Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ІНТЕЛЕКТУАЛЬНИХ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ГРИ В ШАХИ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-3
Чароян Д.О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Гороховатський В.О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектронік

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Чарояну Дмитру Олексійовичу
(прізвище, ім'я, по батькові)

1.Тема роботи Розроблення інтелектуальних програмних засобів для гри в шахи

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 25 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали наукових конференцій, дані інтернет-мережі, методи моделювання в ігрових програмах, середовище розробки Intellij idea.

4. Перелік питань, що потрібно опрацювати в роботі
1.Методи аналізу та прийняття рішень у шаховому штучному інтелекті.
2.Моделі оцінки шахових позицій із використанням алгоритму мінімакс.
3.Здійснення комп'ютерного моделювання шахової гри.
4.Аналіз результатів програмного моделювання для гри шахового бота

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми створення шахового бота. Постановка задачі: розробка інтелектуального агента для шахів. Схеми роботи алгоритму мінімакс та його оптимізації. Візуалізація процесу вибору ходів ботом. Таблиці точності прогнозування найкращих ходів та швидкодії алгоритму. Гістограмні характеристики ефективності алгоритму мінімакс у різних шахових позиціях.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методу	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Гороховатський В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 61 с., 21 рис., 8 лістингів., 2 формули., 41 джерело.

ШАХОВИЙ ШТУЧНИЙ ІНТЕЛЕКТ, АЛГОРИТМ МІНІМАКС, ОЦІНЮВАЛЬНА ФУНКЦІЯ, ПРИЙНЯТТЯ РІШЕНЬ, ГЛИБИНА ПОШУКУ.

Об’єктом роботи є процес прийняття рішень у шаховій грі за допомогою алгоритмів штучного інтелекту.

Метою роботи є розробка шахового бота, що використовує алгоритм мінімаксу для ефективного прийняття рішень у шаховій партії.

У процесі розробки реалізовано класичний алгоритм мінімаксу. Досліджено вплив глибини пошуку на ефективність гри та швидкодію алгоритму. Проведено тестування з використанням різних функцій оцінювання позицій, що дозволило визначити найбільш релевантний підхід до оцінки шахових ходів.

У результаті роботи програмно реалізовано шахового бота, який здатний аналізувати позиції, прогнозувати ходи супротивника та приймати оптимальні рішення у межах заданої глибини пошуку.

CHESS ARTIFICIAL INTELLIGENCE, MINIMAX ALGORITHM, EVALUATION FUNCTION, DECISION MAKING, SEARCH DEPTH.

The object of the work is the decision-making process in chess using artificial intelligence algorithms.

The aim of the work is to develop a chess bot that uses the minimax algorithm for effective decision-making in a chess game.

During the development process, the classical minimax algorithm was implemented. The influence of search depth on gameplay efficiency and algorithm speed was studied. Testing was conducted using various position evaluation functions, allowing for the identification of the most relevant approach to chess move evaluation.

As a result of the work, a chess bot was developed that can analyze positions, predict opponent moves, and make optimal decisions within a given search depth.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI (Artificial Intelligence) – штучний інтелект

ШІ – штучний інтелект

α-β (Alpha-Beta Pruning) – альфа-бета відсікання, оптимізація алгоритму мінімаксу

Minimax – алгоритм прийняття рішень у грі з двома суперниками, спрямований на мінімізацію максимального виграшу опонента

Eval (Evaluation Function) – функція оцінки шахової позиції

Depth (Глибина пошуку) – рівень, до якого алгоритм аналізує можливі ходи вперед

Game tree – дерево всіх можливих ходів з поточного стану

IDE (Integrated Development Environment) – середовище розробки

Heuristic – правило або метод для швидкого прийняття рішень

Stalemate – пат

GUI (Graphical User Interface) – графічний інтерфейс

PGN (Portable Game Notation) – нотація портативних ігор

ВСТУП

Шахи є однією з найстаріших та найскладніших інтелектуальних ігор, яка вимагає від гравців стратегічного мислення, глибокого аналізу та прогнозування дій супротивника. Завдяки стрімкому розвитку обчислювальної техніки та алгоритмів штучного інтелекту комп'ютерні шахові програми досягли рівня гри, який може перевершувати навіть найсильніших людських гросмейстерів [1-3].

Розробка шахових ботів є важливим напрямом у сфері штучного інтелекту, оскільки вона дозволяє досліджувати алгоритми прийняття рішень, оптимізації пошуку та обробки великих обсягів даних у реальному часі. Одним із найефективніших методів, що використовуються у шахових програмах, є алгоритм мінімаксу з альфа-бета відсіканням, який дозволяє значно скоротити обчислювальні витрати та підвищити продуктивність системи [3-6].

Актуальність теми обумовлена зростаючим інтересом до розробки інтелектуальних систем у шахах, які можуть бути використані як для аналізу партій, так і для тренування шахістів різного рівня. Сучасні шахові програми застосовуються не тільки в змаганнях між людьми та комп'ютерами, а й у навчальних процесах, допомагаючи шахістам вдосконалювати свої навички. Вони дозволяють гравцям аналізувати власні помилки, прогнозувати можливі варіанти ходів і оцінювати позицію на дошці з високою точністю.

У даній роботі буде розглянуто процес створення шахового бота на основі алгоритму мінімаксу, а також проведено аналіз ефективності його функціонування.

Для досягнення поставленої мети будуть використані методи математичного моделювання, аналізу алгоритмів, а також експериментальні дослідження продуктивності різних варіантів реалізації шахового бота. У процесі роботи буде проаналізовано сильні та слабкі сторони алгоритму мінімаксу, а також можливі шляхи його вдосконалення.

Результати бакалаврської атестаційної роботи можуть бути використані для подальшого розвитку шахових програм та удосконалення алгоритмів штучного інтелекту в сфері комп’ютерних ігор.

Таким чином, дана робота спрямована на дослідження одного з найперспективніших напрямів штучного інтелекту та його застосування в інтелектуальних іграх. Реалізація шахового бота на основі алгоритму мінімаксу дозволить не лише отримати цінні прикладні висновки, а й створити практично корисний інструмент для шахістів та дослідників у сфері комп’ютерного інтелекту.

1 МЕТОДИ АНАЛІЗУ ПОЗИЦІЙ У ШАХОВІЙ ГРІ

1.1 Термінологія і основні задачі шахового штучного інтелекту

Штучний інтелект (ШІ) у шаховій грі є однією з найвідоміших і найуспішніших галузей застосування комп'ютерних технологій у ігровій індустрії. Шахи, як гра з чіткими правилами та величезною кількістю можливих позицій, стали ідеальним полігоном для розвитку алгоритмів ШІ. У цій роботі розглянемо основну термінологію, пов'язану з шаховим ШІ, а також головні задачі, які вирішуються в цій галузі.

Основна задача шахового ШІ – знаходити найкращі ходи у будь-якій позиції. Для цього використовуються алгоритми пошуку, такі як мінімакс і альфа-бета відсікання, які дозволяють аналізувати тисячі або мільйони позицій за короткий час. Важливим аспектом є баланс між глибиною пошуку і швидкістю обчислень.

Шаховий ШІ також може ефективно прогнозувати стратегії супротивника, спираючись на його попередні ходи та загальну модель гри. Такий прогноз дозволяє ШІ адаптувати свою стратегію і планувати дії на кілька ходів вперед. Прогнози і планування вимагають глибокого аналізу можливих варіантів розвитку гри, що потребує значної кількості обчислювальних ресурсів [6].

Шаховий ШІ повинен вміти оцінювати позицію, враховуючи різні фактори, такі як матеріальний баланс, контроль над ключовими полями, структура пішаків, активність фігур тощо. Для цього використовуються як традиційні евристичні функції, так і сучасні методи машинного навчання.

Шахові рушії, такі як AlphaZero від компанії DeepMind і Leela Chess Zero, мають в основі технології машинного навчання, що дозволяє їм значно підвищити ефективність гри. Вони навчаються через гру з самими собою, що

дозволяє не тільки вивчати вже існуючі стратегії, але й генерувати нові підходи до гри, які раніше були недоступні навіть досвідченим гравцям.

AlphaZero, наприклад, використовуючи глибокі нейронні мережі та підкріплювальне навчання, за кілька годин тренування змогла досягти рівня майстрів та навіть переграти інші суперсучасні шахові рушії. Це відкриття стало справжнім проривом у шахах, оскільки продемонструвало, що ШІ може не лише вивчати вже відомі стратегії, але й винаходити нові, інколи зовсім незвичні для людського розуміння варіанти гри [7].

Однією з найбільш складних частин шахової гри для ШІ є ендшпіль. Це пов’язано з тим, що на пізніх етапах гри кількість фігур на дошці значно зменшується, і тому навіть незначні похибки можуть призвести до катастрофічних наслідків. Аналіз ендшпілів вимагає від програм точних знань про результат кожної можливої позиції, що може бути досягнуто через спеціалізовані бази даних, що містять вже готові рішення для сотень тисяч позицій.

Програми для гри в шахи можуть використовувати ці бази даних для ідеального виконання ходів у кінцівках, гарантуючи перемогу або нічию, а також точно знаючи, коли слід відмовитись від боротьби. Це дозволяє прогресивно підвищувати якість гри і знижувати ймовірність помилок на завершальних етапах партії.

Шаховий ШІ також може ефективно прогнозувати стратегії супротивника, спираючись на його попередні ходи та загальну модель гри. Такий прогноз дозволяє ШІ адаптувати свою стратегію і планувати дії на кілька ходів вперед. Прогнози і планування вимагають глибокого аналізу можливих варіантів розвитку гри, що потребує значної кількості обчислювальних ресурсів.

Однак, незважаючи на значний прогрес у розвитку шахового ШІ, існує кілька серйозних проблем, з якими стикаються розробники шахових рушіїв і однією з головних проблем шахового ШІ є обмеженість обчислювальних

ресурсів. Оскільки шахи містять величезну кількість можливих варіантів гри, навіть найсучасніші алгоритми можуть зіткнутись з проблемою перевантаження, коли кількість розрахунків за обмежений час стає непридатною.

Сучасні шахові рушії намагаються адаптуватися до стилю гри суперника, аналізуючи його попередні ходи і визначаючи слабкі сторони. Це дозволяє ШІ вибирати стратегії, які максимально ускладнюють гру противнику.

Шаховий ШІ не лише аналізує існуючі стратегії, але й генерує нові ідеї. Наприклад, AlphaZero відкрила нові підходи до гри, які раніше не використовувалися людьми, що свідчить про творчий потенціал ШІ.

У реалізації шахового штучного інтелекту важливим аспектом також є обчислювальна складність алгоритмів [8], це характеристика, яка описує, скільки обчислювальних ресурсів, зокрема часу або пам’яті, потрібно для його виконання залежно від розміру вхідних даних. Найпоширенішим способом її подання є асимптотична нотація «велике O» (Big O notation), яка показує, як швидко зростає час або обсяг пам’яті, необхідні для виконання алгоритму, коли обсяг вхідних даних прямує до нескінченності. Існують різні типи складності, зокрема часова, яка враховує кількість базових операцій що виконуються, та просторова, яка визначає скільки оперативної пам’яті потрібно.

Серед базових класів часової складності можна виділити постійну O(1), де час виконання не залежить від розміру даних; лінійну O(n), де час зростає прямо пропорційно до кількості елементів; квадратичну $O(n^2)$, яка типова для простих алгоритмів сортування; та експоненційну $O(2^n)$ чи O(b^d), що характерна для повного перебору або дерева рішень у складних завданнях, наприклад, у настільних іграх

Для багатьох завдань одним з яких є розробка алгоритмів для шахів важливо досягти компромісу між точністю результату та допустимим часом

обчислення, особливо коли йдеться про реальний час або обмежені ресурси. Тому на практиці часто використовуються евристики, наближені методи або комбіновані підходи, які дозволяють знизити обчислювальні витрати при прийнятному рівні якості розв’язання.

1.2 Аналіз алгоритмів шахових ботів та їх модифікацій

Алгоритм мінімакса є одним із фундаментальних підходів у штучному інтелекті для прийняття рішень у іграх з нульовою сумою, таких як шахи, шашки або го [9]. Він дозволяє визначити оптимальну стратегію для гравця, враховуючи, що противник також грає оптимально. Однак через велику кількість можливих ходів у таких іграх алгоритм мінімакса може бути дуже ресурсомістким. Для оптимізації цього процесу було розроблено модифікацію під назвою α-β відсікання, яка значно скорочує кількість аналізованих ходів. Розглянемо принцип роботи алгоритму мінімакса, його недоліки та основні ідеї α-β відсікання.

Алгоритм мінімакса базується на концепції дерева гри, де кожен вузол представляє стан гри (позицію), а кожне ребро можливий хід. Метою алгоритму є ціль знайти такий шлях у дереві, який максимізує виграш для поточного гравця, враховуючи, що противник намагається мінімізувати цей виграш [9].

На рисунку 1.1 представлена концепція дерева гри з якої можна побачити як алгоритм мінімакса перебирає можливі варіанти ходів у ендшпілі.

Побудова дерева гри реалізується так. Алгоритм будує дерево, де кореневий вузол – це поточна позиція, а кожен наступний рівень відповідає можливим ходам гравця або противника.

На листках дерева (кінцевих позиціях) використовується функція оцінки, яка визначає, наскільки вигідною є позиція для поточного гравця.

Алгоритм рекурсивно проходить дерево, по черзі максимізуючи і мінімізуючи значення оцінки на кожному рівні.

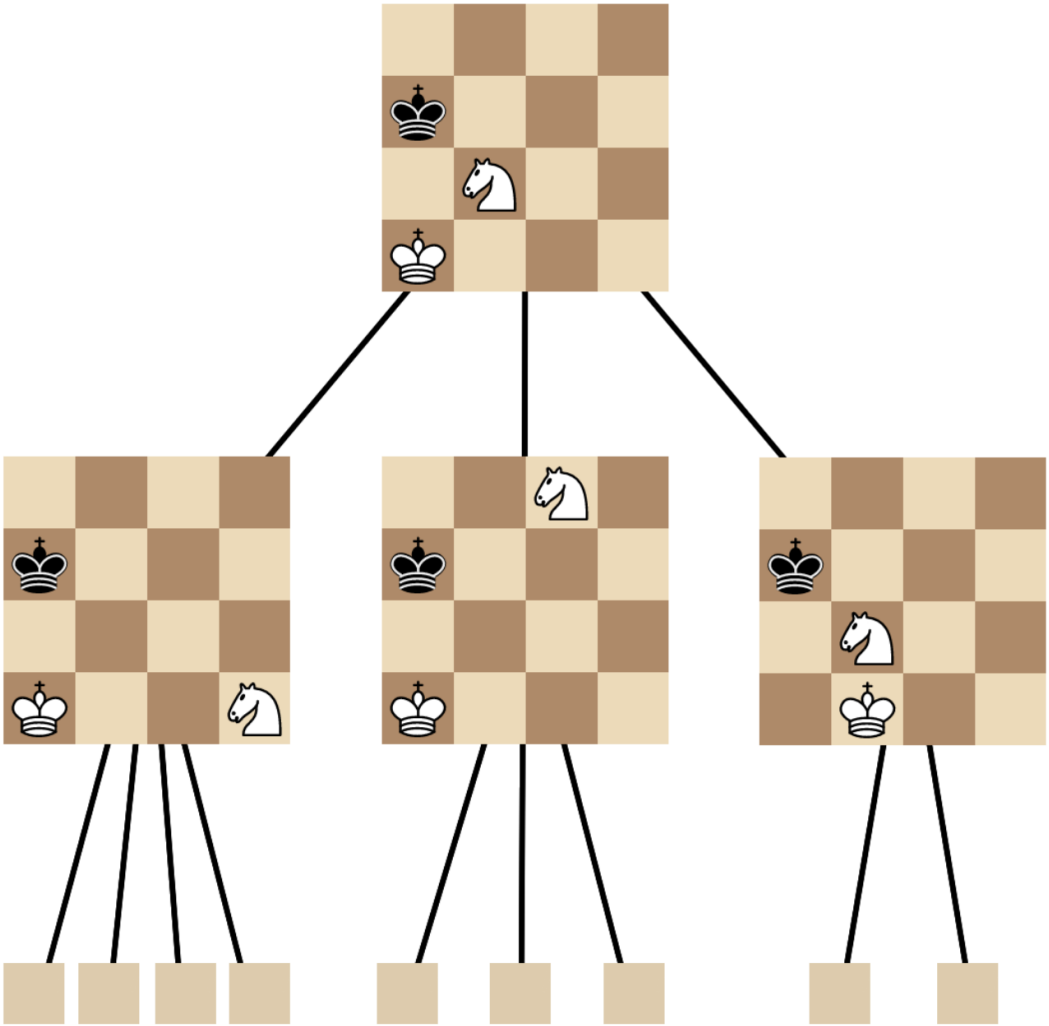

Рисунок 1.1 – Концепція дерева гри [10]

На рівні максимізації (гравець) алгоритм вибирає хід із максимальною оцінкою. На рівні мінімізації (противник) алгоритм вибирає хід із мінімальною оцінкою. На кореневому рівні алгоритм повертає оптимальний хід, який веде до найкращого можливого результату.

Недоліки алгоритму мінімакс полягають у наступному [11].

– у шахах кількість можливих ходів навіть на кілька кроків вперед може бути дуже великою. Наприклад, у середньому на кожен хід є близько 35 можливих варіантів, що призводить до експоненційного зростання кількості позицій;

– через обмеженість обчислювальних ресурсів алгоритм може аналізувати лише обмежену кількість ходів вперед, що може призводити до субоптимальних рішень.

Альфа-бета відсікання (α-β pruning) – це оптимізація алгоритму мінімакса, яка дозволяє скоротити кількість аналізованих ходів, відкидаючи гілки дерева, які заведомо не вплинуть на кінцевий результат. Це досягається за рахунок збереження двох значень, α (альфа) – найкраща оцінка для максимізатора (поточного гравця), та β (бета) – найкраща оцінка для мінімізатора (противника).

Принцип роботи альфа-бета відсікання полягає в ініціалізації (наприклад $\alpha = -\infty$, $\beta = +\infty$) та рекурсивноиу аналізу, де на рівні максимізації алгоритм оновлює значення α, якщо знайдено кращий хід. А на рівні мінімізації алгоритм оновлює значення β, якщо знайдено гірший хід для максимізатора.

Якщо на рівні максимізації поточне значення α стає більшим або рівним β, то подальший аналіз цієї гілки припиняється, оскільки противник ніколи не дозволить максимізатору отримати кращий результат.

Аналогічно, на рівні мінімізації, якщо поточне значення β стає меншим або рівним α, подальший аналіз припиняється.

Переваги α-β відсікання – завдяки відсіканню неперспективних гілок кількість аналізованих позицій значно зменшується, що дозволяє збільшити глибину пошуку. Результат роботи алгоритму з α-β відсіканням ідентичний результату звичайного мінімакса, оскільки відсікаються лише ті гілки, які не впливають на кінцевий вибір ходу.

Алгоритм спочатку аналізує дерево на невеликій глибині, а потім поступово збільшує її, що дозволяє швидше знаходити оптимальні ходи. Використання евристик для оцінки позицій дозволяє скоротити кількість аналізованих ходів, зосередившись на найперспективніших, а збереження результатів аналізу позицій у хеш-таблицях дозволяє уникнути повторних обчислень.

Також крім мінімаксу є і більш складні алгоритми, такі як AlphaZero, цей алгоритм навчається з використанням нейромережі [12]. AlphaZero починає з нуля – великої нейронної мережі з випадковими вагами. Вона створена, щоб навчатися ігор із двома супротивниками, які роблять ходи по черзі, але нічого не знає про кожну окрему гру. Перший крок – AlphaZero вчиться робити хоч і випадкові, але допустимі ходи. Далі AlphaZero грає мільйони партій сама з собою. Після кожної партії вона коригує вагові критерії, намагаючись закодувати (тобто запам’ятати), що принесло користь, а що – ні.

Цей підхід дозволив AlphaZero виграти у трьох найкращих алгоритмів гри у шахи, го та сьоґі StockFish, AlphaGoZero та Elmo. Цю нейромережу можна зобразити наступним чином, як показано на рис. 1.2.

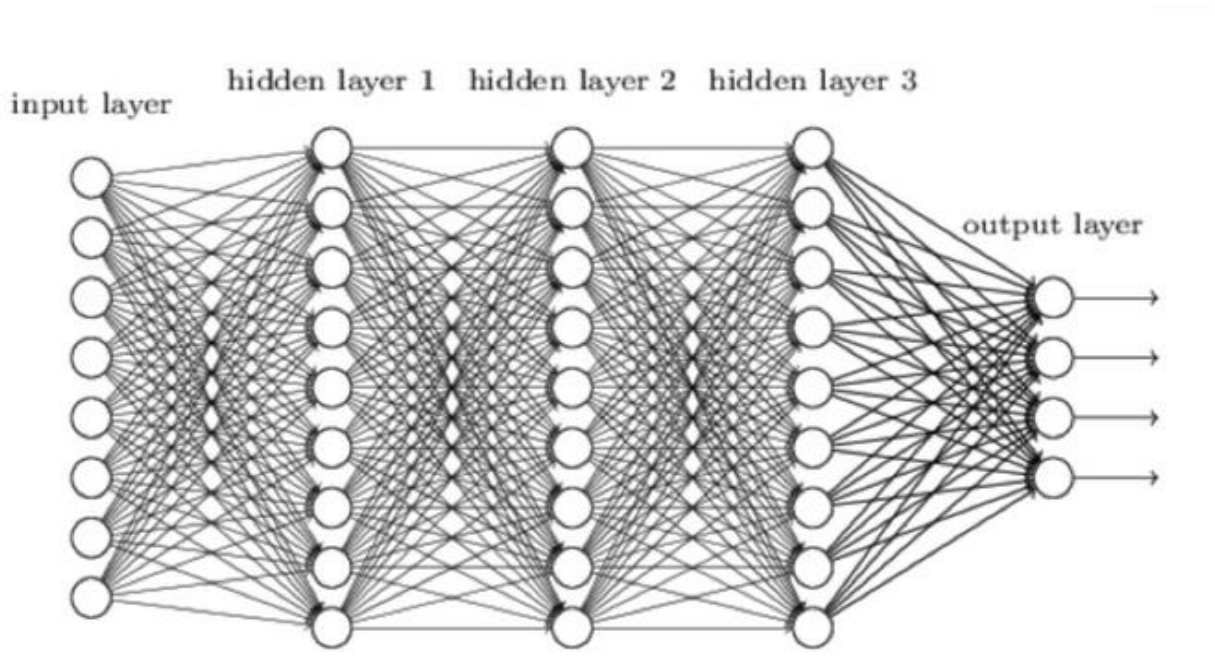

Рисунок 1.2 – Нейромережа AlphaZero

Але у цього алгоритма є 2 недоліки, AlphaZero потрібна чітка заданість правил гри і умов перемоги [12]. Більш того, якщо ми ціль неправильно визначена, ми не зможемо навчити модель так, як нам треба. Друга проблема – це умовно велика кількість необхідних ігор для тренування. Для тренування АльфаЗеро який використовувався у змаганнях з іншими алгоритмами було проведено 20–40 млн ігор.

Також одним із найпопулярнійших шахових ботів є Stockfish, його використовують у багатьох шахових програмах, онлайн-платформах таких як Lichess, Chess.com, а також для аналізу партій. Stockfish поєднує традиційні алгоритми пошуку з сучасними елементами штучного інтелекту [13].

Stockfish є відкритим шаховим рушієм, що базується на класичному алгоритмі мінімакса з альфа-бета відсіканням у поєднанні з глибокою системою евристик. Його основна перевага – надзвичайно висока продуктивність. Stockfish здійснює повний перебір мільйонів позицій на секунду, використовуючи впорядкування ходів, таблиці переходів (transposition tables), евристику «вбивчих ходів», а також відкриту нейромережеву оцінювальну функцію NNUE (Efficiently Updatable Neural Network).

У Stockfish пошук не «вчиться», рушій не тренується самостійно, натомість використовує вручну налаштовані параметри оцінювання, побудовані на шаховій експертизі. Він має вбудовані дебютні книги та ендшпільні таблиці, що забезпечує точну гру в певних класах позицій

З недоліків Stockfish можна виділити те, що ця модель на відміну від AlphaZero не самонавчається, тобто він не аналізує власні помилки чи успіхи а також він не завжди добре бачить позиційну ідею або довгострокову стратегічну перевагу, особливо в закритих позиціях.

Далі роглянемо алгоритм Leela Chess Zero. На відміну від Stockfish, Lc0 базується виключно на глибокій нейронній мережі, натренованій шляхом самостійної гри мільйонами партій. Її архітектура подібна до AlphaZero, однак

відкрита і доступна для дослідження. Рушій використовує алгоритм Монте-Карло з деревом пошуку (MCTS), який не виконує жорсткий перебір, а натомість вибирає статистично найперспективніші варіанти.

Перевага Lc0 – краще стратегічне бачення, особливо в позиціях, де класичні рушії часто оцінюють неточно (наприклад, у «тихих» стратегічних положеннях). Проте ця модель потребує GPU для ефективної роботи, має повільніший пошук, і глибина аналізу суттєво нижча, ніж у Stockfish [10].

Для порівняння цих ШІ зазвичай використовують шаховий рейтинг. Рейтинг Elo – це числова оцінка рівня шахіста (людини чи ШІ), яка показує, наскільки добре він грає в порівнянні з іншими. Його запропонував Арпад Ело, американський фізик угорського походження, у 1960-х роках.

Рейтинг використовується Міжнародною шаховою федерацією (FIDE), на популярних сайтах таких як Lichess та Chess.com або для оцінки ШІ (наприклад, Stockfish, AlphaZero)

Розрахунок базується на очікуваному результаті гри. Формула виглядає наступним чином:

$$Rnew = Rold + K * (W - E). \tag{1.1}$$

Тут Rold – поточний рейтинг гравця; Rnew – оновлений рейтинг; W – фактичний результат гри (1 = перемога, 0.5 = нічия, 0 = поразка); E – очікуваний результат (ймовірність виграти проти суперника з певним рейтингом); K – коефіцієнт чутливості (зазвичай 10–40, але вищий для новачків).

Очікуваний результат обчислюється за наступною формулою:

$$E = \frac{1}{1+10^{(Ropponent-Rplayer)/400}}. \tag{1.2}$$

Тут E – очікуваний результат (ймовірність виграшу гравця проти суперника з певним рейтингом). Значення в діапазоні від 0 до 1; Rplayer – поточний рейтинг гравця (того, чий очікуваний результат ми обчислюємо); Ropponent – рейтинг суперника; 10 – експоненційна складова, яка враховує різницю в рейтингах гравців.

У випадку шахових програм рейтинг визначається за результатами тисяч ігор проти інших рушіїв. Такі ігри зазвичай проводяться в рамках турнірів рушіїв, наприклад TCEC або CCRL, де кожна програма грає багато партій як білими, так і чорними. Завдяки великій кількості зіграних партій результати є статистично значущими. Рейтинги дозволяють об’єктивно порівнювати силу шахових рушіїв незалежно від апаратного забезпечення. Порівняємо Elo-рівнів шахових рушіїв (рис. 1.5).

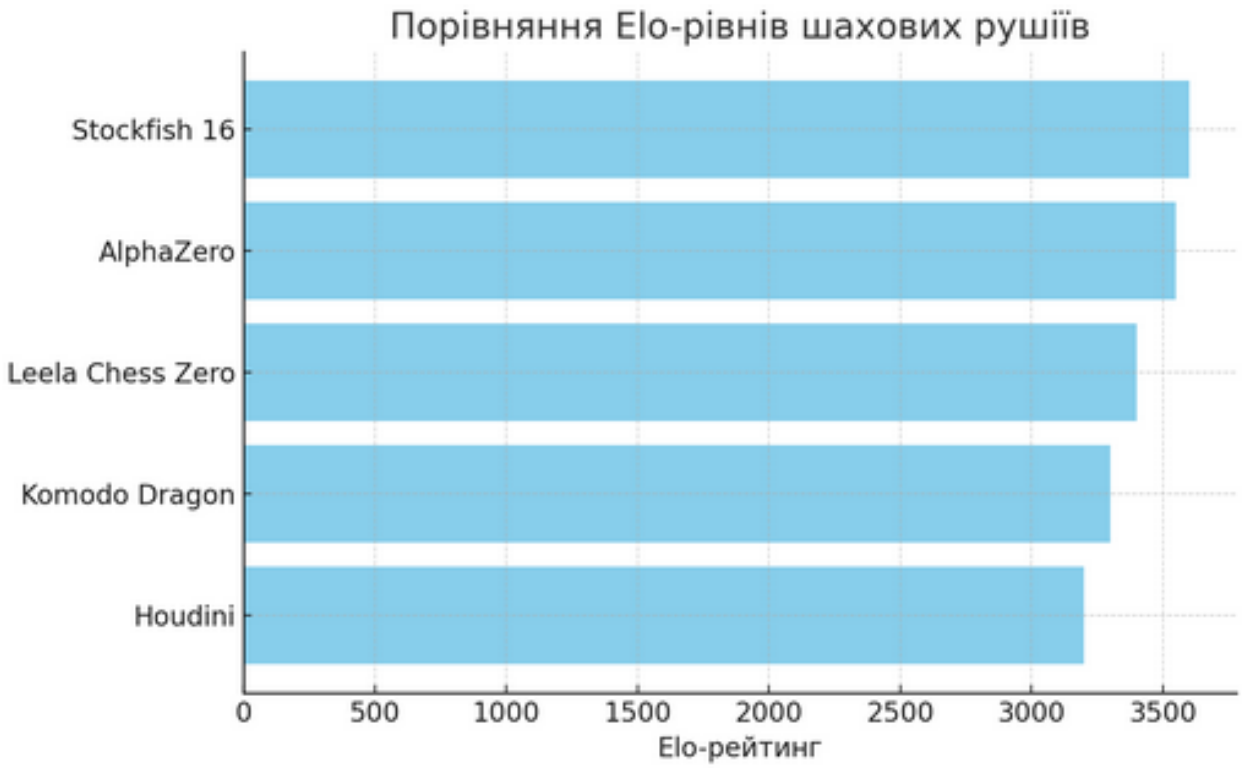

Рисунок 1.5 - Порівняння Elo-рівнів шахових рушіїв

Також можна оцінити шахові рушії за їх продуктивністю, для прикладу порівняємо найпростіший підхід, який аналізує всі можливі ходи на кожному етапі гри без будь-якої оптимізації, мінімакс, мінімакс з альфа-бета відсіканням та AlphaZero.

На рис 1.6 можна побачити, що найбільшу кількість позицій оцінює мінімакс з альфа-бета відсіканням, але сучасні ШІ такі як AlphaZero менше рахують, але грають краще завдяки інтелектуальному відбору ходів, що наближає їх до людського способу мислення.

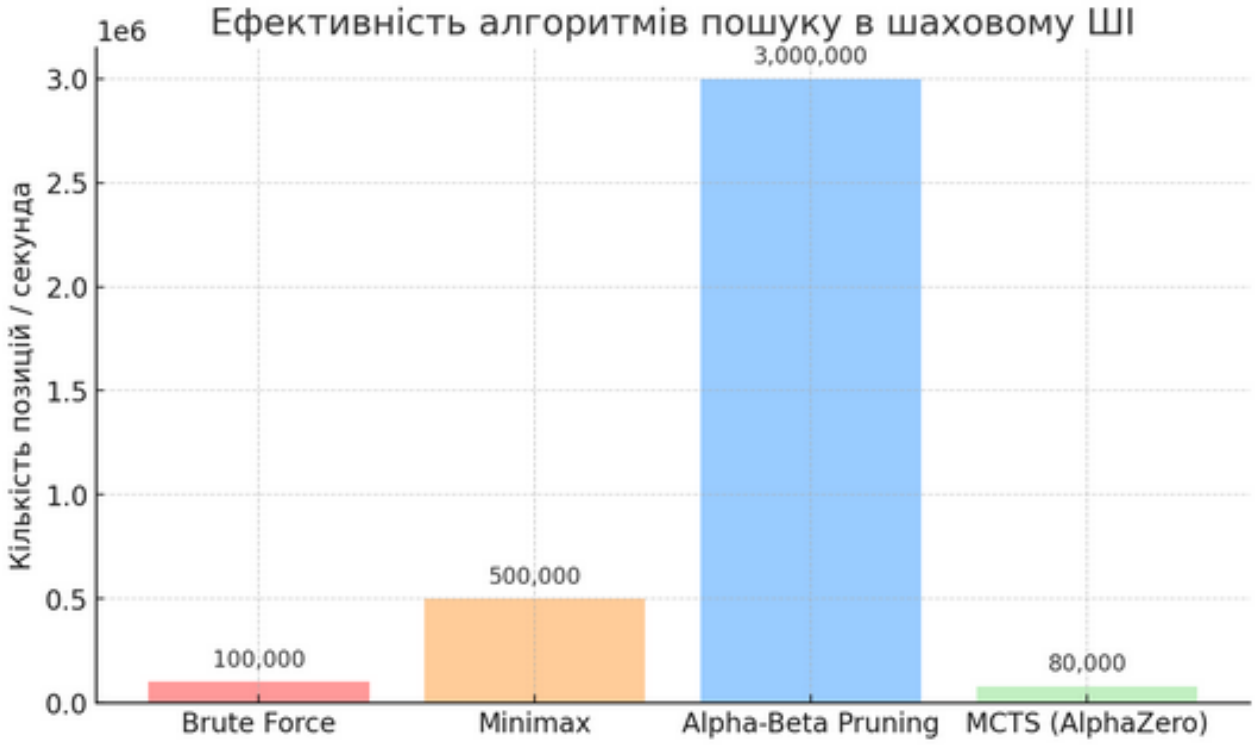

Рисунок 1.6 – Ефективність різних алгоритмів пошуку у шаховому ШІ

1.3 Оцінка позиції функцією оцінювання

Однією з ключових складових шахового штучного інтелекту є функція оцінювання позиції [14]. Ця функція визначає, наскільки вигідною є поточна позиція для одного з гравців, і використовується алгоритмами, такими як

мінімакс або α-β відсікання, для прийняття рішень. Якість функції оцінювання безпосередньо впливає на силу шахового рушія, тому її розробка є важливим етапом у створенні шахового ШІ. Розглянемо основні принципи побудови функцій оцінювання, їх компоненти та способи оптимізації.

Функція оцінювання призначена для кількісного визначення переваги однієї сторони в поточній позиції [14]. Вона враховує різні фактори, такі як матеріальний баланс, позиційні переваги, контроль над ключовими полями тощо. Результат функції – це числове значення, яке показує, наскільки вигідною є позиція для білих або чорних. Наприклад, позиція зі значенням +2.5 може вказувати на перевагу білих, тоді як значення -1.0 – на перевагу чорних.

Розглянемо деякі компоненти функції оцінювання.

Функція оцінювання зазвичай складається з декількох компонентів, кожен із яких враховує певний аспект позиції. Основні компоненти включають матеріальний баланс – це найпростіший і найважливіший компонент. Він враховує кількість і вартість фігур на дошці.

Стандартні значення фігур: пішак = 1, кінь = 3, слон = 3, тура = 5, ферзь = 9

Матеріальний баланс обчислюється як різниця між сумою значень фігур білих і чорних. Наприклад, якщо білі мають ферзя, а чорні – туру і слона, то матеріальний баланс буде +9 - (5 + 3) = +1.

Позиційні фактори, за якими ШІ оцінює перевагу того чи іншого гравця враховують розташування фігур і пішаків на дошці. Вони можуть включати такі моменти [15]:

– контроль над центром: фігури, розташовані в центрі дошки, зазвичай мають більший вплив на гру;

– безпека короля: позиція короля (наприклад, захищеність від шахів) є критично важливою;

– активність фігур: фігури, які контролюють багато полів або беруть участь у атаці, вважаються більш цінними;

– структура пішаків: міцна структура пішаків (наприклад, відсутність ізольованих або подвоєних пішаків) є перевагою.

Деякі функції оцінювання враховують додаткові параметри, такі як простір (кількість доступних полів для руху фігур) та темпи (швидкість розвитку фігур і пішаків).

Ендшпільні фактори також мають великий вплив бо у ендшпілі важливість короля зростає, тому його активність може бути додатково оцінена.

Розглянемо приклад спрощеної функції оцінювання, яка враховує матеріальний баланс і позиційні фактори [15].

Оцінка = Матеріальний баланс + Позиційні фактори.

У цій схемі поняття матеріальний баланс = (вартість фігур білих) - (вартість фігур чорних).

Також позиційні фактори = (контроль центру білих - контроль центру чорних) + (безпека короля білих - безпека короля чорних).

Наприклад, якщо білі мають матеріальну перевагу +1 і контролюють центр на 0.5 більше, а їх король безпечніший на 0.2, то загальна оцінка буде:

Оцінка = 1 + 0.5 + 0.2 = +1.7.

Приклад 2 (загроза мату): Білі загрожують матом у два ходи. Чорні можуть захиститися, але втратять фігуру.

Оцінка: +3.0 (значна перевага білих, але ще не виграна позиція).

1.4 Огляд інструментів комп’ютерної графіки

Під час розробки настільних ігор, таких як шахи, особливу увагу слід приділяти вибору інструментів комп’ютерної графіки, які відповідають функціональним вимогам та забезпечують належну продуктивність і зручність користувача [16-25]. У середовищі Java одним з найпоширеніших варіантів є

бібліотека Swing, яка надає базові можливості для створення віконного інтерфейсу.

У реалізації шахів за допомогою Swing шахова дошка часто створюється як сітка з 64 панелей (наприклад, JPanel), кожна з яких змінює свій колір залежно від координат. Фігури зазвичай додаються як зображення (через ImageIcon або безпосередньо через Graphics.drawImage(...)), або ж малюються вручну у методі paintComponent(Graphics g). Цей підхід дозволяє повністю контролювати логіку відображення, але при цьому потребує великої кількості «ручної» роботи.

На рис 1.7 показані всі доступні нам компоненти Swing у Java які ми можемо використати для сторення програми.

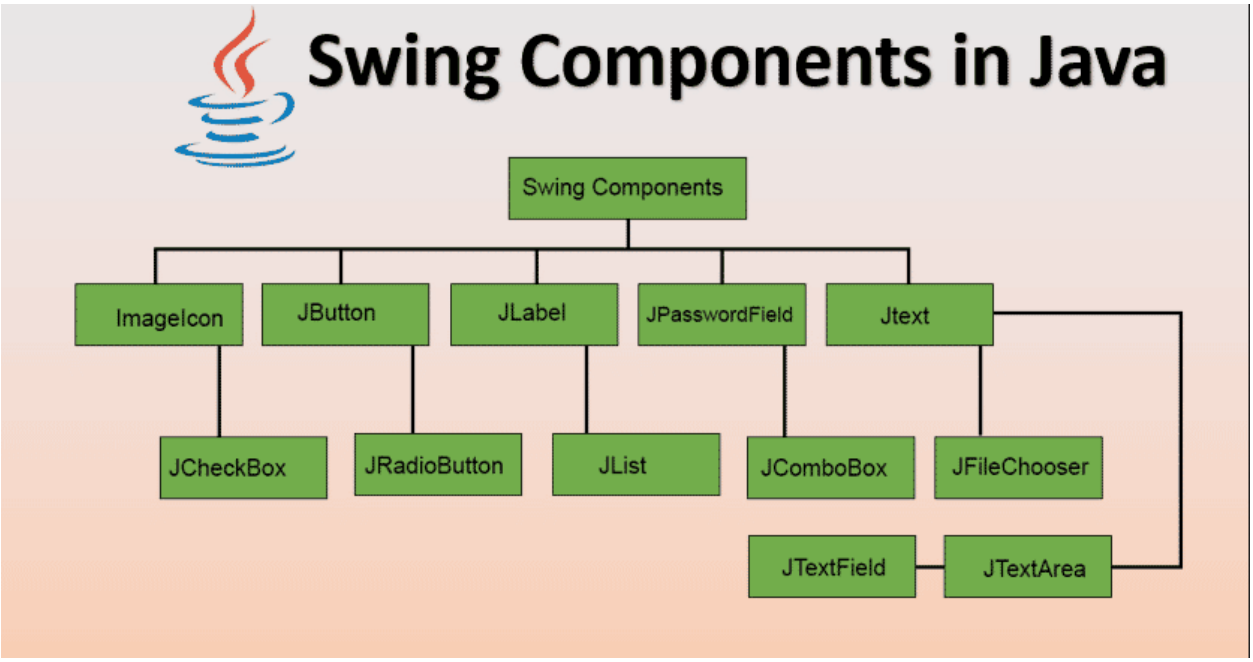

Рисунок 1.7 – Компоненти Java Swing [26]

У той же час сучасніша альтернатива – JavaFX [26] – пропонує більш гнучкий механізм побудови інтерфейсу. Тут шахову дошку можна реалізувати за допомогою сітки (GridPane), а фігури – як елементи ImageView, які можна стилізувати та трансформувати. Завдяки підтримці CSS, JavaFX дозволяє швидко змінювати зовнішній вигляд елементів, а вбудовані засоби анімації

(класи TranslateTransition, FadeTransition тощо) дають змогу створювати плавні переміщення фігур. Наприклад, при здійсненні ходу фігури можна не просто змінити її позицію, а й додати анімацію переміщення з однієї клітини на іншу. Це значно підвищує візуальну якість інтерфейсу.

Для проєктів, де важлива швидка візуалізація або творче експериментування з графікою, використовують середовище Processing. Це бібліотека, яка дозволяє створювати інтерактивну графіку з мінімальним обсягом коду. Наприклад, шахову дошку можна побудувати простим циклом for, який малює квадрати через rect(x, y, size, size), а фігури – через image() або за допомогою векторної графіки (наприклад, примітивів ellipse, line, triangle тощо). Processing більше підходить для візуальних демонстрацій або освітніх проєктів, оскільки не має повноцінної системи керування подіями і не призначений для великих GUI-додатків [26-33].

Коли ж ідеться про складнішу графіку або необхідність у високій продуктивності, доцільним є використання OpenGL – низькорівневої графічної бібліотеки, що дозволяє безпосередньо працювати з відеокартою. У Java її можна підключити через JOGL або LWJGL. Наприклад, шахова дошка може бути представлена у вигляді текстурованої площини, а кожна фігура – у вигляді двовимірного спрайта або навіть трьохвимірної моделі. Перевага OpenGL у тому, що вона дозволяє створювати ефекти освітлення, тіней, обертання камери, масштабування сцени тощо, однак її використання потребує значно більше знань з комп’ютерної графіки та математики. Рендеринг фігур тут реалізується вручну через шейдери та буфери, що забезпечує повний контроль над усім графічним конвеєром.

Окремий клас рішень становлять ігрові рушії, такі як Unity чи Unreal Engine. Їх варто розглядати як альтернативу у разі переходу на інші мови програмування (наприклад, C# для Unity). Завдяки потужним інструментам побудови сцен, фізики, анімацій та взаємодії, рушії дозволяють реалізувати шахи у трьохвимірному форматі з реалістичним освітленням, анімованими

ходами фігур, музичним супроводом та мережевим режимом гри. Візуальні компоненти в Unity створюються за допомогою Canvas та UI-елементів, тоді як сцена може містити трьохвимірні моделі фігур, які реагують на події миші або сенсора.

Вибір графічного інструменту безпосередньо впливає на структуру, функціональність та зовнішній вигляд гри. Для класичних реалізацій у межах Java найдоцільніше використовувати Swing або JavaFX, які забезпечують достатню функціональність при помірній складності. У той же час, якщо мета створити візуально насичений, анімаційний або трьохвимірний інтерфейс, слід звернути увагу на OpenGL або ігрові рушії, що дають ширші можливості, але вимагають більш глибокої технічної підготовки.

1.5 Постановка задачі

Метою роботи є реалізація шахового штучного інтелекту, що базується на алгоритмі мінімаксу, та тестування ефективності різних підходів до оцінки позицій. Зокрема, досліджується вплив глибини пошуку на якість гри бота, швидкість обчислень та точність вибору оптимального ходу.

Для досягнення мети необхідно вирішити наступні завдання:

– розробити структуру даних для представлення шахової дошки розміром 8×8 клітинок, описати та реалізувати всі типи фігур: пішак, кінь, слон, тура, ферзь, король. Реалізувати правила гри, включаючи можливі ходи для кожної фігури;

– створити функцію, яка повертає числову оцінку поточного стану гри з точки зору гравця, врахувати матеріальний баланс фігур, а також такі позиційні фактори як контроль центра, захищеність і безпека короля, активність фігур, розвиток у дебюті та структура пішаків;

– реалізувати алгоритму мінімакса, який аналізує дерево гри і вибирає оптимальний хід, враховуючи можливі дії противника. Вибрати відповідну

глибину пошуку (наприклад, 2–3 півходи) для балансу між продуктивністю й точністю;

– розробити інтерфейс для взаємодії з користувачем: реалізувати простий інтерфейс для гри проти бота. Забезпечити можливість введення ходів користувачем та відображення ходів бота;

– протестувати бота на різних позиціях, включаючи дебюти, мітельшпілі та ендшпілі. Визначити слабкі сторони бота такі як обмежена глибина пошуку, неефективна оцінка деяких позицій і покращити функцію оцінювання шляхом додавання нових факторів.

2 МОДЕЛІ ПРИЙНЯТТЯ РІШЕНЬ У ШАХОВІЙ ГРІ

2.1 Суть мінімаксного підходу

Мінімаксний підхід є важливим методом в теорії прийняття рішень, оптимізації та управлінні, особливо в умовах невизначеності та конфлікту. Він базується на принципі мінімізації максимальних можливих втрат, що дозволяє знаходити оптимальні рішення в найгірших сценаріях.

Теорема мінімаксу стверджує, що для кожної гри для двох осіб з нульовою сумою та скінченним числом стратегій, існує таке значення V та змішані стратегії для кожного гравця, такі, що (а) Для будь-якої стратегії Гравця 2, Гравець 1 може гарантувати собі виграш V, і (б) Для будь-якої стратегії Гравця 1, Гравець 2 може гарантувати собі виграш (-V) [6].

Це рівнозначно тому, що стратегія Гравця 1 гарантує його виграш V, незалежно від стратегії гравця 2, а також що Гравець 2 теж може гарантувати собі виграш -V. Назва алгоритм мінімаксу виникла тому, що кожен гравець мінімізує максимально можливу винагороду для іншого гравця, так як гра з нульовою сумою, він також максимізує свою винагороду.

Приклад гри з нульовою сумою, де А і Б роблять ходи, показує рішення мінімаксу (рис. 2.1). Припустимо, що у кожного гравця є три варіанти, і розглянемо матрицю для А, яка відображена справа. Припустимо, що платіжна матриця для Б така ж матриця зі зворотним знаком (наприклад, якщо вибір А1 і Б1, то Б платить А 3). Тоді вибір мінімаксу А є А2 з найгіршим результатом - платити 1, у той час як простий вибір мінімаксу для Б є Б2 з найгіршим результатом – нічого не платити.

	Б обирає Б1	Б обирає Б2	Б обирає Б3
А обирає А1	+3	–2	+2
А обирає А2	–1	0	+4
А обирає А3	–4	–3	+1

Рисунок 2.1 – Мінімаксні стратегії

Однак це рішення не є стійким, тому що якщо Б вважає, що А вибере А2, то Б вибере Б1, щоб отримати 1, а якщо А вважає, що Б вибере Б1, то А вибере А1, щоб отримати 3, тому Б вибере Б2, і врешті-решт обидва гравці усвідомлюють труднощі зробити вибір. Таким чином, потрібна більш стабільна стратегія.

Деякі варіанти гірші за інші, і можуть бути усунені: А не буде обирати А3, тому що А1 або А2 дають кращий результат, незалежно від того, що обере Б; Б не буде вибирати Б3, так як і Б1, і Б2 дають кращі результати, незалежно від того, що обере А.

А може уникнути того, щоб робити очікувані виплати більш ніж на 1/3 обираючи А1 з імовірністю 1/6 і А2 з імовірністю 5/6, незалежно від того, що вибирає Б. Б може забезпечити очікуваний приріст, принаймні на 1/3 за допомогою випадкової стратегії вибору Б1 з імовірністю 1/3 і Б2 з імовірністю 2/3, незалежно від того, що вибирає А. Ці мінімаксні стратегії тепер є стабільними і не можуть бути поліпшені.

Простий алгоритм мінімаксу може бути тривіально змінений, щоб додатково повертати саму стратегію разом з результатом мінімаксу.

Лістинг 2.1 Псевдокод алгоритму мінімакс:

```
function integer minimax(node, depth)
    if node is a terminal node or depth <= 0:
        return the heuristic value of node
    α = -∞
    for child in node:                    # evaluation is identical for both players
        α = max(α, -minimax(child, depth-1))
    return α
```

Розглянемо ще один приклад [34]. Припустимо, що в грі є максимум два можливих кроки для кожного гравця на кожен хід. Алгоритм генерує дерево праворуч, де кола є ходи максимізуючого гравця, а квадрати являють собою ходи суперника (мінімізуючий гравець). Через обмеженість обчислювальних ресурсів, як описано вище, дерево обмежено на 4 кроки уперед.

Алгоритм оцінює кожен вузол за допомогою евристичних функцій оцінки, отримані значення показано на малюнку. Ходи, де максимізуючий гравець виграє, оцінені як додатна нескінченність, в той час як кроки, які приведуть до перемоги мінімізуючого гравця, оцінені як від'ємна нескінченність. На рівні 3 алгоритм буде вибирати, для кожного вузла, найменше зі значень його дочірніх вузлів, і призначить це значення тому ж вузлу (наприклад, вузол зліва буде вибирати мінімальне з 10 і «+ ∞», тому призначить собі значення 10).

Наступний крок, на рівні 2, полягає у виборі найбільшого значення для кожного вузла серед його дочірніх вузлів. Знову ж таки, кожне значення присвоюється кожному батьківському вузлу. Алгоритм продовжує оцінки максимального і мінімального значень дочірніх вузлів по черзі, аж поки не досягне кореневого вузла, де він вибирає хід з найбільшим значенням (представлено на малюнку синьою стрілкою). Це крок, який гравець повинен зробити для того, щоб звести до мінімуму максимально можливої втрати.

2.2 Роль глибини пошуку в прийнятті рішень

Роль глибини пошуку в прийнятті рішень у шахах є одним із найважливіших факторів у шаховому аналізі. Глибина пошуку визначає, наскільки далеко вперед шаховий двигун або гравець може прорахувати варіанти [35].

Глибина пошуку (або «глибина дерева варіантів») означає кількість ходів, які аналізуються наперед [35, 36].

Глибина 1 – аналізуються тільки можливі відповіді після одного ходу.

Глибина 2 – аналізується відповідь суперника після кожного можливого ходу.

Глибина 10 – прораховується послідовність 10 ходів для обох сторін.

Глибина може бути різною для різних шахових програм:

Людина зазвичай бачить 3-5 ходів вперед (у складних позиціях менше). Гросмейстери можуть вираховувати 10-15 ходів у форсованих варіантах. Stockfish, Lc0, AlphaZero прораховують 30-50 ходів у деяких позиціях [36].

Невелика глибина (1-3 ходи) – використовується для швидких рішень, Може не побачити далекоглядних комбінацій. Наприклад, шаховий рушій на 2-й глибині може вибрати хід, який виглядає хорошим, але не помітити загрозу через 4 ходи.

Середня глибина (4-10 ходів) – більш точний аналіз варіантів, враховує короткострокові тактичні загрози, використовується для аналізу середньої складності позицій.

Велика глибина (10-30 ходів) – дозволяє оцінювати позицію стратегічно. Використовується для пошуку складних комбінацій. Дає точну оцінку ендшпільних позицій.

Приклад: уявімо позицію, де можливий жертвенний хід, що через 5 ходів призведе до виграшу.

На глибині 3 програма не побачить виграш і відмовляється від жертви, а на глибині 7 програма розраховує матову атаку та правильно жертвує матеріал.

Найпопулярнішими алгоритмами зараз є:

– мінімакс – базовий алгоритм: гравець намагається максимізувати свій результат, а суперник – мінімізувати. Чим глибший пошук, тим точніше оцінюється позиція;

– альфа-бета відсікання – оптимізований мінімакс: відсікає гілки дерева варіантів, які не мають сенсу розглядати. Значно прискорює пошук, дозволяючи аналізувати більше ходів за менший час;

– Монте-Карло (нейромережі: alphaZero, lc0) – використовує статистичний аналіз можливих партій. Однією з важливих переваг є те що цей алгоритм може глибоко оцінювати позицію навіть без точного прорахунку всіх варіантів [37].

Головне обмеження глибини пошуку – це обчислювальна складність, бо кількість можливих ходів у шахах росте експоненційно. Вже після 3 ходів це – ($\sim 10^3$ варіантів). Після 6 ходів ($\sim 10^6$ варіантів), а після 10 ходів – мільярди можливостей. Тому навіть суперкомп’ютери не можуть аналізувати всі можливі варіанти на великій глибині.

Замість глибокого аналізу часто використовуються евристичні алгоритми такі як таблиці ендшпілю (повні бази даних ідеальних ходів у кінцівках) чи шахові патерни (досвідчена оцінка позиції на основі принципів).

З цього ми маємо, що глибший пошук відповідає точнішій оцінці позиції. Занадто мала глибина призводить до помилкових рішень, а доповнення альфа-бета відсікання дозволяють швидше обчислювати варіанти.

У нашій роботі ми будемо використовувати глибину пошуку 2 для забезпечення швидкодії нашої програми.

2.3 Розширення алгоритму за допомогою евристик

Шахові рушії використовують евристики для прискорення пошуку та покращення оцінки позицій. Евристики допомагають відсіювати погані ходи та зосереджуватися на найперспективніших варіантах [38].

Евристика – це правило, яке допомагає знаходити хороші ходи швидше, ніж повний перебір варіантів. В шахах евристики дозволяють пріоритезувати кращі ходи та відкидати завідомо слабкі варіанти, Скорочувати час аналізу без втрати якості гри.

Евристики працюють разом із алгоритмами пошуку (наприклад, Мінімакс, Альфа-Бета Відсікання).

Основні евристики у шахових рушіях включають у себе:

– принцип «Перший хороший хід» (First Good Move). Замість повного перебору рушій перевіряє ходи за пріоритетом. Якщо знаходить хороший варіант, припиняє пошук глибше. Прискорює аналіз, але іноді пропускає кращі варіанти; наприклад якщо є хід, який дає матеріальну перевагу, рушій його виконає, не шукаючи інших;

– впорядкування ходів (Move Ordering). Рушій аналізує спочатку найкращі ходи, а потім менш важливі. Це підвищує ефективність альфа-бета відсікання.

Типовими пріоритетами ходів є шахи (особливо з жертвами, що ведуть до атаки), наступним пріориітетом є взяття фігур (особливо, якщо немає компенсації для суперника), далі загрози (наприклад, напад на ферзя) та розвиток фігур і стратегічні ходи [38].

Наприклад рушій спершу розглядає хід, що дає мат або виграє ферзя, і лише потім аналізує ходи, що дають дрібну позиційну перевагу.

– евристика «Вбивчих ходів» (Killer Moves). Якщо певний хід раніше виявився ефективним, рушій перевіряє його першим в інших варіантах. Часто працює у складних тактичних позиціях.

Наприклад якщо в одній варіації жертва слона призвела до матової атаки, рушій розглядає подібні жертви в інших позиціях.

– принцип «Жертва та виграш» (SEE – Static Exchange Evaluation). Аналізує, чи вигідне взяття фігури на конкретному полі. Визначає, чи втратить гравець більше матеріалу після взяття.

Наприклад якщо можна взяти туру, але суперник одразу забирає ферзя у відповідь, рушій відмовиться від цього варіанту.

– евристика «Ітеративне заглиблення» (Iterative Deepening). Спочатку аналізує позицію на малій глибині, потім поступово заглиблюється. Використовується для обмеження часу пошуку.

Наприклад спочатку рушій аналізує варіанти на 4 ходи вперед, потім на 6, 8 і т.д.

Евристики зменшують кількість обчислень – рушій не прораховує всі ходи, а лише найважливіші, прискорюють пошук – завдяки впорядкуванню ходів можна швидше знаходити сильні ходи. Підвищують точність – рушій уникає «дурних» ходів і краще оцінює позицію. Підсилюють стратегію – враховують не лише тактику, а й довгострокові фактори [38].

Також для покращення ходів ШІ часто використовуються бази даних [39]. Наприклад для ситуацій в яких кількість ходів обмежена, а найкращій хід для цієї позиції вже є в базі даних ШІ може просто взяти цей хід з бази, це пришвидшить хід та покращіть вибір.

На рисунку 2.2 вказан приклад типового інтерфейсу для використання бази даних ендшпілю. Для кожного ходу білих таблиці показують число ходів до виграшу. Походивши Кс6 чи Фа6+, вони виграють за 5 ходів, отже, це оптимальні ходи.

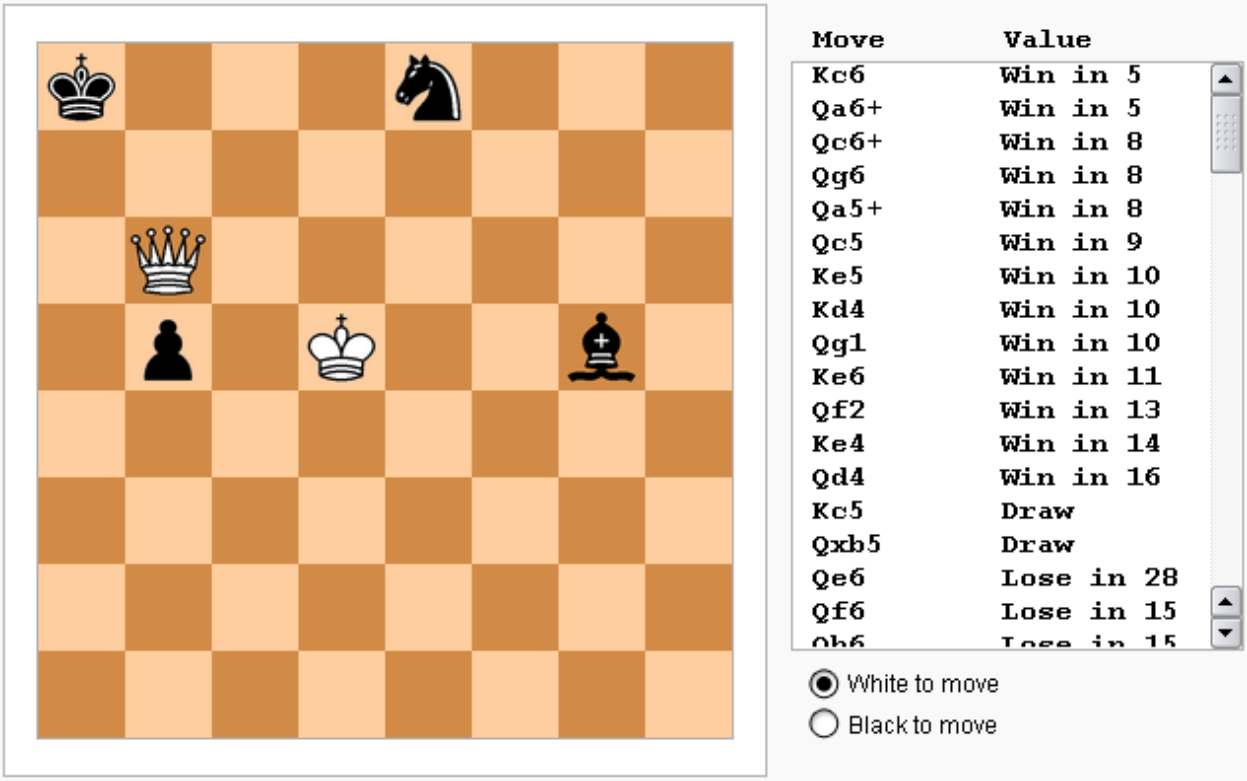

Рисунок 2.2 – Бази даних ендшпілю

2.4 Моделювання системи

Для забезпечення ефективного планування та реалізації архітектури шахового штучного інтелекту було використано UML-діаграму класів, що є частиною комп’ютерного моделювання. UML (Unified Modeling Language) – це уніфікована мова моделювання, яка дозволяє наочно представити структуру програмної системи, зв’язки між об’єктами, їх атрибути та поведінку. UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем [40].

При модифікації системи об’єктний підхід дозволяє легко включати в систему нові об’єкти і виключати застарілі без істотної зміни її

життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об’єктів.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код (рис 2.3).

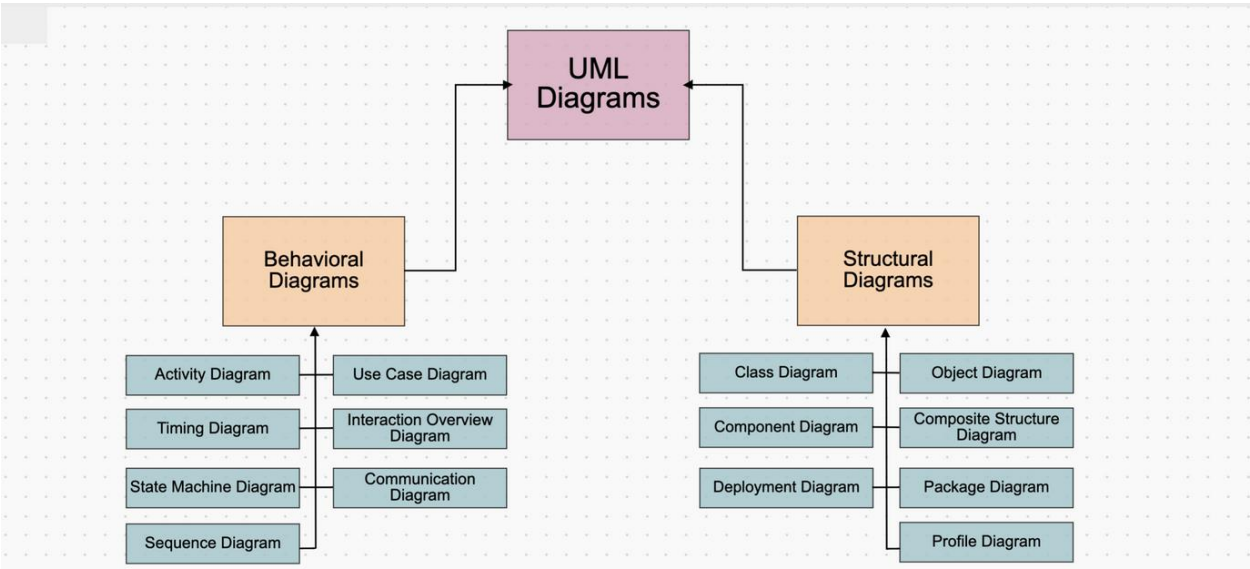

Рисунок 2.3 – Види UML діаграм

Діаграма класів, наведена нижче на рисунку 2.4, демонструє основні компоненти системи гри в шахи, включаючи логіку дошки, фігур, управління, а також механізм оцінки ходів та штучного інтелекту.

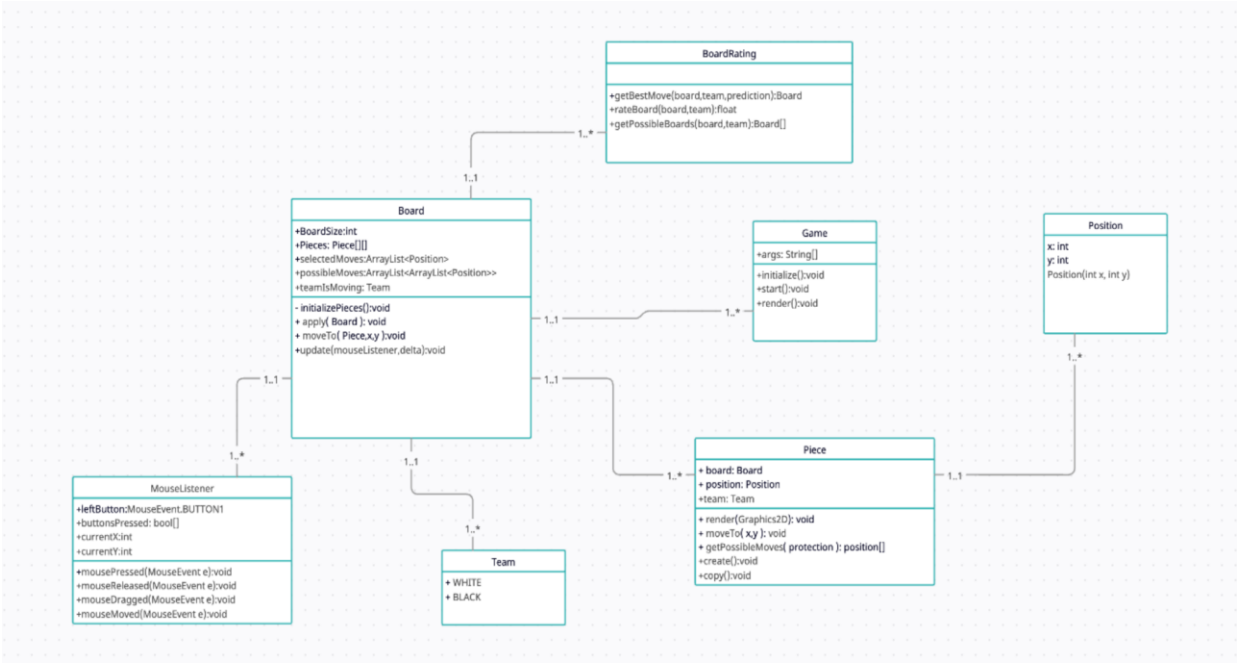

Рисунок 2.4 – UML діаграма класів

Основні компоненти UML-діаграми включають в себе такі класи.

Клас Board відповідає за збереження стану шахової дошки, містить методи для ініціалізації, переміщення фігур та оновлення стану. Він пов’язаний з фігурами (Piece) та позиціями (Position), а також інтегрується з обробкою миші (MouseListener) для взаємодії користувача.

Клас Piece представляє шахову фігуру, зберігає її позицію та команду (Team). Містить методи для переміщення, рендерингу та визначення можливих ходів через getPossibleMoves().

Клас BoardRating реалізує логіку штучного інтелекту, відповідає за оцінку поточного стану гри (rateBoard) і генерує можливі варіанти розвитку гри (getPossibleBoards). Саме цей клас бере участь у виборі найкращого ходу (getBestMove).

Клас Game виступає як точка входу в програму, відповідає за ініціалізацію, запуск та рендеринг гри.

Клас MouseListener реалізує обробку подій миші, необхідну для інтерактивного керування фігурами на дошці.

Клас Position інкапсулює координати фігур на дошці (x, y), а клас Team визначає команду (чорні або білі фігури).

UML-діаграма класів дозволила чітко визначити структуру системи до початку реалізації; виділити залежності між компонентами, зокрема зв’язки між дошкою, фігурами, позиціями та обробкою вводу. Проєктувати логіку штучного інтелекту в контексті загальної архітектури; полегшити супровід та розширення системи в майбутньому. Служити документацією, що полегшує розуміння проекту іншими розробниками.

UML-моделювання стало невід’ємною частиною етапу проектування системи, допомогло виявити потенційні проблеми на ранньому етапі.

2.5 Огляд інструментарію для створення застосунку

Для реалізації шахової гри з використанням штучного інтелекту було обрано перевірений стек технологій, який забезпечує ефективну розробку, зручну підтримку коду та легке розгортання. Нижче наведено опис основних інструментів та принципів ООП, використаних у проекті.

IntelliJ IDEA – потужна інтегрована середа розробки для Java, що надає зручні засоби для написання коду, рефакторингу, налагодження та тестування. Інтеграція з Git дозволяє легко вести контроль версій, а вбудований Swing UI Designer спрощує побудову графічного інтерфейсу.

Java – основна мова реалізації, яка завдяки платформонезалежності та багатому набору стандартних бібліотек дозволяє створювати складні об’єктні моделі гри, обчислювальні алгоритми для бота та графіку через Swing і java.awt.

Maven – інструмент для автоматизованої збірки, який структурує проєкт у стандартизовані каталоги (src/main/java, src/test/java, resources) та керує залежностями через файл pom.xml. Maven забезпечує відтворюваність збірки та простоту підключення бібліотек.

Lombok – бібліотека для генерації шаблонного коду за допомогою анотацій (*@Data*, *@Getter*, *@Setter*, *@NoArgsConstructor*, *@AllArgsConstructor*). Це дозволяє зосередитися на бізнес-логіці, позбавившися ручного написання гетерів, сетерів і методів toString(), equals(), hashCode().

У проєкті також використовуються ключові принципи ООП, що забезпечують гнучкість, розширюваність та підтримуваність коду.

Абстракція дозволяє виділяти сутності предметної області (шахова дошка, фігура, позиція) у вигляді окремих класів з чітко визначеними інтерфейсами. Наприклад, клас Piece інкапсулює загальні методи getPossibleMoves(), а конкретні фігури (Pawn, Queen) наслідують цей інтерфейс і реалізують лише специфічні алгоритми руху.

Інкапсуляція полягає в приховуванні внутрішнього стану об’єкта та наданні доступу через публічні методи. У класі Board стан дошки (двовимірний масив pieces[][]) є приватним, а для взаємодії з ним використовуються методи setPiece(), getPiece(), що гарантує контроль цілісності даних.

Наслідування дозволяє створювати ієрархію класів: базовий клас DefaultPiece містить загальні атрибути та поведінку, тоді як підкласи (King, Rook, Bishop тощо) успадковують логіку та розширюють її власними правилами. Це зменшує дублювання коду та спрощує підтримку.

Поліморфізм дає змогу взаємодіяти з об’єктами різних класів через один інтерфейс. Наприклад, методи пошуку ходів викликаються через посилання типу DefaultPiece, і фактична реалізація getPossibleMoves() буде викликатися динамічно, залежно від конкретного об’єкта-фігури.

Також були використані такі правила проектування як принцип єдиної відповідальності (SRP) – кожен клас виконує лише одну чітко визначену роль: Board відповідає за стан гри, BoardRating – за оцінювання позиції, Game – за запуск і цикл подій, а MouseListener – за обробку користувацького вводу.

Відкритість/закритість (OCP) – класи відкриті для розширення (додавання нових фігур чи алгоритмів оцінювання), але закриті для модифікації.

Нові алгоритми оцінки можна впроваджувати, створивши новий клас, що реалізує інтерфейс BoardEvaluator, без зміни існуючого коду, та dependency injection – для підключення компонентів (наприклад, схеми евристик або стратегії пошуку) використовуються конструктори з параметрами або фабрики, що спрощує модульне тестування і зміну реалізації шляхом передачі інших реалізацій через параметри.

Таким чином, комбінування сучасних інструментів розробки (IntelliJ IDEA, Java, Maven, Lombok) з дотриманням принципів ООП забезпечило чітку структуру проєкту, повторне використання коду, легкість тестування та подальшого розширення функціональності.

3 РЕЗУЛЬТАТИ КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ

3.1 Створення проекту

Для розробки програмного забезпечення зі створення шахового бота було обрано середовище розробки під назвою IntelliJ Idea, яке чудово підходить для розробки на мові програмування Java.

Середовищє розробки IntelliJ IDEA також підтримує такі корисні інструменти для проведення тестування і розробки як Junit, засоб складання Maven.

До складу також входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів а також підтримує технології Java EE, UML-діаграм, та підрахунок покриття коду.

Для візуалізації нашого проекту будемо використовувати бібліотеку javax.swing яка надасть нам всі необхідні компоненти візуалізації та java.awt.event для інтеграції з мишкою.

Спочатку треба знайти чорно-біле зображення усіх шахових фігур, та заімпортувати файл з фігурами у проект (рис 3.1).

Рисунок 3.1 - Зображення шахових фігур у форматі png

Далі за допомогою бібліотеки swing та її компонента JFrame створимо поле для гри 512x512 і поділимо поле на рівні клітини 8x8 (рис. 3.2).

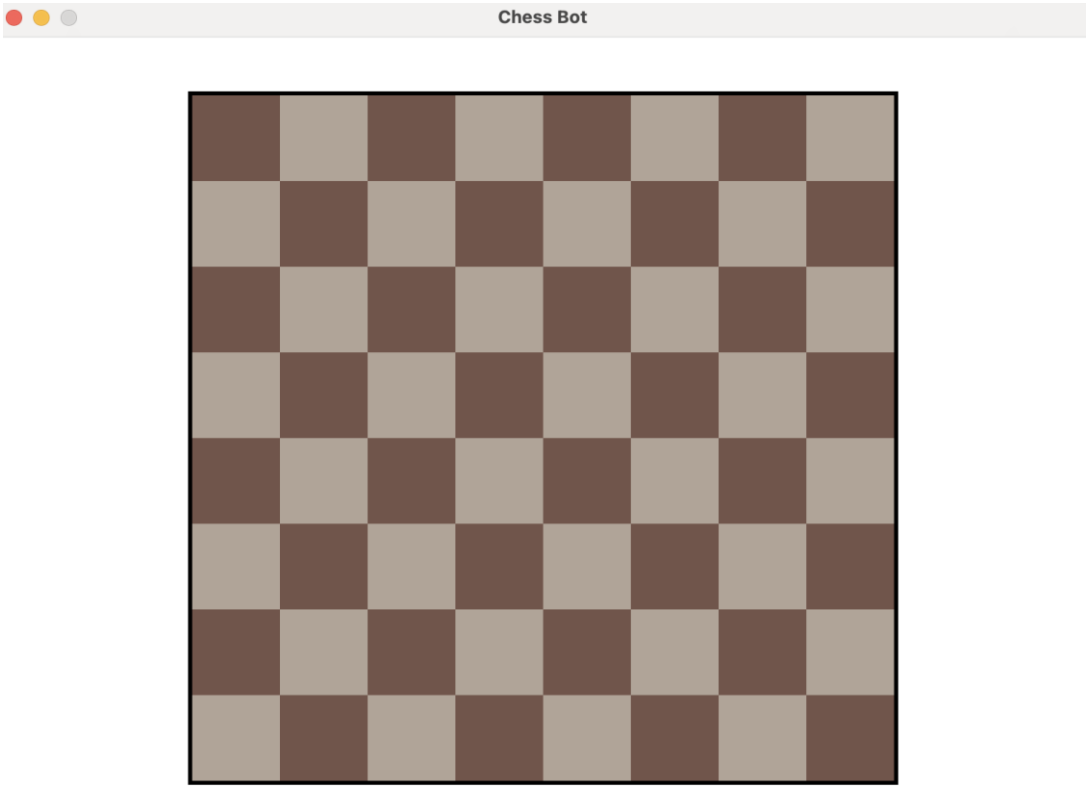

Рисунок 3.2 – Розфарбоване поле для гри

Після цього пройдемось по всіх клітинах та додамо коричневий колір парним клітинам та білий не парним:

Лістинг 3.1 Реалізація зозфарбовки поля:

```
for (int x = 0; x < pieces.length; x++) {
  for (int y = 0; y < pieces[0].length; y++) {
  g.setColor(((x + y) % 2 == 0) ? new Color(0x755449) : new Color(0xB3A396));
  g.fillRect(boardX + x * cellWidth, boardY + y * cellHeight, cellWidth, cellHeight);
```

```
    }
}
```

Далі додамо фігури, для цього фикористаємо до цього завантажений png файл та біблітоку ImageIO.

Лістинг 3.2 Реалізація відображення фігур на UI:

```
piecesSpriteSheet = ImageIO.read(Objects.requireNonNull(Board.class.getResourceAsStream("/pieces.png")));
```

Для кожної фігури створимо свої класи, які будуть наслідуватися від базового класу фігури, та зв’яжемо ці класи з зображеннями. У кожного класа буде конструктор з полями Board та Team що би при створеннні об’єкта ми змогли вказати команду та місцезнаходження фігури. Тепер розставимо фігури по дошці, для цього ми повинні вказати координати x та y для кожної фігури, а також команду гравця.

Наприклад для розміщення на дошці ферзя ми повинні створити об’єкт Queen вказати її команду та координати (у цьому випадку x координата буде дорівнювати 4, а y буде рівним 7 для чорних, для білих x буде 4, а y 0.

Лістинг 3.3 Реалізація розміщення фігур на дошці:

```
setPiece(new Queen(this, Team.BLACK), 4, 7);
setPiece(new Queen(this, Team.WHITE), 4, 0);
```

На рисунку 3.3 показан результат розмыщення ферзів на дошці.

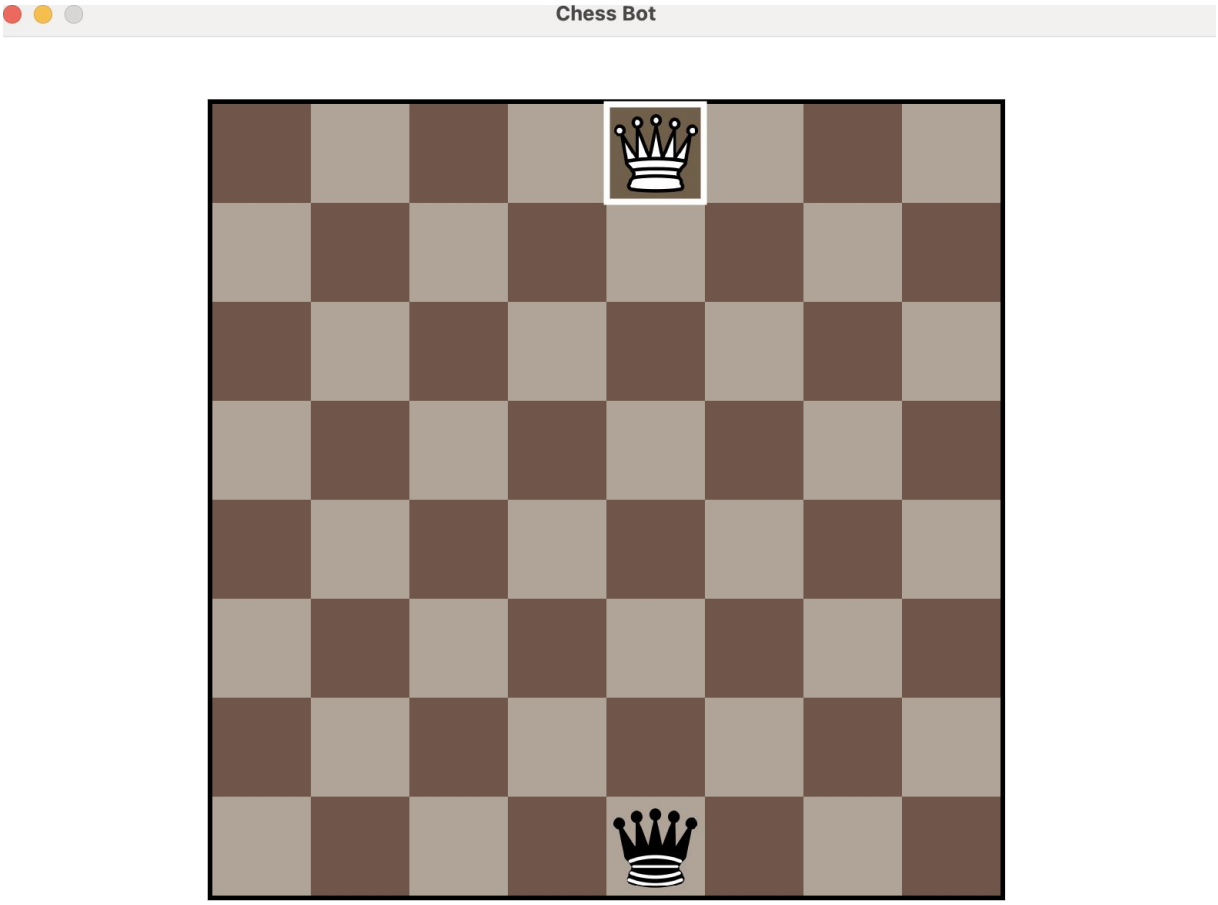

Рисунок 3.3 – Розстановка ферзів

Зробимо це для всіх фігур (коней, слонів, королей, тур). Для створення пішаков використаємо массив, щоб не писати однаковий код вісім разів для кожної команди.

Лістинг 3.4 Створення пішаков використовуючи массиви:

```
for (int x= 0; x < pieces[0].length; x++) {
  setPiece(new Pawn(this, Team.BLACK), x, 6);
  setPiece(new Pawn(this, Team.WHITE), x, 1);
}
```

Для кращої візуалізації також додамо салатове підсвічування для команди яка робить хід. Для цього додамо перевірку яка буде перевіряти яка команда зараз робить хід.

Лістинг 3.5 Реалізація підсвічування:

```
if (getTeam() == board.getTeamIsMoving()) {
  g.setColor(new Color(0x1040ff40, true));
  g.fillRect(x, y, width + 1, height + 1);
}
```

Вигляд поля після цих змін наведено на рис 3.4.

Рисунок 3.4 – Повний вигляд поля

Далі для можливості контактувати з фігурами під’єднаємо бібліотеку java.awt.event, яка буде відслідковувати рух та використання миші.Для цього потрібно розширити інтерфейс нашого класу за допомогою implements.

Лістинг 3.6 Реалізація використання мищі:

public class MouseListener implements java.awt.event.MouseListener, MouseMotionListener

Це дасть нам можливість переписати необхідні нам методи для реалізації поведінки миші, такі як mousePressed, mouseReleased, mouseDragged та mouseMoved. У цих методах ми будемо слідкувати за положенням миші та давати знати коду коли ми натискаємо на фігуру, або клікаємо на клітку поля для того щоб перемістити її. Також необхідно додати метод refresh(), який буде відслідковувати зміни в положенні миші та реалізовувати логіку відтискання миші. Для перевірки працездатності відслідковування мищі додамо додаткове логування (рис 3.5).

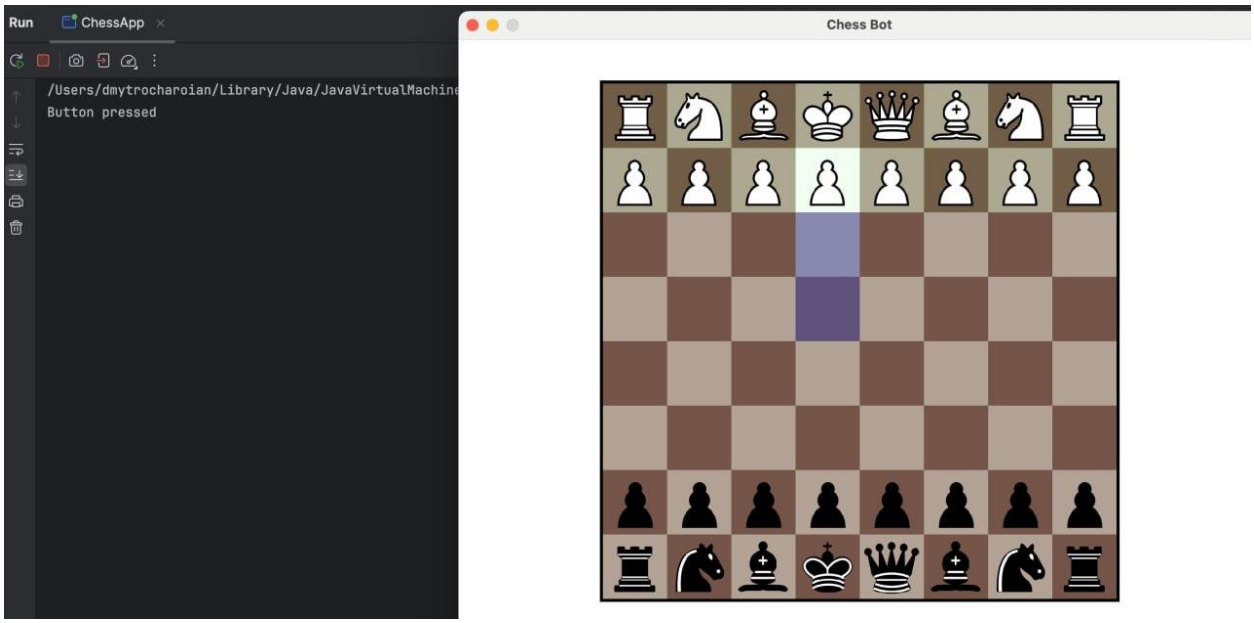

Рисунок 3.5 – Результат логування дій миші

Тепер треба для кожного класу фігур додати можливість розрахувати всі можливі ходи, розглянемо метод getPossibleMoves на прикладі фігури короля.

Метод getPossibleMoves буде визначати всі можливі ходи для фігури в один крок у восьми напрямках (вперед, назад, вліво, вправо і по діагоналях). Він повинен повертати список об’єктів Position, які представляють координати клітинок, куди фігура може перейти. Також передамо параметр protection який впливає на те, чи враховуються клітинки, зайняті своїми фігурами, як потенційні цілі (для захисту).

На початку створюється список positions, куди збиратимуться всі дозволені позиції для ходу. Далі задається масив directions, який містить вісім пар значень dx і dy – це зсуви по горизонталі та вертикалі відповідно до восьми напрямків на полі (вправо, вправо-вниз, вниз, вліво-вниз, вліво, вліво-вгору, вгору, вправо-вгору).

Цикл for перебирає кожен із напрямків. Для кожного напрямку обчислюються нові координати newX і newY як сума поточної позиції фігури та зміщення в обраному напрямку. Далі перевіряється, чи ці координати знаходяться в межах дошки. Якщо координати допустимі, отримується фігура, що стоїть на цій клітинці (board.getPiece(newX, newY)).

Після цього перевіряється, чи ця клітинка порожня (тобто piece == null), або якщо на ній стоїть фігура супротивника (piece.getTeam() != getTeam()) чи якщо передано protection == true. У разі виконання однієї з цих умов, нова позиція вважається допустимою і додається до списку можливих ходів. Таким чином, метод повертає всі позиції, куди фігура може легально переміститися з поточної точки у один хід в будь-якому з восьми напрямків, враховуючи захист, якщо це потрібно.

Також реалізуємо пересування фігур та додамо невелику анімацію для плавності руху (рис 3.6).

```
public void update(float delta) {  1 usage
    xMovAnim *= Math.pow(0.0005f, delta);
    yMovAnim *= Math.pow(0.0005f, delta);
}
```

Рисунок 3.6 – X та Y для анімації руху

Тепер, коли наші фігури можуть пересуватися по полю, настав час реалізувати логіку автоматичного ходу для чорної команди за допомогою алгоритму мінімакс. Ми хочемо, щоб коли настає черга ходу для чорних фігур, код автоматично викликав метод, який реалізує мінімакс і обирає найкращий хід.

Спочатку перевіряється, чи команда, яка повинна зробити хід, це чорні. Якщо так, то ми накопичуємо час у змінній timer, додаючи до неї значення delta (проміжок часу між кадрами у грі). Це дозволяє створити невелику затримку перед тим, як бот зробить хід, щоб усе не відбувалося миттєво.

Далі, якщо накопичений час перевищив 0.5 секунди і гра ще не завершена, ми обнуляємо timer і фіксуємо час початку обчислень. Після цього викликається метод getBestMove, якому передається поточна дошка, команда, яка має ходити (тобто чорні), та глибина пошуку – у нашому випадку 2. Цей метод аналізує ситуацію на полі і повертає найкращий хід за логікою мінімакса. Отриманий хід застосовується до дошки за допомогою методу apply. У результаті бот робить хід автоматично, ніби «обміркувавши» ситуацію, і ми також можемо за потреби заміряти час, який був витрачений на це рішення (рис 3.7).

Рисунок 3.7 – Пересування фігур на дошці

Розроблення алгоритму пошуку найкращого ходу почнемо з головного методу – getBestMove. Метод getBestMove буде призначений для вибору найкращого можливого ходу для певної команди на основі оцінки стану ігрової дошки. Він реалізуватиме елементарну форму алгоритму передбачення (він же мінімакс з обмеженою глибиною), де параметр prediction визначатиме глибину передбачення майбутніх ходів. Метод приймає три параметри: поточну дошку (Board board), команду, яка повинна зробити хід (Team team), і ціле число prediction, яке вказує, скільки кроків наперед слід передбачати.

Спочатку за допомогою методу getPossibleBoards (який ми заімплементимо пізніше) сгенеруються всі можливі дошки після одного ходу цієї команди. Далі створимо змінну bestBoard, яка буде містити найкращий варіант дошки, і змінну rating, яка збереже найвищу оцінку. Потім метод перебире кожен можливий варіант дошки. Якщо глибина передбачення більша за нуль, рекурсивно викликається getBestMove, щоб проаналізувати наступний

хід суперника або майбутні ходи на глибину prediction - 1. Таким чином, метод намагатиметися врахувати не тільки поточний хід, але й можливі дії у відповідь.

Після цього кожна отримана дошка буде оцінюватися методом BoardRating.rateBoard, який поверне числову оцінку для команди, що аналізується. Якщо рейтинг поточної дошки кращий за попередній максимум, метод оновлюватиме найкращий рейтинг і зберігатиме відповідну дошку в bestBoard. У підсумку, після аналізу всіх можливих варіантів, метод поверне дошку, яка забезпече найвищу оцінку для заданої команди з урахуванням передбачення ходів на певну глибину.

Далі треба реалізовати метод для отримання усіх доступних дошок getPossibleBoards. Метод getPossibleBoards призначений для того, щоб згенерувати всі можливі варіанти ігрової дошки після одного ходу будь-якої фігури певної команди.

Він буде приймати два параметри: поточний стан дошки (Board board) та команду, для якої потрібно знайти варіанти ходів (Team team). На початку створюється порожній список boards, у який поступово додаються нові варіанти дошки. Далі метод проходить по всіх клітинках дошки за допомогою вкладених циклів. Для кожної клітинки отримується фігура, що в ній знаходиться. Якщо клітинка порожня (тобто фігура дорівнює null), або якщо фігура належить іншій команді, то ця клітинка пропускається.

Якщо ж фігура належить тій команді, для якої ми шукаємо можливі ходи, тоді викликається метод getPossibleMoves(false) для отримання всіх можливих позицій, куди ця фігура може перейти. Для кожного можливого ходу створюється копія поточної дошки, після чого фігура на копії дошки пересувається на нову позицію згідно з поточним ходом. Отримана нова дошка після ходу додається до списку можливих варіантів. Після завершення проходу по всіх фігурах та їх можливих ходах, метод повертає список усіх

можливих станів дошки, які можуть виникнути після одного ходу будь-якої фігури заданої команди.

Метод оцінювання дошки, який буде оцінювати кожну дошку згідно Value фігур - rateBoard. Для цього встановимо оцінку на 0.0 для кожної дошки, та будемо регулювати value в залежності від ситуації на дошці. Для цього проходимо по всіх клітинках дошки (двовимірний масив pieces[x][y]) і якщо в клітинці немає фігури – пропускаємо.

Далі отримуємо вартість фігури - int value = DefaultPiece.getValue(); та якщо фігура належить нашій команді – додаємо її вагу до загального рахунку, а якщо суперника – віднімаємо її вартість.

Лістинг 3.7 Реалізація підрахунку очок ходу:

```
if (DefaultPiece.getTeam() == team) {
    score += value * 8.0f;
} else {
   score -= value * 8.0f;
}
```

Після цього отримуємо всі можливі ходи фігури з нашого методу getPossibleMoves та для кожної потенційної цілі (позиції, куди може сходити фігура), перевіряємо, чи є там інша фігура.

Якщо цільова фігура нашої команди, і атакуюча – теж наша то можливо, ми її захищаємо і це дає нам невеликий бонус. Якщо атакуюча ворожа то вона нам загрожує і ми отримуємо штраф.

Якщо ціль – ворожа фігура і ми атакуємо, то бонус за можливу атаку. Якщо противник атакує нас то невеликий штраф. І нарешті Повертаємо підраховане число – загальну оцінку стану дошки для команди team - return score.

Цей метод дозволяє ШІ оцінювати, наскільки вигідна певна позиція на дошці в рамках алгоритму мінімакс або іншого пошуку найкращого ходу і тепер замість нас чорними фігурами буде керувати шаховий ШІ на основі алгоритма мінімакс.

3.2 Модифікация та рефакторинг коду

У процесі розробки шахового додатка було проведено декілька важливих модифікацій і рефакторингів, які покращили зручність користування програмою, її візуальну складову та функціональні можливості.

Одним з нововведень стала реалізація підсвічування всіх можливих ходів вибраної фігури. Після натискання на фігуру користувача на дошці, усі допустимі клітинки для переміщення підсвічуються синім кольором (рис 3.8).

Рисунок 3.8 – Приклад підсвідки ходів фігури ферзя

Ця функція дозволяє покращити інтуїтивну взаємодію з інтерфейсом. Зменшити кількість помилкових ходів та сприяє швидшому прийняттю рішень, особливо для початківців.

Для реалізації цієї функціональності був доданий метод getPossibleMoves() у клас Piece, який повертає список позицій, доступних для переміщення фігури. У класі Board було додано логіку візуального відображення цих позицій при рендерінгу поля (рис 3.9).

Рисунок 3.9 – Приклад підсвідки ходів фігури коня

Ще одна важлива зміна – вимірювання часу, який потрібен штучному інтелекту для обчислення ходу. Це було реалізовано для аналізу продуктивності алгоритму мінімакс, подальшої оптимізації (наприклад,

глибини передбачення) та відображення інформації для користувача – наприклад, «Бот обдумував хід: 43 mc».

Для реалізації було використано наступну логіку.

Спочатку фіксується час початку обчислення, тобто момент, коли бот починає обдумувати свій хід. Це здійснюється за допомогою функції, яка отримує поточний час у наносекундах.

Далі викликається метод getBestMove, який аналізує поточну дошку, команду, яка має зробити хід, і глибину передбачення (у даному випадку 2). Метод визначає найкращий хід для бота з урахуванням можливих варіантів розвитку подій, після чого цей хід застосовується до поточної дошки.

Після завершення обчислень знову фіксується час, вже як час завершення розрахунку. Потім обчислюється тривалість, протягом якої бот приймав рішення, шляхом віднімання початкового часу від кінцевого.

На завершення ця тривалість в наносекундах конвертується в мілісекунди і виводиться у консоль у вигляді повідомлення: «Бот обдумував хід: X мс», де X – це кількість мілісекунд, витрачених на прийняття рішення. Таким чином, користувач отримує уявлення про те, скільки часу зайняло обчислення найкращого ходу.

Під час додавання нового функціоналу також було проведено частковий рефакторинг: Винесено логіку рендерингу підсвічування в окремий метод, що підвищило читабельність коду. Визначення можливих ходів було перенесено до моделі фігур, що відповідає принципам інкапсуляції. Зменшено дублювання коду у класі Board, пов’язане з оновленням стану після ходу.

Виміряний час виводиться в консоль гри після завершення ходу комп’ютера (рис 3.10).

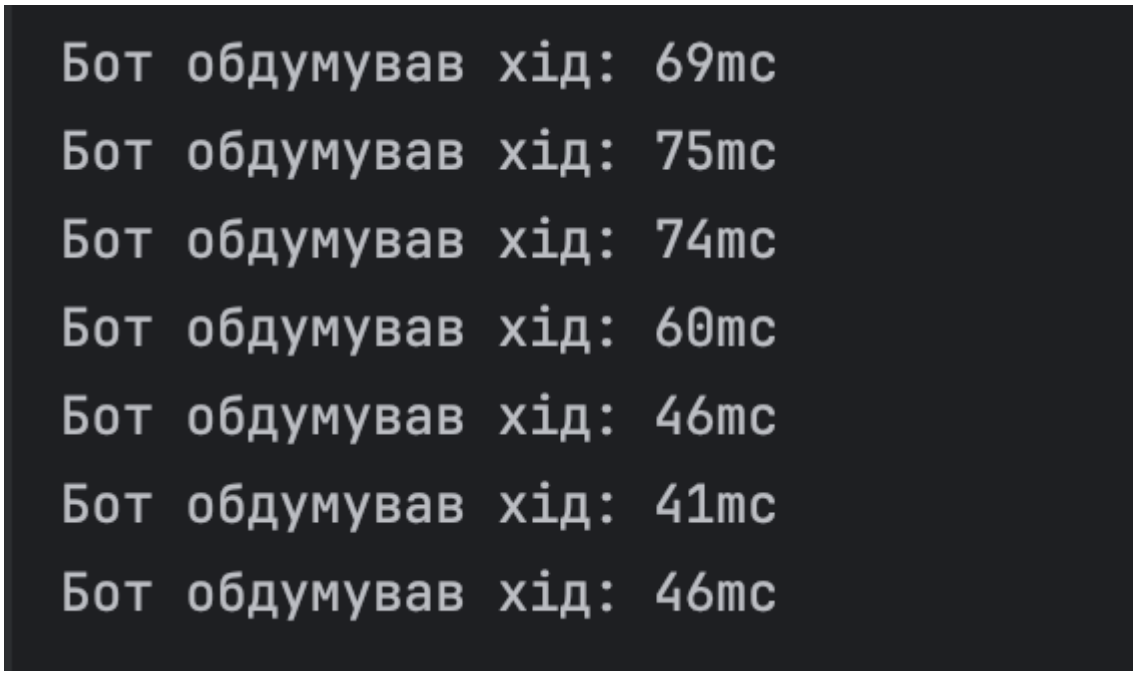

Рисунок 3.10 – Приклад заміру часу ходу бота

3.3 Заходи щодо поліпшення застосунку

Застосунок шахового ШІ має великий потенціал до удосконалення. Наприклад, у майбутньому можна вдосконалити такі аспекти проєкту:

− темна тема: додати можливість перемикання між світлою та темною темами. Це забезпечить користувачам зручне використання застосунку в різних умовах освітлення, зменшить навантаження на очі у темний час доби та покращить загальний досвід користування;

− додати базу ендшпілів для точнішої гри на завершальних етапах парті, такі бази, наприклад таблиці Штокфіша чи Lomonosov tablebases, містять наперед прораховані ідеальні ходи для позицій з малою кількістю фігур. Можна буде додати перевірку якщо на дошці менше певної кількості фігур (наприклад, ≤6), використовувати базу для ідеального ходу;

− покращити евристичну функцію оцінки позиції додавши більше факторів в оцінку такі як контроль центру, активність фігур, слабкі поля,

пішакову структуру, безпеку короля, або використати готові оцінювальні модулі;

− реалізувати адаптивний рівень складності для гравців, наприклад додати налаштування глибини пошуку в залежності від вибору складності, гравець зможе вибрати «легко», «середньо», «важко»;

− додати інтерфейс аналізу партій який допоможе гравцю аналізувати свої партії та покращувати результат або Інтегрувати зовнішній аналізатор (наприклад, з Stockfish) для оцінки кожного ходу;

Отже, застосунок має широкий спектр можливостей для майбутніх покращень, що дозволить розширити його функціонал та підвищити зручність користування.

ВИСНОВКИ

У кваліфікаційній роботі був розроблений застосунок для гри в шахи з використанняш ШІ на основі алгоритму мінімакс.

Основною метою роботи було створення інтелектуального компонента, здатного ефективно протистояти гравцю, а також забезпечення стабільної та зручної взаємодії користувача з програмою.

Під час розроблення були використані сучасні технології та бібліотеки. Для розробки серверної частини застосунку використовувалися такі технології як Java Core та Swing для створення графічного інтерфейсу, а також бібліотека AWT (Abstract Window Toolkit) для забезпечення виведення елементів графічного інтерфейсу та взаємодії з користувачем. Застосунок був розроблений за принципами об’єктно-орієнтованого програмування, що дозволило зробити код більш структурованим, розширюваним і зручним для подальшого вдосконалення. Процес розробки відбувався в середовищі IntelliJ IDEA, що забезпечило зручний та ефективний інструмент для написання та налагодження коду. Це середовище також надало потужні засоби для аналізу та оптимізації роботи програми.

Графічний інтерфейс реалізований у вигляді класичного віконного додатку з інтуїтивно зрозумілим розміщенням елементів керування та шахової дошки. Для точного підрахунку часу ходу бота в реалізації було використано методи бібліотеки System, яks дозволяють вимірювати час, витрачений на розрахунок ходу.

Робочу версію застосунку успішно розгорнуто і вона функціонує відповідно до запланованих вимог. Після детального аналізу виявлено можливості для подальшого вдосконалення застосунку. Ці покращення охоплюють як технічні аспекти, так і покращення користувацького досвіду, що дозволить зробити застосунок ще більш ефективним та зручним для користувачів.

Результатом роботи є програмний інструмент, який використовується для гри в шахи. Він може бути основою для подальших досліджень у галузі штучного інтелекту та розробки навчальних інструментів.

Результати роботи апробовано у вигляді тез доповідей під час XXIX Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті», онлайн конференції «Комп’ютерний зір, системний аналіз та математичне моделювання» [41].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. David Levy, Monty Newborn (1991). *How Computers Play Chess*.

2. Ivan Bratko (2011). *Prolog Programming for Artificial Intelligence* (4th ed.).

3. Edward Lasker (1960). *Chess and Machine Intuition*.

4. A Step-by-Step Guide to Building a Simple Chess AI URL: https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/ (дата звернення 11.04.2025).

5. Minimax Algorithm Applied to Chess Engines URL: https://fvcalderan.github.io/myworks/articles/minimax_chess_engine.pdf (дата звернення 20.03.2025).

6. Creating an AI that Plays Chess (Minimax Algorithm + Alpha-beta Pruning) URL: https://journeyaiart.com/blog-creating-an-ai-that-plays-chess-minimax-algorithm-alphabeta-pruning-48784 (дата звернення 15.04.2025).

7. Toby Walsh (2002). Advanced Topics in Artificial Intelligence: Lecture Notes in Computer Science.

8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein (2009). *Introduction to Algorithms* (3rd ed.).

9. Stuart Russell, Peter Norvig (2020). *Artificial Intelligence: A Modern Approach* (4th ed.).

10. Matthew Sadler, Natasha Regan (2019). *Game Changer: AlphaZero's Groundbreaking Chess Strategies and the Promise of AI*. New In Chess. ISBN: 978-90-5691-879-6.

11. Claude E. Shannon (1950). *Programming a Computer for Playing Chess*. Philosophical Magazine, Series 7, Volume 41, Issue 314. (дата звернення 17.04.2025).

12. Jon Kleinberg, Éva Tardos (2005). *Algorithm Design*. Pearson Education.

13. Wolfgang Ertel (2018). *Introduction to Artificial Intelligence*. Springer. ISBN: 978-3-319-58488-4.

14. Robert Hyatt, Albert Gower, Harry Nelson (1997). *Crafty Chess Engine Documentation*.(дата звернення 17.04.2025)

15. Jeremy Silman. *How to Reassess Your Chess: Chess Mastery Through Chess Imbalances*. Siles Press, 2007.

16. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. Advanced Information Systems, 9(1), 5–12.

17. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2025) Image description compression in classification structural methods, IEEE Access, vol. 13, pp. 43631-43641.

18. Gorokhovatskyi V., Tvoroshenko I. (2024) An effective method for transforming an image description into a compact vector for classification. Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / V. Snytyuk (Editor). – Kyiv: Publishing House «Caravela», 25-28.

19. Gorokhovatskyi, V., Gadetska, S., Stiahlyk, N. (2024) Classification of images based on distance assessment. Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / V. Snytyuk (Editor). – Kyiv: Publishing House «Caravela», 22-24.

20. Pupchenko, D., Gorokhovatskyi, V. (2024) Accelerated filtration of ultrasound images. Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / V. Snytyuk (Editor). – Kyiv: Publishing House «Caravela», 69-72.

21. Гороховатський В.О., Гадецька С.В., Стяглик Н.І. (2019) Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. Радіоелектроніка, інформатика, управління, №2, с. 100–107.

22. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudáková M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, IEEE Access, vol. 12, pp. 73376-73385.

23. Gorokhovatskyi V., Gadetska S., Stiahlyk N. (2020) Image structural classification technologies based on statistical analysis of descriptions in the form of bit descriptor set. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020), 2608, pp. 1027-1039.

24. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, Computers, Materials & Continua, vol. 80, no. 2, pp. 3085-3106.

25. Gadetska, S.V., Gorokhovatskyi, V. O., Stiahlyk, N. I., Vlasenko, N.V. Statistical data analysis tools in image classification methods based on the description as a set of binary descriptors of key points. Radio Electronics, Computer Science, Control, 2021, №4, pp. 58-68.

26.Swing Components in Java https://www.educba.com/swing-components-in-java. (дата звернення 05.05.2025)

27. Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., Vlasenko, N. (2023) Explanation of CNN Image Classifiers with Hiding Parts. In: J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot (eds), Explainable Deep Learning Artificial Intelligence, pp. 125-146, Academic Press, 346 p.

28. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, Indonesian Journal of Electrical Engineering and Computer Science, 33 (1), 113-125.

29. Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating Image Classification based on a Model for Estimating Descriptor-to-Class Distance. International Journal of Computing, 22(4), 485-492.

30. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, IEEE Access, 11, 126938-126949.

31. V. Gorokhovatsky, Y. Putyatin and V. Stolyarov (2017) Research of Effectivenss of Structural Image Classification Methods using Cluster Data Model, Radio Electronics Computer Science Control, vol. 3, no. 42, pp. 78-85.

32. Gorokhovatsky V.A., Putyatin Ye.P. (2009) Image Likelihood Measures of the Basis of the Set of Conformities. Telecommunications and Radio Engineering, 68 (9), pp. 763-778.

33. Gorokhovatsky, V.A., Putyatin, Y.P. (2008) Structural recognition of images on the basis of voting models of attributes of typical points, Data recording, storage and processing, 10(4), 75-85.

34. V. A. Gorokhovatskiy, (2011), Compression of descriptions in the structural image recognition, Telecommunications and Radio Engineering, vol. 70, no. 15, pp. 1363–1371.

35. Gorokhovatskyi, V., Vlasenko, N. (2021). Редукція опису зображення у складі множини дескрипторів на основі метричного критерію інформативності. Advanced Information Systems, 5(4), pp. 10-16.

35 David Levy, Monty Newborn. *How Computers Play Chess*. W. H. Freeman, 1991.

36. Silver, D., Hubert, T., Schrittwieser, J., et al. (2018). *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. Science, 362(6419), 1140–1144.

37. Leela Chess Zero. *Technical Overview and FAQ*. https://lczero.org/ (дата звернення 15.04.2025).

38. K. A. Gelfand, M. G. Shvartsman, V. G. Chirov, & E. D. Roth (2010). Chess Programming: A Technical Survey

39. Hsu, F. (2002). Behind Deep Blue: Building the Computer that Defeated the World Chess Champion.

40. Booch, G., Rumbaugh, J., & Jacobson, I. (1999). The Unified Modeling Language User Guide.

41. Чароян Д. О. (2025) Інтелектуальні засоби у шаховій грі. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.).* Харків: ХНУРЕ, 2025. Т.7. С. 167-168.