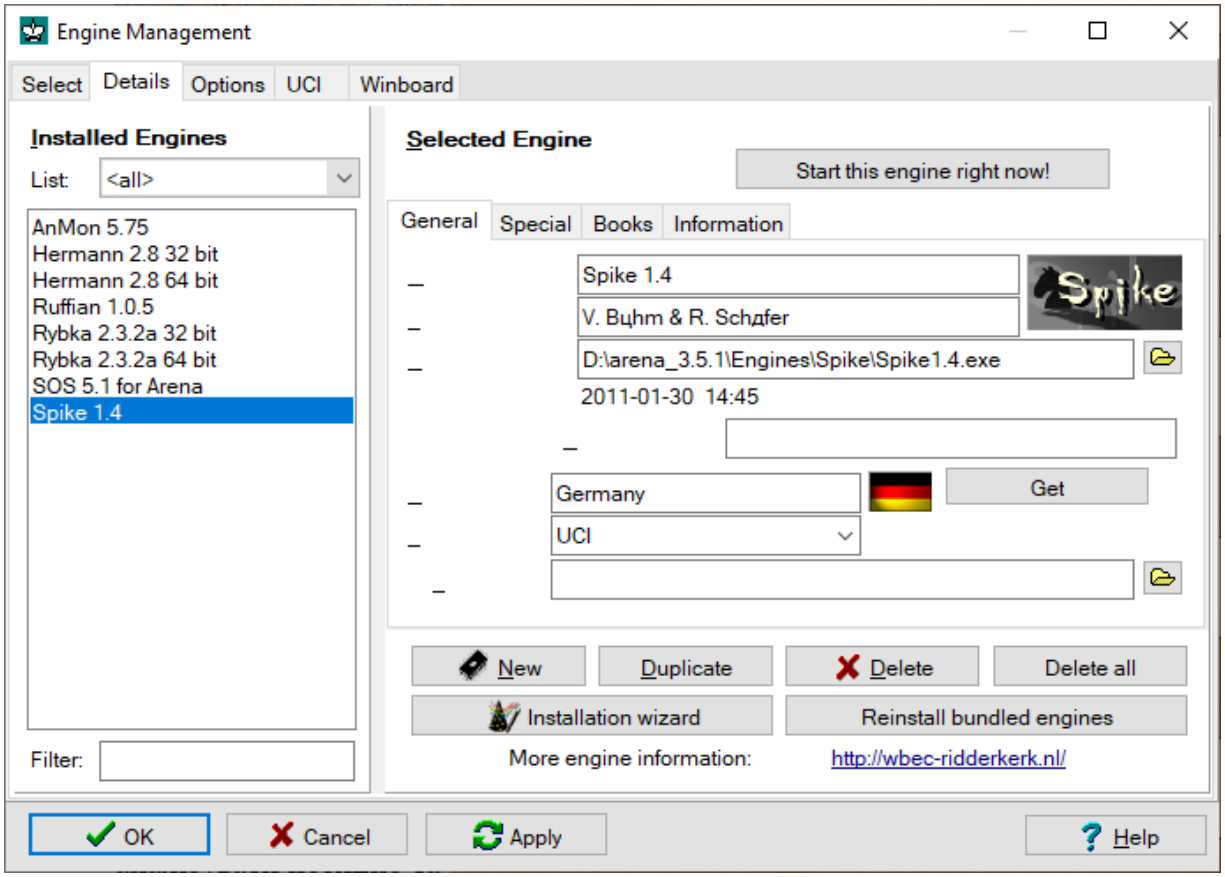

Рисунок 1.12 – Вибір шахового рушія

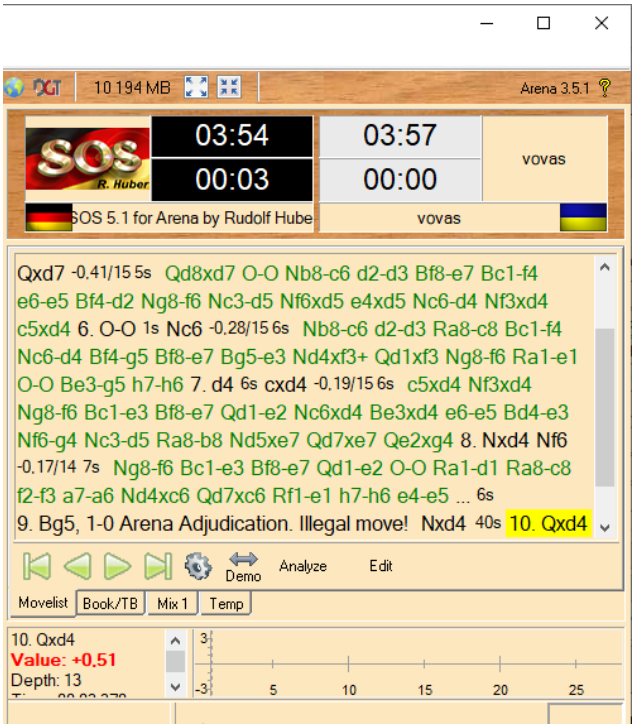

Рисунок 1.13 – Історія гри та оцінка позиції

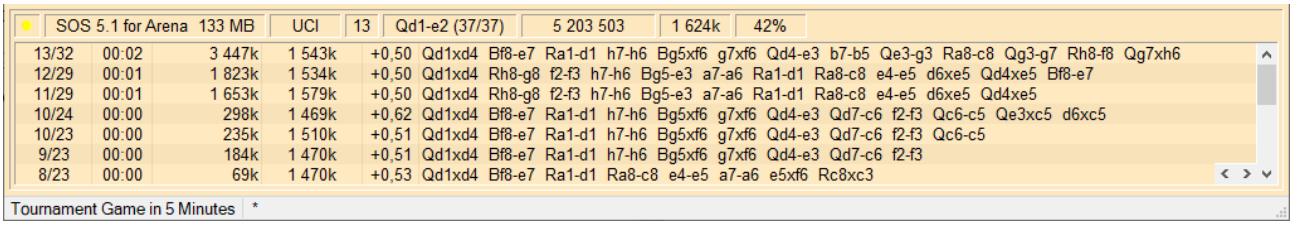

Рисунок 1.13 – Запропоновані комп’ютером ходи

Підсумовуючи аналіз графічного інтерфейс та його базових можливостей можливо виділити такі переваги:

- Гарний та зрозумілий для юзера графічний інтерфейс
- Компактність(програма важить менше 100 мегабайтів)

Та недоліки:

- Доступні рушії іноді дуже довго розраховують наступний хід
- Не має можливості повноцінно конфігурувати шахові рушії

1.3 Постановка задач

Після аналізу декількох шахових рушіїв разом з графічними інтерфейсами були поставлені такі задачі:

- Створити власний шаховий рушій, який зможе аналізувати позицію, та робити відповідні кроки
- Створити власний графічний інтерфейс для цього шахового рушія, за допомогою якого користувач має нагоду взаємодіяти з рушієм

Для виконання поставлених задач необхідно:

- Провести аналіз методологій програмування, та вибрати найбільш доцільну для виконання поставлених задач
- На основі аналізу методологій вибрати мову програмування та провести її аналіз
- Провести аналіз використаних бібліотек(якщо вони необхідні)

2 АНАЛІТИЧНА ЧАСТИНА

2.1 Аналіз методологій програмування

Під час розробки програми для вирішення якоїсь конкретної проблеми, в нашому випадку це розробка шахового рушія разом з графічним інтерфейсом, виконується аналіз цих проблем разом з плануванням розробки та контролем розробки програмного продукти. Сукупність усіх цих кроків називаються методологією програмування. Існує декілька методологій програмування, проте далеко не всі з них є повністю завершеними та чітко сформульованими, мною були вибрані найпопулярнішими з них:

Процедурне програмування – концепція яка заснована на поділ на процедури, які відповідають за якийсь окремий функціонал, а сукупність цих процедур утворюють програму. Ця методологія може використовуватися для нескладних програм, наприклад для створення калькулятора в якому кожна його операція(додавання, віднімання тощо) подається у вигляді окремої процедури, тому вона не підходить для розробки необхідного мені продукту.[8]

Функціональне програмування – По аналогії з процедурним програмуванням ця концепція заснована поділом проблеми на окремі функції. Кожна функція самостійна та виконує якусь конкретну задачу, після чого ці функції об’єднують в програму. Використання цієї методології можливе при розробці прототипу, проте вона доволі погано структурує проект, що веде за собою складність при розробці складного або великого продукту, тому вона також не підходить для розробки.[9]

Об’єктно орієнтоване програмування – це методологія розробки яка передбачає розбиття програми на окремі об’єкти або сутності, та описання як вони буть взаємодіяти між собою, їх логіку та поведінку. Об’єкт являє собою якісь дані та методи які взаємодіють з ними. Задля уособлення об’єктів в програмуванні використовуються класи який має в собі процедури разом з

функціями які реалізують логіку цього об’єкта (як він взаємодіє з іншими об’єктами або окремий його функціонал). Існують три основні парадигми на яких базується ООП – поліморфізм, інкапсуляція та успадкування.

Поліморфізм – це можливість отримувати різний результат під час виклику методів тієї ж назви в одному класі, або декількох класах об’єднаних наслідуванням.

Прикладом цього може слугувати те, що при натисканні клавіши Enter кожні програми будуть реагувати по різному, в якомусь випадку після натискання будуть зчитуватися дані, в іншому виконається перенесення на нову строку.

Інкапсуляція – це обмеження доступу до полів або методів класу з інших об’єктів, тобто можливість приховати деталі об’єкта надаючи інтерфейс який буде взаємодіяти з ним.

Успадкування – це концепція яка дозволяє винести спільні функції або поля до батьківського класу, від якого потім будуть успадковуватися інші класи.

Наприклад усі автомобілі можуть їхати та мають колеса, тому ми можемо винести колеса в окреме поле, а можливість їхати в окремий метод батьківського класу від якого потім будуть успадковуватися інші класи , наприклад клас який реалізує машину марки «BMW» .

Оскільки дана методологія дозволяє дуже гарно структурувати код та спрощує написання складних програм, то при створенні програмного продукту буде використовуватися саме ця методолгія.[10]

2.2 Аналіз мови програмування

На основі аналізу методологій програмування необхідно обрати мову, яка створена за допомогою принципів об’єктно орієнтованого програмування. Мною була обрана мова Java.

Java – одна з найпопулярніших, строго типізована та об’єктно-орієнтована мова програмування яка вперше з’явилася в 1995 році. Додатки на Java зазвичай транслюються в спеціальний байт-код, тому можуть працювати на будь-якій комп’ютерній архітектурі , для якої існує реалізація віртуальної Java-машини(JVM).[1]

Найбільш популярною версією Java і досі є Java 8, незважаючи на те що вже доступна Java 14 більша частина розробників(приблизно 42 відсотки) віддає перевагу 8 версії, а другою по популярності версією(приблизно 11 відсотків), яка лише починає набирати популярність є Java 11, яка вийшла у 2018 році.

Однією з особливостей Java є транслювання свого коду в спеціальний байт-код незалежний від платформи. [11]Цим Java відрізняється від стандартних інтерпретуємих мов код яких одразу виконується інтерпретатором. Проте в той же час Java не є чисто компілюємою мовою як C++ або C.[2]

Java є мовою з «Сі-подобним» синтаксисом, тобто якщо ви володієте наприклад C++ то оволодіти нею буде легше.

Ще однією особливістю Java є збирач сміття, який автоматично звільняє пам’ять від раніше використаних об’єктів, на відміну від вказаного раніше C++.

Java підтримує поліморфізм, наслідування, статичну типізацію. Об’єктно-орієнтований підхід дозволяє вирішувати задачі по створення великих, але в той же час гнучких та розширюваних додатків. [13]

Окрім цього Java є чутливою до регістру – User і user є різними сутностями, для найменування методів використовується lowerCamelCase, а для найменування класів - UpperCamelCase .

У Java назва файлів програми завжди має точно співпадати з назвою класа. Наприклад якщо клас називається ExampleClass, то файл має називатися ExampleClass.java .[3]

Для розробки на мові Java нам необхідний спеціальній комплект інструментів (Oracle JDK або Open JDK). Різницею між ними є ліцензування, Oracle JDK можна використовувати лише для персональних потреб а також для

розробки, тестування, демонстрації продукту. В інших випадках необхідно придбати ліцензію. В той же час Open JDK є повністю безкоштовним. [12]

Під час зборки проекту розробник може використати зовнішні збирачі такі як Maven, Gradle, Ant та інші. Зовнішні збирачі призначені для того, щоб розробник міг описати конфігураційний файл для проекту у текстовому файлі на відміну. Однією з таких популярних систем є Maven який буде використано далі у проекті.

Особливостями та перевагами Maven є:

- Конфігурація Maven лежить в одному файлі – pom.xml, підтримується різні типи зборки – Jar, War, Ear.
- Введена стандартна структура каталогів для проекту, це дозволяє вам по замовченню не вказувати де лежать файли та ресурси та куди їх необхідно помістити після компіляції.
- Підтримується модульна архітектура проекту та профілі.
- Для центрального зберігання бібліотек є центральний репозиторій. Усі популярні бібліотеки публікують та оновлюють у ньому після чого групують по трьом параметрам : groupId, artifactId, version. Завдяки цьому розробнику непотрібно скачувати залежність та класти її у проект , вони об’являються в pom.xml за допомогою тегу dependency та автоматично скачуються при зборці проекту. [14]

2.3 Аналіз необхідних бібліотек

Lombok – це бібліотека для Java яка допомагає у зменшенні часу необхідного для написання коду. Після підключення надається доступ до анотацій (наприклад @ToString або @Data) які дають можливість не писати велику кількість необхідних речей таких як геттери та сеттери, конструктори, методи для порівняння та багато інших, а лише проставити необхідну анотацію.[15]

Після компіляції бібліотека на базі використаних анотацій генерує необхідний код та додає його до вихідних класів. При необхідності ці анотації можливо конфігурувати для того, щоб змінити вихідний код відповідно до потреб. Для її підключення необхідно у файл pom.xml який генерується Maven додати відповідну dependency як показано на рисунку 2.1:

```
<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.16</version>
        <scope>provided</scope>
    </dependency>
```

Рисунок 2.1 - Необхідна залежність

Swing – це бібліотека яка використовується для створення графічних інтерфейсів мовою Java. Swing був розроблений компанією Sun Microsystems. Він містить у собі графічні компоненти такі як таблиці, кнопки, тощо.[4]

Swing відноситься до бібліотек з класу JFC, які є набором бібліотек для створення графічних інтерфейсів. Починаючи з версії Java 1.2 ця бібліотека включена до Java Runtime Environment.

Розробники цієї бібліотеки взяли за основу вже існуючу бібліотеку AWT та доповнили її, зробивши більш гнучкою. Важливою відмінністю Swing від AWT є те, що її компоненти не зв’язані з операційною системою і через це є більш стабільними та швидкими. Такі компоненти в Java називаються легковаговими.[5]

2.4 Аналіз алгоритмів для створення шахового рушія

Існує декілька можливостей для створення рушіїв, найкращими з яких є створення нейронної мережи або програми яка буде перевіряти усі можливі варіанти та відсікати непотрібні за допомогою альфа-бета відсічення. Нижче проведений аналіз обох цих методів.

2.4.1 Нейронні мережі

Штучна **нейронна мережа** – це набір простих елементів, які називають нейронами. Ці нейрони отримують дані на вхід, обробляють ці дані змінюючи свій стан(збуджуються) та роблять вихід який залежить від даних та збудження. Для створення мережи утворюється орієнтовано зважений граф за допомогою під'єднання виходу деяких нейронів з входом інших. Функції які виконують збудження та ваги зазвичай змінюються спеціальним процесом – навчання, який має свої правила задані відповідно до задачі.[7]

Перші спроби імітувати людське мислення було зроблено в 1943 році Уореном Маккалохом та Уолтером Пітсом які створили статтю про працю штучних нейронів та показали модель такої мережи за допомогою електричних схем. Разом з цим було створено модель навчання людини психологами. У 1960 роки під керівництвом Натанієля Рочестера були створені першу штучні мережі. Ці мережі застосовувалися для розв'язування багатьох задач, таких як аналіз штучного зору, погоди, тощо. Проте ці мережі не мали змогу вирішити всі проблеми. Тому з 1970 року почалося активний пошук застосування цих мереж разом з їх покращенням, доки у 2007 році не був створений алгоритм, який виконував глибоке навчання нейронних мереж.

Основними складовими нейронної мережі є **нейрони**, **з'єднання** та **ваги**, **функція поширення** та **правила навчання**.

Розглянемо складові **нейронів**. Нейрон який має мітку j, яка має вхід $p_j(t)$ від попередніх нейронів складається з:

- Функції входу(збудження) – f, ціль якої обчислити нове збудження у конкретний час $t+1$ з $a_j(t), \theta_j$ разом з мережевим входом $p_j(t)$, в результаті чого отримується відношення $a_j(t+1) = f(a_j(t), p_j(t), \theta_j)$
- Збудження $a_j(t)$ залежного від конкретного параметру з часом
- Та $f_{out}(a_j(t))$ – функція виходи яка зазвичай являється тотожною функцією

Окрім цього існують нейрони входу та виходу, які необхідні для входу та виходу у мережу.

Уся мережа має складатися з **з'єднань** , які мають передавати вихід нейрона *a* до входу нейрона *b*. Це значить, що нейрон *a* є попередником нейрону *b* і в той же час нейрон *b* являється наступником для нейрона *a*. Кожне з'єднання має свою **вагу** - w_{ab}.

Після цього, необхідне існування **функції поширення** – яка буде займатися обчисленням входу $p_j(t)$ ведучого до нейрону *j* який отримується з виходів попередніх нейронів - $o_i(t)$ та має вигляд $\sum_i \quad o_i(t)w_{ij}$.

І наприкінці мережа має мати **правило навчання** яке містить алгоритм чи правило за допомогою яких будуть змінюватися параметри мережі задля отримання придатного виходу з входу до цієї мережі. Зазвичай навчання полягає у зміненні ваги разом з порогами змінних мереж.

Після проведення аналізу, необхідно виділити переваги та недоліки використання нейронних мереж для поставлених задач.

Переваги:

- Здатні навчатися на наборах прикладів, що дуже доцільно в шахах, адже існує велика кількість баз партій.
- Такі мережі виконують аналіз набагато швидше, адже вони не перевіряють усі можливі варіанти.
- Подальше покращення гри такого рушія може виконуватися за допомогою навчання, без зміни коду.
- Можливість отримати неочікувані результати проведення гри, які потім можливо аналізувати та використовувати

Недоліки

- Не може повністю імітувати людське мислення, що веде за собою складність при аналізі людиною ігор таких мереж
- Не має можливості повноцінно регулювати глибини обчислювання ходів, через що, налаштувати таку мережу для гри проти людини – дуже складно і може призвести до помилок.
- Необхідність проводити навчання, що значить неможливість одразу після розробки мережі додавання її до програми, окрім цього, для покращення такої мережи необхідно велика кількість часу та ресурсів.
- Необхідність спеціалізованого потужного обладнання для навчання такої мережи

2.4.2 Альфа-бета відсічення

Одним із перших та ефективних алгоритмів для написання шахових рушіїв є звичайний перебір можливих позицій. Рушій перебирає усі можливі позиції які можуть з’явитися після ходу, оцінює їх та обирає варіант з найкращою оцінкою.

Проте недоліком звичайного перебору є те, що він є неоптимізованим і буду кожен раз оцінювати однакові позиції які можна отримати різними ходами, окрім цього будуть оцінюватися навіть ті ходи, які є поганими в будь-якому випадку, що буде дуже сильно впливати на швидкодію такого рушія. Зрозуміло, що коли в людства з'явиться можливість обчислювати усі можливі позиції в шахах за лічені секунди, то такий алгоритм буде найкращим з точки зору оцінювання ходів, проте наразі обчислити 10^{43} можливих позицій неможливо.

Саме тому такі алгоритми намагаються оптимізувати відсікаючи деякі кроки, не обчислюючи повторні позиції та багатьма іншими способами.

При використанні таких алгоритмів для пошуку наступного кроку створюють дерево з теперішнім кроком у початку , ребрами які є кроками та вузлами які є новими позиціями. Для вибору наступного кроку використовують алгоритм який вибере крок який принесе для тебе найбільшу користь та мінімізує вигоду супротивника. Такий алгоритм називається мінімакс.

Для оптимізації даного алгоритму використовується альфа-бета відсікання – головна ідея якого що ми маємо інтервал відсікання з границями альфа та бета відповідно і відсікати всі вузли які не будуть попадати під цей інтервал.

Для оцінки вузлів існує велика кількість методів, проте зазвичай оцінюється матеріал гравця даючи фігурам певну цінність, оцінки взяття, та позиції фігури на дошці. Наприклад 2 слона в центрі будуть мати більшу цінність ніж 2 слони в кутках, 2 тури на одній горизонталі білу ніж 2 тури на різних, тощо.

Переваги:

- Здатні підключити до такого алгоритму бази даних дебютів та ендшпілів дозволяє імітувати людську гру
- Можливість регулювання глибини аналізу ходів, за допомогою чого з'являється можливість конфігурувати рушій для гри проти людини

Недоліки

- Подальше покращення гри такого рушія потребує покращення алгоритму або коду.
- Більш повільна швидкість аналізу

2.5 Висновки

Після аналізу 2 можливих алгоритмів для створення шахового рушія були виділені їх головні переваги та недоліки.

Нейронні мережи є більш швидшим та оптимізованим алгоритмом, проте з багатьма важливими недоліками. Їх дуже важко конфігурувати для гри з реальною людиною, адже єдиною ціллю цієї мережи буде виграти, тому вона сама буде формувати принципи гри та оцінювати позицію, не зважаючи на вже існуючі правила гри. Отримати користь з такої гри зможуть лише професійні гравці які будуть аналізувати партії такого рушія та робити висновки з них. Проте для початківців та звичайних гравців ніякої користі з цього не буде, адже в них буде можливості виграти такий рушій, а на аналіз однієї партії потрібні будуть неділі.

На відмінну від нейронних мереж алгоритм альфа- бета відсікання є менш оптимізованим, та не має можливості покращуватися передивляючись вже існуючи бази ігор, або гравши проти самого себе. Проте головною особливістю даного алгоритму є те, що в нього можливо закласти загальновідомі основи шахів такі як, цінність фігур, де ними краще ходити, та що краще ними брати. Окрім цього зменшення глибини аналізу зможе дати можливість підігнати дану програму під певний рейтинг, що дасть можливість налаштувати її під гру проти початківців.

На основі цього, при розробці рушія для моєї програми буде використано алгоритм альфа-бета відсікання.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Аналіз задачі

Спочатку необхідно розробити логіку для створення програми, для цього було проведено аналіз поставленої перед нами задачі.

При розробці використано ООП методологія програмування та принципи розробки SOLID. При плануванні логіки програми перше що необхідно це розділити поставлену задачу на окремі модулі, які потім необхідно зібрати в необхідний проект.

Створення необхідного прототипу можливо розділити на такі частини:

- Створення та конфігурація проекту, підключення залежностей.
- Створення функціоналу для шахової дошки.
- Створення інтерфейсу та його реалізації який буде реалізовувати шахову дошку.
- Створення алгоритму який буде оцінювати позицію та робити крок

Окрім цього необхідно побудувати діаграми класів.

3.2 Створення та конфігурація проекту, підключення залежностей

Для розробки проекту буде використовуватися Maven у поєднанні з Java 15, тому спочатку необхідно в середі програмування (в моєму випадку – IntelIJ IDEA) створити пустий проект, та налаштувати версію Java разом з підключенням залежностей як показано на рисунках 3.1, 3.2 та 3.3 відповідно.

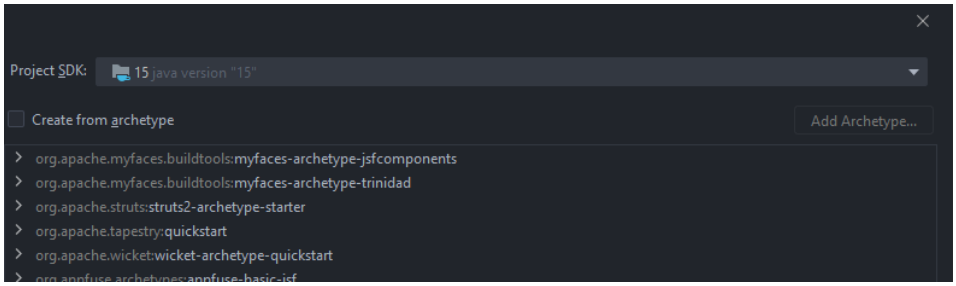

Рисунок 3.1 - Створення пустого проекту

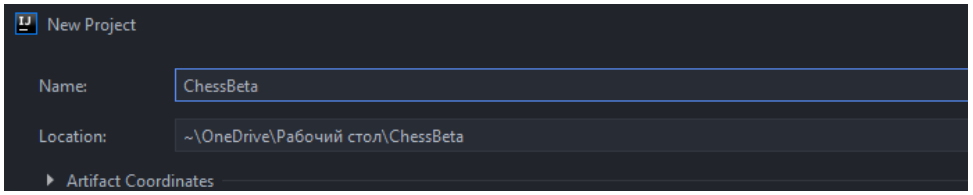

Рисунок 3.2 - Створення пустого проекту

```
<properties>
    <maven.compiler.source>15</maven.compiler.source>
    <maven.compiler.target>15</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.16</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>15</source>
                <target>15</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Рисунок 3.3 - Конфігурація Java разом з підключенням залежностей

3.3 Створення функціоналу для шахової дошки

Перше зо необхідно при розробці нашої програми, це можливість моделювати гру, тому для цього необхідно розробити шахову дошку. Для цього мною було використано масив строк в який були занесе пусті значення та значення які відповідають за деякі фігури(рисунок 3.4).

```
static String chessBoard[][] = {
        {"r", "k", "b", "q", "a", "b", "k", "r"},
        {"p", "p", "p", "p", "p", "p", "p", "p"},
        {" ", " ", " ", " ", " ", " ", " ", " "},
        {" ", " ", " ", " ", " ", " ", " ", " "},
        {" ", " ", " ", " ", " ", " ", " ", " "},
        {" ", " ", " ", " ", " ", " ", " ", " "},
        {"P", "P", "P", "P", "P", "P", "P", "P"},
        {"R", "K", "B", "Q", "A", "B", "K", "R"}};
```

Рисунок 3.4 – Масив для імітації дошки

Де r - тура,k - кінь ,b - слон,q - королева,a - король,p – пішак чорного кольору. Окрім цього ці ж самі букви у великому регістрі позначають ці ж фігури проте білого кольору.

Спочатку був створений метод який перевіряє чи король знаходиться у безпеці перевіряючи чи якась фігура атакує короля. Цей метод необхідний для того щоб була можливість корректно обчислити усі можливі ходи, та реагувати на шах королю.

Окрім цього були зроблені методи які відповідають за знаходження можливих кроків фігур, які перебира.ть цей масив, знаходять перевіряючи куди фігура має походити, та чи буде після цього король у безпеці використовуючи уже існуючий метод, та метод який відпопідають за крок в який передається крок

який юзер намагається зробити, перевіряється чи цей крок є можливим, та якщо є то змінється масив який відповідає за дошку.

3.4 Створення графічного інтерфейсу

Перше що необхідне при створенні графічного інтерфейсу, це головне вікно, на якому будуть відображатися шахова дошка разом з фігурами. Окрім цього необхідно зробити можливість гравцю обирати фігури за які він планує грати та розташувати фігури відповідно до його вибору.

Для розташування фігур на дошці спочатку їх необхідно помістити в картинку з якої буде зчитуватися потрібна нам частина. Було створена одну загальну картинку на якій намальовані варіації усіх фігур для двох кольорів(рисунок 3.5).

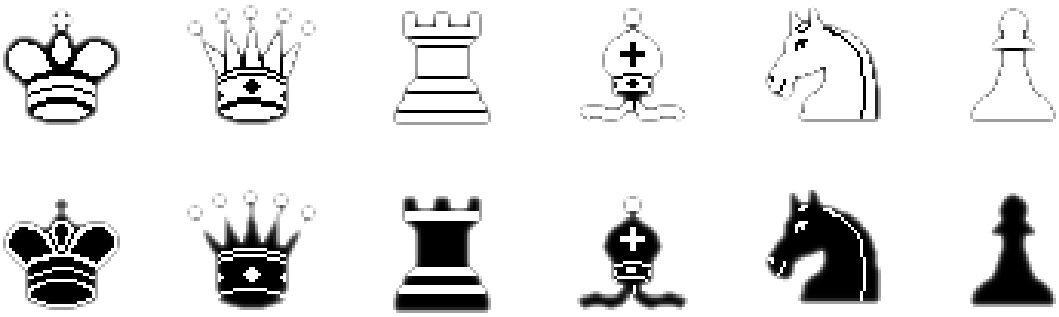

Рисунок 3.5 – Варіації шахових фігур

Для відображення картинки на дошці спочатку була створена дошка, яка є клітинками які по черзі заповнюються двома кольорами та після чого визивається метод який буде проходити по масиву дошки там відображати відповідну частину з малюнка.

Після цього було створено вікно з MouseListener та MouseMotionListener які будуть використовуватися для отримання даних куди користувач буде переводити фігурку для свого кроку. Вони повинні зчитувати координати з яких та куди користувач намагається зробити крок та передавати їх в метод для кроку який було створено раніше перемальовуючи дошку.

3.5 Створення алгоритму оцінки

Спочатку необхідно створити метод який буде обчислювати оцінку позиції, для цього мною були створено масиви з приблизною оцінкої місць на яких фігури будуть найбільш ефективними(пішак, тура, кінь, слон, королева та король на початку та кінці гри відповідно на рисунках 3.6, 3.7, 3.8, 3.9, 3.10, 3.11 відповідно)

```
private Integer pawnEval[][] = {
        {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}};
```

Рисунок 3.6 – Оцінка розташування пішака

```
private Integer rookEval[][] = {
        {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}};
```

Рисунок 3.7 – Оцінка розташування тури

```
private Integer knigtEval[][] = {
        {-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}};
```

Рисунок 3.8 – Оцінка розташування коня

```
private Integer bishopEval[][] = {
        {-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}};
```

Рисунок 3.9 – Оцінка розташування слона

```
private Integer queenEval[][] = {
        {-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}};
```

Рисунок 3.10 – Оцінка розташування королеви

```
private Integer kingStartEval[][] = {
        {-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}};

private Integer kingEndGameEval[][] = {
        {-50, -40, -30, -20, -20, -30, -40, -50},
        {-30, -20, -10, 0, 0, -10, -20, -30},
        {-30, -10, 20, 30, 30, 20, -10, -30},
        {-30, -10, 30, 40, 40, 30, -10, -30},
        {-30, -10, 30, 40, 40, 30, -10, -30},
        {-30, -10, 20, 30, 30, 20, -10, -30},
        {-30, -30, 0, 0, 0, 0, -30, -30},
        {-50, -30, -30, -30, -30, -30, -30, -50}};
```

Рисунок 3.11 – Оцінка розташування короля на початку та кінці партії

Ці масиви заповнені приблизними даними взятими на основі аналізу принципів шахів яких навчають початківців, що зможе імітувати гру реальної людини.

Після чого необхідно розробити можливість оцінки ходу, для чого я спочатку реалізував оцінку фігур які має гравець, для цього я додавав рейтинг за кожну фігуру, та окрім цього додавав додаткові бали за деякі переваги (такі як два слона).

Після оцінки фігур я змінював рейтинг обчислюючи чи буде вигідним взяття надавши кожній фігурі відповідне значення та загальну оцінку позиції яку я брав з масивів які показано раніше відповідно до фігури гравця.

Зібравши усі ці данні я рекурсивно визивав метод який відповідає за альфа бета відсікання аналізуючи усі можливі кроки та повертаючи крок який би мінімізував користь для гравця та максимізував для рушія.

3.6 Діаграми класів

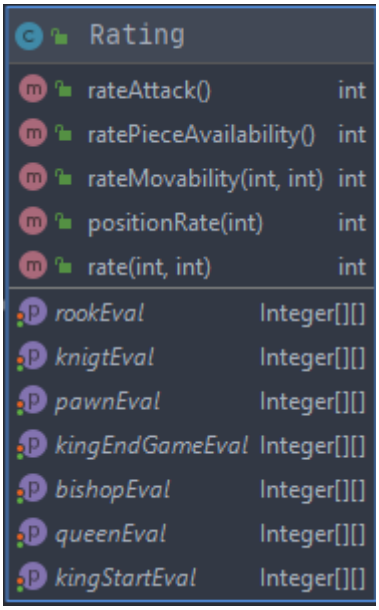

Рисунок 3.12 – Діаграма класу для оцінки

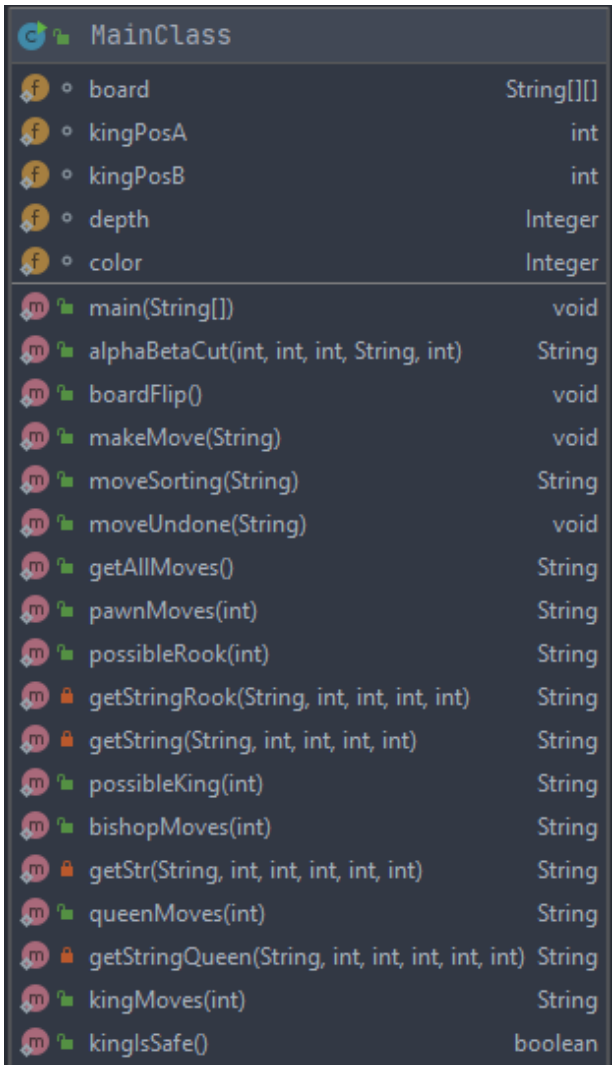

Рисунок 3.13 – Діаграма класу з шаховою дошкою

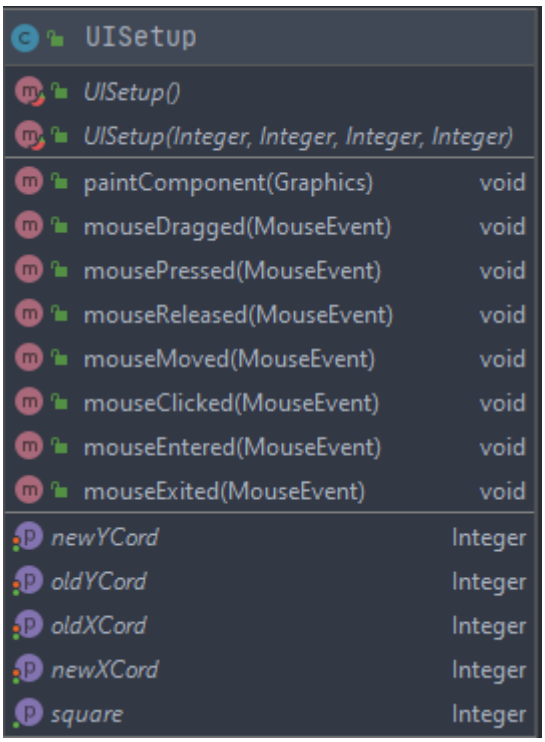

Рисунок 3.14 – Діаграма класів для графічного інтерфейсу

3.7 Результати виконання програми

Після запуску та компіляції програми юзер вибирає колір за який він хочу грати(рисунок 3.15) та грає проти рушія(рисунки 3.16 та 3.17 демонструють хід партії). Виконання поставленого завдання можна рахувати успышним.

Рисунок 3.13 – Діаграма класу з шаховою дошкою

Рисунок 3.13 – Діаграма класу з шаховою дошкою

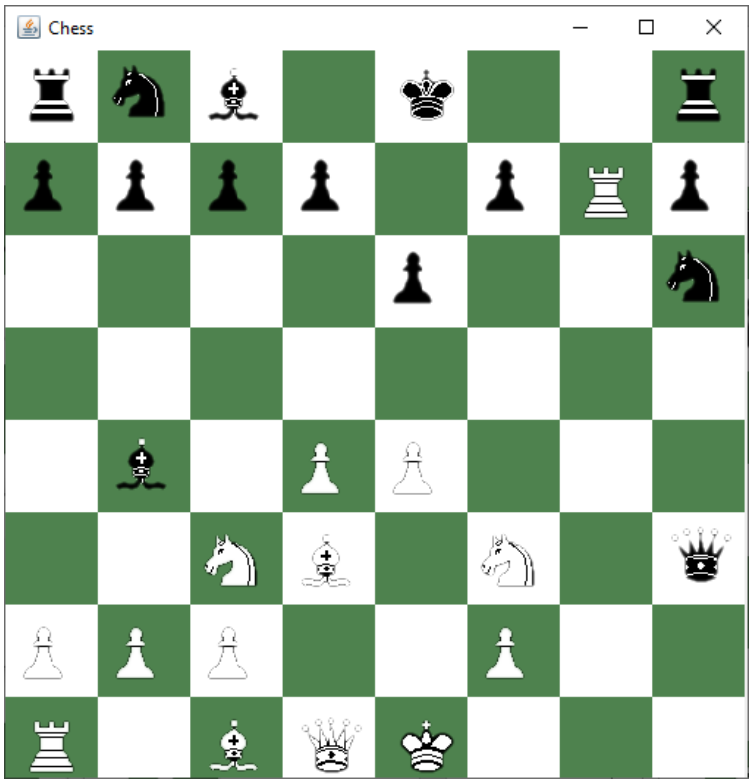

Рисунок 3.13 – Діаграма класу з шаховою дошкою

● ВИСНОВОК

Під час створення дипломної роботи мною було виконано такі задачі:

- Створено власний шаховий рушій, який зможе аналізувати позицію, та робити відповідні кроки. Для цього був проведений аналіз двох можливих алгоритмів, а саме нейронні мережи та альфа-бета відсічення і вибраний останній варіант.
- Створено власний графічний інтерфейс для цього шахового рушія, за допомогою якого користувач має нагоду взаємодіяти з рушієм на базі Java Swing.

Для виконання поставлених задач було провдено:

- Аналіз методологій програмування, та вибрано найбільш доцільну для виконання поставлених задач, а саме об’єктно орієнтоване програмування.
- Окрім цього було додатково проаналізовано принципи SOLID які також були використані при розробці програми
- На основі аналізу методологій вибрано мову програмування java та проведено її аналіз
- Аналіз використаних бібліотек

● СПИСОК ЛІТЕРАТУРИ

1. Герберт Шилдт: Java: руководство для начинающих, 7-е изд. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 816 с.

2. Джошуа Блох: Java. Эффективное программирование: Пер. с англ. - СПб. : "Лори", 2020. - 342 с.

3. Герберт Шилдт: Java. Полное руководство. Том 1: Пер. с англ. - СПб. : " Диалектика ", 2019. – 530-600 с.

4. Джошуа Блох: Swing: Эффектные пользовательские интерфейсы: Пер. с англ. - СПб. : "Лори", 2018. – 143-1 с.

5. Герберт Шилдт: Swing: A Beginner`s Guide : – NY: Wiley, 2019 - 212 с.

6. Ян Гудфеллоу: Глубокое Обучение: Пер. с англ. - СПб. : "Лори", 2019. - 492 с.

7. Введення до методологій програмування [Електронний ресурс]. – Точка доступу:URL: https://coderlessons.com/tutorials/akademicheskii/metodologii-programmirovaniia/metodologii-programmirovaniia-vvedenie

8. Введення до процедурної методології програмування [Електронний ресурс]. – Точка доступу:URL: https://uk.icyscience.com/procedural-programming

9. Введення до функціональної методолгії програмування [Електронний ресурс]. – Точка доступу:URL: https://uk.wikipedia.org/wiki/Функційне_програмування

10. Введення до ООП методології програмування [Електронний ресурс]. – Точка доступу:URL: https://tproger.ru/experts/oop-in-simple-words/

11. Медведев В.І. Особенности объектно-ориентированного программирования на C++/CLI, C# и Java. – Казань: РИЦ «Школа», 2008. – 360 с.: ил. – (Серия «Современная прикладная математика и информатика»).

12. П. Наутон, Г. Шилдт. Java 2. Наиболее полное руководство в подлиннике.– СПб.: БХВ-Петербург, 2000. – 1072 с.: ил.

13. Введеня до мови програмування Java [Електронний ресурс]. – Точка доступу: URL: https://metanit.com/java/tutorial/

14. Інформація про Java [Електронний ресурс]. – Точка доступу: URL: https://ru.wikipedia.org/wiki/Java

15. Інформація про Lombok [Електронний ресурс]. – Точка доступу: URL: https://habr.com/ru/post/345520/

16. Мінімакс та альфа-бета відсікання [Електронний ресурс]. – Точка доступу: URL: https://habr.com/ru/company/edison/blog/474680/

17. Інформація про штучній нейронні мережі [Електронний ресурс]. – Точка доступу: URL: https://futurum.today/shtuchni-neironni-merezhi-shcho-tse-take/

18. Ідея для написання шахового рушівя [Електронний ресурс]. – Точка доступу: URL: https://habr.com/ru/post/111210/

● ДОДАТОК

Лістинг коду програми:

```
import javax.swing.*;

public class MainClass {

   static String board[][] = {
         {"br", "bk", "bb", "bq", "ba", "bb", "bk", "br"},
         {"bp", "bp", "bp", "bp", "bp", "bp", "bp", "bp"},
         {" ", " ", " ", " ", " ", " ", " ", " "},
         {" ", " ", " ", " ", " ", " ", " ", " "},
         {" ", " ", " ", " ", " ", " ", " ", " "},
         {" ", " ", " ", " ", " ", " ", " ", " "},
         {"WP", "WP", "WP", "WP", "WP", "WP", "WP", "WP"},
         {"WR", "WK", "WB", "WQ", "WA", "WB", "WK", "WR"}};

   static int kingPosA, kingPosB;
   static Integer depth = 4;
   static Integer color = -1;

   public static void main(String[] args) {
      while (!"WA".equals(board[kingPosA / 8][kingPosA % 8])) kingPosA++;
      while (!"ba".equals(board[kingPosB / 8][kingPosB % 8])) kingPosB++;

      JFrame mainFrame = new JFrame("Chess");
      mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      UISetup userInterface = new UISetup();
      mainFrame.add(userInterface);
      mainFrame.setSize(522, 535);
      mainFrame.setVisible(true);

      color = JOptionPane.showOptionDialog(null, "Play as white?",
            "Choose color", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE,
            null, null, JOptionPane.YES_OPTION);
      if (color == JOptionPane.NO_OPTION) {
         makeMove(alphaBetaCut(depth, Integer.MAX_VALUE, Integer.MIN_VALUE, "", 0));
         boardFlip();
         mainFrame.repaint();
      }

      mainFrame.repaint();

   }

   public static String alphaBetaCut(int depth, int beta, int alpha, String moveString, int pl) {
      Rating ratine = new Rating();

      String possibleMoves = getAllMoves();
      if (possibleMoves.length() == 0 || depth == 0)
         return moveString + (ratine.rate(possibleMoves.length(), depth) * (pl * 2 - 1));

      possibleMoves = moveSorting(possibleMoves);

      pl = 1 - pl;
```

```
    String returningString = "";
    int valueRate;

    for (int i = 0; i < possibleMoves.length(); i += 5) {
        makeMove(possibleMoves.substring(i, i + 5));
        boardFlip();
        returningString = alphaBetaCut(depth - 1, beta, alpha, possibleMoves.substring(i, i + 5), pl);
        valueRate = Integer.parseInt(returningString.substring(5));
        boardFlip();
        moveUndone(possibleMoves.substring(i, i + 5));

        if (pl == 0) {
            if (valueRate <= beta) {
                beta = valueRate;
                if (depth == MainClass.depth)
                    moveString = returningString.substring(0, 5);
            }
        } else {
            if (valueRate > alpha) {
                alpha = valueRate;
                if (depth == MainClass.depth)
                    moveString = returningString.substring(0, 5);
            }
        }
        if (alpha >= beta) {
            if (pl == 0) {
                return moveString + beta;
            } else {
                return moveString + alpha;
            }
        }
    }
    if (pl == 0) {
        return moveString + beta;
    } else {
        return moveString + alpha;
    }
}

public static void boardFlip() {
    String temp;
    int r, c;
    for (int i = 0; i < 32; i++) {
        r = i / 8;
        c = i % 8;
        if (Character.isUpperCase(board[r][c].charAt(0))) {
            temp = board[r][c].toLowerCase();
        } else {
            temp = board[r][c].toUpperCase();
        }
        if (Character.isUpperCase(board[7 - r][7 - c].charAt(0))) {
            board[r][c] = board[7 - r][7 - c].toLowerCase();
        } else {
            board[r][c] = board[7 - r][7 - c].toUpperCase();
        }
        board[7 - r][7 - c] = temp;
    }
    int kingPosTemp = kingPosA;
    kingPosA = 63 - kingPosB;
    kingPosB = 63 - kingPosTemp;
```

```
}

public static void makeMove(String move) {
    if (move.charAt(4) != 'P') {
        int x1 = Character.getNumericValue(move.charAt(0));
        int y1 = Character.getNumericValue(move.charAt(1));
        int x2 = Character.getNumericValue(move.charAt(2));
        int y2 = Character.getNumericValue(move.charAt(3));
        board[x2][y2] = board[x1][y1];
        board[x1][y1] = " ";
        if ("WA".equals(board[x2][y2])) {
            kingPosA = 8 * x2 + y2;
        }
    } else {
        int col1 = Character.getNumericValue(move.charAt(0));
        int col2 = Character.getNumericValue(move.charAt(1));
        board[1][col1] = " ";
        board[0][col2] = String.valueOf(move.charAt(3));
    }
}

public static String moveSorting(String list) {
    Rating rating = new Rating();
    int[] score = new int[list.length() / 5];
    for (int i = 0; i < list.length(); i += 5) {
        makeMove(list.substring(i, i + 5));
        score[i / 5] = -rating.rate(-1, 0);
        moveUndone(list.substring(i, i + 5));
    }
    StringBuilder newListA = new StringBuilder();
    String newListB = list;
    for (int i = 0; i < Math.min(6, list.length() / 5); i++) {
        int max = -1000000, maxLocation = 0;
        for (int j = 0; j < list.length() / 5; j++) {
            if (score[j] > max) {
                max = score[j];
                maxLocation = j;
            }
        }
        score[maxLocation] = -1000000;
        String substring = list.substring(maxLocation * 5, maxLocation * 5 + 5);
        newListA.append(substring);
        newListB = newListB.replace(substring, "");
    }
    return newListA + newListB;
}

public static void moveUndone(String move) {
    if (move.charAt(4) != 'P') {
        int x1 = Character.getNumericValue(move.charAt(0));
        int y1 = Character.getNumericValue(move.charAt(1));
        int x2 = Character.getNumericValue(move.charAt(2));
        int y2 = Character.getNumericValue(move.charAt(3));

        board[x1][y1] = board[x2][y2];
        board[x2][y2] = String.valueOf(move.charAt(4));
        if ("WA".equals(board[x1][y1])) {
            kingPosA = 8 * x1 + y1;
        }
    } else {
```

```
        int col1 = Character.getNumericValue(move.charAt(0));
        int col2 = Character.getNumericValue(move.charAt(1));
        board[1][col1] = "WP";
        board[0][col2] = String.valueOf(move.charAt(2));
    }
}

public static String getAllMoves() {
    StringBuilder allMoves = new StringBuilder();

    for (int i = 0; i < 64; i++) {
        switch (board[i / 8][i % 8]) {
            case "WP" -> allMoves.append(pawnMoves(i));
            case "WR" -> allMoves.append(possibleRook(i));
            case "WK" -> allMoves.append(possibleKing(i));
            case "WB" -> allMoves.append(bishopMoves(i));
            case "WQ" -> allMoves.append(queenMoves(i));
            case "WA" -> allMoves.append(kingMoves(i));
        }
    }
    return allMoves.toString();
}

public static String pawnMoves(int move) {
    StringBuilder moveList = new StringBuilder();
    String oldPiece;
    int rate = move / 8, c = move % 8;

    for (int j = -1; j <= 1; j += 2) {
        try {
            if (Character.isLowerCase(board[rate - 1][c + j].charAt(0)) && move >= 16) {
                oldPiece = board[rate - 1][c + j];
                board[rate][c] = " ";
                board[rate - 1][c + j] = "WP";
                if (kingIsSafe()) {
                    moveList.append(rate).append(c).append(rate - 1).append(c + j).append(oldPiece);
                }
                board[rate][c] = "WP";
                board[rate - 1][c + j] = oldPiece;
            }
        } catch (Exception ignored) {
        }

        try {
            if (Character.isLowerCase(board[rate - 1][c + j].charAt(0)) && move < 16) {
                String[] temp = {"WQ", "WR", "WK", "WB"};
                for (int k = 0; k < 4; k++) {
                    oldPiece = board[rate - 1][c + j];
                    board[rate][c] = " ";
                    board[rate - 1][c + j] = temp[k];
                    if (kingIsSafe()) {
                        moveList.append(c).append(c + j).append(oldPiece).append(temp[k]).append("WP");
                    }
                    board[rate][c] = "WP";
                    board[rate - 1][c + j] = oldPiece;
                }
            }
        } catch (Exception ignored) {
        }
    }
```

```
        try {
            if (" ".equals(board[rate - 1][c]) && move >= 16) {
                oldPiece = board[rate - 1][c];
                board[rate][c] = " ";
                board[rate - 1][c] = "WP";
                if (kingIsSafe()) {
                    moveList.append(rate).append(c).append(rate - 1).append(c).append(oldPiece);
                }
                board[rate][c] = "WP";
                board[rate - 1][c] = oldPiece;
            }
        } catch (Exception ignored) {
        }
        try {
            if (" ".equals(board[rate - 1][c]) && move < 16) {
                String[] temp = {"WQ", "WR", "WB", "WK"};
                for (int k = 0; k < 4; k++) {
                    oldPiece = board[rate - 1][c];
                    board[rate][c] = " ";
                    board[rate - 1][c] = temp[k];
                    if (kingIsSafe()) {
                        moveList.append(c).append(c).append(oldPiece).append(temp[k]).append("WP");
                    }
                    board[rate][c] = "WP";
                    board[rate - 1][c] = oldPiece;
                }
            }
        } catch (Exception ignored) {
        }

        try {
            if (" ".equals(board[rate - 1][c]) && " ".equals(board[rate - 2][c]) && move >= 48) {
                oldPiece = board[rate - 2][c];
                board[rate][c] = " ";
                board[rate - 2][c] = "WP";
                if (kingIsSafe()) {
                    moveList.append(rate).append(c).append(rate - 2).append(c).append(oldPiece);
                }
                board[rate][c] = "WP";
                board[rate - 2][c] = oldPiece;
            }
        } catch (Exception ignored) {
        }
        return moveList.toString();
    }

    public static String possibleRook(int move) {
        String moves = "", oldPiece;
        int rate = move / 8, c = move % 8;
        int tempo = 1;

        for (int j = -1; j <= 1; j += 2) {
            try {
                while (" ".equals(board[rate][c + tempo * j])) {
                    moves = getString(moves, rate, c, tempo, j);
                    tempo++;
                }
                if (Character.isLowerCase(board[rate][c + tempo * j].charAt(0))) {
                    moves = getString(moves, rate, c, tempo, j);
                }
```

```
            } catch (Exception ignored) {
            }
            tempo = 1;

            try {
                while (" ".equals(board[rate + tempo * j][c])) {
                    moves = getStringRook(moves, rate, c, tempo, j);
                    tempo++;
                }
                if (Character.isLowerCase(board[rate + tempo * j][c].charAt(0))) {
                    moves = getStringRook(moves, rate, c, tempo, j);
                }
            } catch (Exception ignored) {
            }
            tempo = 1;
        }
        return moves;
    }

    private static String getStringRook(String moves, int rate, int c, int tempo, int j) {
        String oldPiece;
        oldPiece = board[rate + tempo * j][c];
        board[rate][c] = " ";
        board[rate + tempo * j][c] = "WR";
        if (kingIsSafe()) {
            moves = moves + rate + c + (rate + tempo * j) + c + oldPiece;
        }
        board[rate][c] = "WR";
        board[rate + tempo * j][c] = oldPiece;
        return moves;
    }

    private static String getString(String moves, int rate, int c, int tempo, int j) {
        String oldPiece;
        oldPiece = board[rate][c + tempo * j];
        board[rate][c] = " ";
        board[rate][c + tempo * j] = "WR";
        if (kingIsSafe()) {
            moves = moves + rate + c + rate + (c + tempo * j) + oldPiece;
        }
        board[rate][c] = "WR";
        board[rate][c + tempo * j] = oldPiece;
        return moves;
    }

    public static String possibleKing(int move) {
        StringBuilder moves = new StringBuilder();
        String oldPiece;
        int rate = move / 8, c = move % 8;

        for (int j = -1; j <= 1; j += 2) {
            for (int k = -1; k <= 1; k += 2) {
                if (j != 0 || k != 0) {
                    try {
                        if (Character.isLowerCase(board[rate + j][c + k * 2].charAt(0)) || " ".equals(board[rate + j][c + k
* 2])) {
                            oldPiece = board[rate + j][c + k * 2];
                            board[rate][c] = " ";
                            board[rate + j][c + k * 2] = "WK";
                            if (kingIsSafe()) {
```

```
                        moves.append(rate).append(c).append(rate + j).append(c + k * 2).append(oldPiece);
                    }
                    board[rate + j][c + k * 2] = oldPiece;
                    board[rate][c] = "WK";
                }
            } catch (Exception ignored) {
            }

            try {
                if (Character.isLowerCase(board[rate + j * 2][c + k].charAt(0)) || " ".equals(board[rate + j * 2][c
+ k])) {
                    oldPiece = board[rate + j * 2][c + k];
                    board[rate][c] = " ";
                    board[rate + j * 2][c + k] = "WK";
                    if (kingIsSafe()) {
                        moves.append(rate).append(c).append(rate + j * 2).append(c + k).append(oldPiece);
                    }
                    board[rate + j * 2][c + k] = oldPiece;
                    board[rate][c] = "WK";
                }
            } catch (Exception ignored) {
            }
        }
    }
}

return moves.toString();
}

public static String bishopMoves(int moves) {
    String move = "", oldPiece;
    int rate = moves / 8, c = moves % 8;
    int temp = 1;

    for (int j = -1; j <= 1; j += 2) {
        for (int k = -1; k <= 1; k += 2) {
            if (j != 0 || k != 0) {
                try {
                    while (" ".equals(board[rate + temp * j][c + temp * k])) {
                        move = getStr(move, rate, c, temp, j, k);
                        temp++;
                    }
                    if (Character.isLowerCase(board[rate + temp * j][c + temp * k].charAt(0))) {
                        move = getStr(move, rate, c, temp, j, k);
                    }
                } catch (Exception ignored) {
                }
                temp = 1;
            }
        }
    }

    return move;
}

private static String getStr(String move, int rate, int c, int temp, int j, int k) {
    String oldPiece;
    oldPiece = board[rate + temp * j][c + temp * k];
    board[rate][c] = " ";
    board[rate + temp * j][c + temp * k] = "WB";
```

```
        if (kingIsSafe()) {
            move = move + rate + c + (rate + temp * j) + (c + temp * k) + oldPiece;
        }
        board[rate + temp * j][c + temp * k] = oldPiece;
        board[rate][c] = "WB";
        return move;
    }

    public static String queenMoves(int move) {
        String moves = "", oldPiece;
        int rate = move / 8, c = move % 8;
        int temp = 1;

        for (int j = -1; j <= 1; j++) {
            for (int k = -1; k <= 1; k++) {
                if (j != 0 || k != 0) {
                    try {
                        while (" ".equals(board[rate + temp * j][c + temp * k])) {
                            moves = getStringQueen(moves, rate, c, temp, j, k);
                            temp++;
                        }
                        if (Character.isLowerCase(board[rate + temp * j][c + temp * k].charAt(0))) {
                            moves = getStringQueen(moves, rate, c, temp, j, k);
                        }
                    } catch (Exception ignored) {
                    }
                    temp = 1;
                }
            }
        }

        return moves;
    }

    private static String getStringQueen(String moves, int rate, int c, int temp, int j, int k) {
        String oldPiece;
        oldPiece = board[rate + temp * j][c + temp * k];
        board[rate][c] = " ";
        board[rate + temp * j][c + temp * k] = "WQ";
        if (kingIsSafe()) {
            moves = moves + rate + c + (rate + temp * j) + (c + temp * k) + oldPiece;
        }
        board[rate + temp * j][c + temp * k] = oldPiece;
        board[rate][c] = "WQ";
        return moves;
    }

    public static String kingMoves(int move) {
        String moveList = "", oldPiece;
        int kingTemp;
        int rate = move / 8, c = move % 8;
        for (int j = 0; j < 9; j++) {
            if (j != 4) {
                try {
                    if (Character.isLowerCase(board[rate - 1 + j / 3][c - 1 + j % 3].charAt(0)) || " ".equals(board[rate -
1 + j / 3][c - 1 + j % 3])) {
                        oldPiece = board[rate - 1 + j / 3][c - 1 + j % 3];
                        board[rate][c] = " ";
                        board[rate - 1 + j / 3][c - 1 + j % 3] = "WA";
                        kingTemp = kingPosA;
```

```
                kingPosA = move + (j / 3) * 8 + j % 3 - 9;
                if (kingIsSafe()) {
                    moveList = moveList + rate + c + (rate - 1 + j / 3) + (c - 1 + j % 3) + oldPiece;
                }
                board[rate][c] = "WA";
                board[rate - 1 + j / 3][c - 1 + j % 3] = oldPiece;
                kingPosA = kingTemp;
            }
        } catch (Exception ignored) {
        }
    }
  }
  return moveList;
}

public static boolean kingIsSafe() {
  int temp = 1;

  for (int j = -1; j <= 1; j += 2) {
    for (int k = -1; k <= 1; k += 2) {
      try {
        while (" ".equals(board[kingPosA / 8 + temp * j][kingPosA % 8 + temp * k])) {
          temp++;
        }

        if ("bb".equals(board[kingPosA / 8 + temp * j][kingPosA % 8 + temp * k]) ||
            "bq".equals(board[kingPosA / 8 + temp * j][kingPosA % 8 + temp * k]))
          return false;
      } catch (Exception ignored) {
      }
      temp = 1;
    }
  }

  for (int j = -1; j <= 1; j += 2) {
    try {
      while (" ".equals(board[kingPosA / 8 + temp * j][kingPosA % 8])) {
        temp++;
      }

      if ("br".equals(board[kingPosA / 8 + temp * j][kingPosA % 8]) ||
          "bq".equals(board[kingPosA / 8 + temp * j][kingPosA % 8]))
        return false;
    } catch (Exception ignored) {
    }

    temp = 1;
    try {
      while (" ".equals(board[kingPosA / 8][kingPosA % 8 + temp * j])) {
        temp++;
      }

      if ("br".equals(board[kingPosA / 8][kingPosA % 8 + temp * j]) ||
          "bq".equals(board[kingPosA / 8][kingPosA % 8 + temp * j]))
        return false;
    } catch (Exception ignored) {
    }
    temp = 1;
  }
```

```
        for (int j = -1; j <= 1; j += 2) {
            for (int k = -1; k <= 1; k += 2) {
                try {
                    if ("bk".equals(board[kingPosA / 8 + j][kingPosA % 8 + k * 2]))
                        return false;
                } catch (Exception ignored) {
                }
                try {
                    if ("bk".equals(board[kingPosA / 8 + j * 2][kingPosA % 8 + k]))
                        return false;
                } catch (Exception ignored) {
                }
            }
        }

        if (kingPosA >= 16) {
            try {
                if ("bp".equals(board[kingPosA / 8 - 1][kingPosA % 8 - 1]))
                    return false;
            } catch (Exception ignored) {
            }
            try {
                if ("bp".equals(board[kingPosA / 8 - 1][kingPosA % 8 + 1]))
                    return false;
            } catch (Exception ignored) {
            }
        }

        for (int j = -1; j <= 1; j++) {
            for (int k = -1; k <= 1; k++) {
                if (j != 0 || k != 0) {
                    try {
                        if ("ba".equals(board[kingPosA / 8 + j][kingPosA % 8 + k]))
                            return false;
                    } catch (Exception ignored) {
                    }
                }
            }
        }

        return true;
    }

}
```

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import javax.swing.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class UISetup extends JPanel implements MouseListener, MouseMotionListener {

    private Integer oldXCord;
    private Integer oldYCord;
    private Integer newXCord;
    private Integer newYCord;

    private final Integer square = 63;

    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        this.addMouseListener(this);
        this.addMouseMotionListener(this);

        for (int i = 0; i < 64; i += 2) {
            g.setColor(new Color(255, 255, 255));
            g.fillRect((i % 8 + (i / 8) % 2) * square, (i / 8) * square, square, square);
            g.setColor(new Color(78, 130, 78));
            g.fillRect(((i + 1) % 8 - ((i + 1) / 8) % 2) * square, ((i + 1) / 8) * square, square, square);
        }

        Image allPiecesImage = new ImageIcon(getClass().getClassLoader()
                .getResource("AllPieces.png")).getImage();

        int xPeaceNumber, yPeaceNumber;

        for (int i = 0; i < 64; i++) {
            xPeaceNumber = -1;
            yPeaceNumber = -1;

            switch (MainClass.board[i / 8][i % 8]) {
                case "WP" -> {
                    xPeaceNumber = 5;
                    yPeaceNumber = 0;
                }
                case "bp" -> {
                    xPeaceNumber = 5;
                    yPeaceNumber = 1;
                }
                case "WR" -> {
                    xPeaceNumber = 2;
                    yPeaceNumber = 0;
                }
                case "br" -> {
                    xPeaceNumber = 2;
```

```
                yPeaceNumber = 1;
            }
            case "WK" -> {
                xPeaceNumber = 4;
                yPeaceNumber = 0;
            }
            case "bk" -> {
                xPeaceNumber = 4;
                yPeaceNumber = 1;
            }
            case "WB" -> {
                xPeaceNumber = 3;
                yPeaceNumber = 0;
            }
            case "bb" -> {
                xPeaceNumber = 3;
                yPeaceNumber = 1;
            }
            case "WQ" -> {
                xPeaceNumber = 1;
                yPeaceNumber = 0;
            }
            case "bq" -> {
                xPeaceNumber = 1;
                yPeaceNumber = 1;
            }
            case "WA" -> {
                xPeaceNumber = 0;
                yPeaceNumber = 0;
            }
            case "ba" -> {
                xPeaceNumber = 0;
                yPeaceNumber = 1;
            }
        }

        if (xPeaceNumber != -1 && yPeaceNumber != -1)

            g.drawImage(allPiecesImage, (i % 8) * square, (i / 8) * square,
                    (i % 8 + 1) * square, (i / 8 + 1) * square,
                    xPeaceNumber * 64, yPeaceNumber * 64, (xPeaceNumber + 1) * 64,
                    (yPeaceNumber + 1) * 64, this);
    }
}

//Check if mouse was dragged inside chess board
@Override
public void mouseDragged(MouseEvent e) {

    if (e.getX() < 8 * square && e.getY() < 8 * square) {
        newXCord = e.getX();
        newYCord = e.getY();
        repaint();
    }
}

//Check if mouse was pressed inside chess board
@Override
public void mousePressed(MouseEvent e) {
    if (e.getX() < 8 * square && e.getY() < 8 * square) {
```

```
            oldXCord = e.getX() / square;
            oldYCord = e.getY() / square;
        }
    }

    //Check if mouse was released inside chess board
    @Override
    public void mouseReleased(MouseEvent e) {
        if (e.getX() < 8 * square && e.getY() < 8 * square) {
            newXCord = e.getX() / square;
            newYCord = e.getY() / square;
            String move;
            if (e.getButton() == MouseEvent.BUTTON1) {
                //Check if move is promotion
                if (newYCord == 0 && oldYCord == 1 && "WP".equals(MainClass.board[oldYCord][oldXCord])) {

                    move = "" + oldXCord + newXCord + MainClass.board[newYCord][newXCord] + "QP";
                    //Check if regular move
                } else {
                    move = "" + oldYCord + oldXCord + newYCord + newXCord +
MainClass.board[newYCord][newXCord];
                }
                String legalMoves = MainClass.getAllMoves();
                if (legalMoves.replaceAll(move, "").length() < legalMoves.length()) {
                    MainClass.makeMove(move);
                    MainClass.boardFlip();
                    MainClass.makeMove(MainClass.alphaBetaCut(MainClass.depth, Integer.MAX_VALUE,
Integer.MIN_VALUE, "", 0));
                    MainClass.boardFlip();
                    repaint();
                }
            }
        }
    }

    @Override
    public void mouseMoved(MouseEvent e) {
    }

    @Override
    public void mouseClicked(MouseEvent e) {
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }

    @Override
    public void mouseExited(MouseEvent e) {
    }
}
```

```
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter

public class Rating {

    public int rateAttack() {
        int rating = 0;
        int tempKingPosition = MainClass.kingPosA;

        for (int i = 0; i < 64; i++) {
            switch (MainClass.board[i / 8][i % 8]) {
                case "WP" -> {
                    MainClass.kingPosA = i;
                    if (!MainClass.kingIsSafe())
                        rating -= 64;
                }
                case "WR" -> {
                    MainClass.kingPosA = i;
                    if (!MainClass.kingIsSafe())
                        rating -= 500;
                }
                case "WK", "WB" -> {
                    MainClass.kingPosA = i;
                    if (!MainClass.kingIsSafe())
                        rating -= 300;
                }
                case "WQ" -> {
                    MainClass.kingPosA = i;
                    if (!MainClass.kingIsSafe())
                        rating -= 900;
                }
            }
        }
        MainClass.kingPosA = tempKingPosition;
        if (!MainClass.kingIsSafe())
            rating -= 200;
        return rating / 2;
    }

    //Rates piece availability
    public int ratePieceAvailability() {
        int rating = 0, bishopCount = 0;
        for (int i = 0; i < 64; i++) {
            switch (MainClass.board[i / 8][i % 8]) {
                case "WP":
                    rating += 100;
                    break;
                case "WR":
                    rating += 500;
                    break;
                case "WK":
                    rating += 300;
                    break;
                case "WB":
                    bishopCount++;
                    break;
                case "WQ":
```

```
                rating += 900;
                break;
            }
        }
        if (bishopCount >= 2) {
            rating += 300 * bishopCount;
        } else {
            if (bishopCount == 1) {
                rating += 250;
            }
        }
        return rating;
    }

    //Rates according to the flexibilty in possible moves
    public int rateMovability(int movesAvailableCount, int depth) {
        int rating = 0;
        rating += movesAvailableCount;    //5 points for each possible move
        if (movesAvailableCount == 0) {    //checkmate or stalemate condition
            if (!MainClass.kingIsSafe()) {    //checkmate
                rating += -200000 * depth;
            } else {    //stalemate
                rating += -150000 * depth;
            }
        }
        return rating;
    }

    //Rates position
    public int positionRate(int mat) {

        int rate = 0;

        for (int i = 0; i < 64; i++) {
            switch (MainClass.board[i / 8][i % 8]) {
                case "WP":
                    rate += pawnEval[i / 8][i % 8];
                    break;
                case "WR":
                    rate += rookEval[i / 8][i % 8];
                    break;
                case "WK":
                    rate += knigtEval[i / 8][i % 8];
                    break;
                case "WB":
                    rate += bishopEval[i / 8][i % 8];
                    break;
                case "WQ":
                    rate += queenEval[i / 8][i % 8];
                    break;
                case "WA":
                    if (mat >= 1750) {
                        rate += kingStartEval[i / 8][i % 8];
                        rate += MainClass.kingMoves(MainClass.kingPosA).length() * 10;
                    } else {
                        rate += kingEndGameEval[i / 8][i % 8];
                        rate += MainClass.kingMoves(MainClass.kingPosA).length() * 30;
                    }
                    break;
            }
```

```
        }
        return rate;
    }

    //rates position with depth
    public int rate(int avaliableMovesCount, int depth) {
        int rating = 0;
        int piecAvailable = ratePieceAvailability();

        rating += rateAttack() + piecAvailable + rateMovability(avaliableMovesCount, depth) +
positionRate(piecAvailable);
        MainClass.boardFlip();
        rating -= rateAttack() - piecAvailable - rateMovability(avaliableMovesCount, depth) -
positionRate(piecAvailable);
        MainClass.boardFlip();
        return -(rating + depth * 50);
    }
}
```