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)

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ГРИ В ШАХИ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ PYTHON
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-1

Тарасов А.М.
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Вечірська І.Д.
(посада, прізвище, ініціали)

Допускається до захисту

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

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Тарасову Артуру Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для гри в шахи з використанням технології Python

затверджена наказом університету від 16 травня 2022 року № 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 травня 2022 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова програмування Python, бібліотека для розробки ігор PyGame, середовище розробки застосунків PyCharm, графічний редактор для обробки та створення зображень Adobe Photoshop, операційна система Linux.

4. Перелік питань, що потрібно опрацювати в роботі
1. Предметна область та алгоритм.
2. Опис технологій та програмної реалізації базового функціоналу.
3. Опис програмної реалізації штучного інтелекту.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) скріншоти дизайну, скріншоти ходу гри та пояснення головних фрагментів коду, алгоритм для штучного інтелекту.

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
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз технічних засобів	26.04.22-30.04.22	
5	Розробка базового фунціоналу	01.05.22-12.05.22	
6	Програмна реалізація штучного інтелекту	13.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Перевірка на плагіат	01.06.22	
9	Рецензування	02.06.22	
10	Підготовка презентації та доповіді	03.06.22-04.06.22	
11	Занесення роботи в електронний архів	05.06.22	
12	Попередній захист кваліфікаційної роботи	06.06.22	

Дата видачі завдання 18 квітня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ _____ доц. Вечірська І.Д.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 55 с., 30 рис., 1 дод., 30 джерел.

МОВА PYTHON, БІБЛІОТЕКА PYGAME, СЕРЕДОВИЩЕ РОЗРОБКИ PYCHARM, ГРАФІЧНИЙ РЕДАКТОР ADOBE PHOTOHOP.

Об’єктом роботи є розробка гри «Шахи».

Метою кваліфікаційної роботи є розроблення гри «Шахи» з використанням простого штучного інтелекту для гри проти комп’ютера.

Стрімкий розвиток комп’ютерної техніки в останні роки, поява найпотужніших графічних прискорювачів і центральних процесорів сприяло не менше бурхливому розвитку індустрії комп’ютерних ігор. Видатні розробки цієї галузі – це складні програми, як правило, з дуже високими вимогами до апаратної частини комп’ютера. Ця робота спрямована на розвиток навичок з Python програмуванням.

Під час розробки були використані: мова програмування високого рівня Python, бібліотеки мови Python середовище розробки для Python-PyCharm.

PYTHON LANGUAGE, PYGAME LIBRARY, PYCHARM DEVELOPMENT ENVIRONMENT, GRAPHIC EDITOR ADOBE PHOTOSHOP.

The object of work is the development of the game "Chess". The rapid development of computer technology in recent years, the emergence of the most powerful graphics accelerators and CPUs have contributed no less to the rapid development of the computer game industry. Outstanding developments in this field are complex programs, usually with very high requirements for computer hardware. This work aims to develop skills with Python programming.

During the development were used: high-level programming language Python, Python language libraries development environment for Python-PyCharm

ЗМІСТ

ВСТУП

На даний момент існує багато комп’ютерних ігор, так само різноманітні підходи в їх створенні.

Наприклад, популярні останнім часом швидкі ігри (найчастіше розраховані на багато користувачів, через мережу Інтернет) створюються за технологією Unity або Unreal Engine з використанням мови JavaScript, Java, Swift або C++. Такі ігри дуже популярні на різноманітних платформах (такі як ПК, Ігрові консолі, Браузер чи мобільний пристрій).

Великі, складні ігри з реалістичною 3D графікою пишуться на C ++, окремі модулі до них можуть бути написані на інших мовах (наприклад AI - «штучний інтелект» – на Python). Взагалі, мова для створення гри вибирається як компроміс між вимогами до гри, до комп’ютерного чи мобільного заліза і до вартості розробки.

На переважній більшості мобільних телефонів встановлено ARM процесори. Тому гри для мобільних телефонів на платформі Android та iOS пишуться на спеціальній мові для мобільних телефонів Swift та Kotlin.

Також існують спеціальні програми для написання ігор. Яскравий приклад – програма Unreal Engine. Ця програма дозволяє змоделювати захоплюючу гру навіть без знання мов програмування!

В рамках кваліфікаційної роботи необхідно розробити невелику програму гри в «Шахи» із застосуванням графіки на мові високого рівня з використанням об’єктно-орієнтованого програмування та розробити штучний інтелект для гри проти комп’ютера.

«Шахи» – популярна гра людей будь-якого віку. Вона відрізняється простотою, сприяє розвитку уваги і елементарної логіки. У даній роботі ця гра реалізована на мові Python.

1 ПРЕДМЕТНА ОБЛАСТЬ ТА АЛГОРИТМ

1.1 Алгоритми гри у «Шахи»

У шахів досить прості правила. Дві протиборчі сторони, шість різновидів фігур та одна мета – дати мат супернику.

Але при цьому варіативність шахів просто величезна. Існує 400 унікальних комбінацій першого ходу – 20 варіантів першого півходу білих та 20 варіантів відповіді чорних. З кожним наступним перебігом кількість унікальних позицій збільшується на рівень.

Загальна кількість унікальних партій у шахи становить приблизно 10 у степені 120.

Шахам не загрожує бути повністю порахованим. Тому в бій вступають алгоритми оцінки позиції та дерево можливих ходів [1] (рис. 1.1).

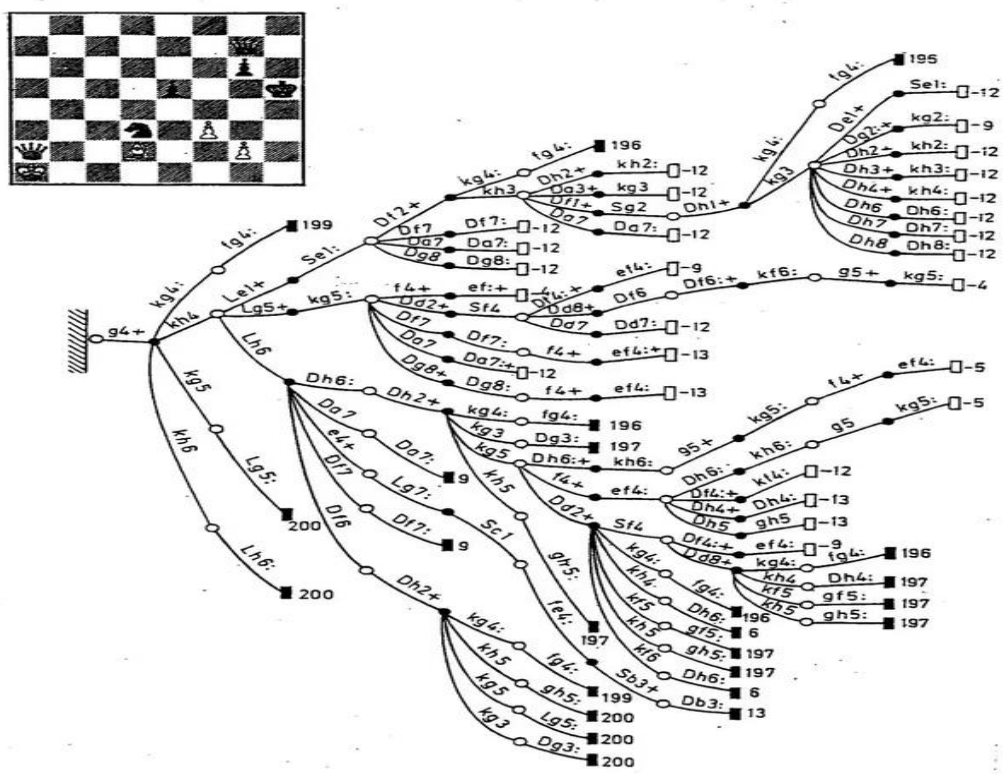

Рисунок 1.1 – дерево можливих ходів

У шахівниці у кожної фігури є своя цінність, яка вимірюється в пішаках:

– Кінь – 3 пішаки;

– Слон – 3 пішаки;

– Ладья – 5 пішаків;

– Ферзь – 9 пішаків;

– Пішак – 1 пішак;

– Король – безцінний, бо його втрата означає програш партії. Аналіз сучасних машин підтверджує істинність такої оцінки. Так, залежно від позиції на дошці комп'ютер оцінює ферзя в 9 – 12 пішаків, човном – у 5 – 6, коня та слона – у 3–5. Короля ж машина оцінює у 300 пішаків. Це задає максимальну межу оцінки.

Щоб було зрозуміліше, перевага в 0,5 пішаків – це вже непогано для шахіста. У цілий пішак – серйозна перевага. У 3 пішаки – переважна перевага, яку можна практично без проблем довести до перемоги. Але рахункові можливості машини обмежені. Іноді вона показує оцінку +51 або щось на зразок. Це означає, що алгоритм бачить колосальну перевагу білих у позиції та матеріалі, але не може знайти конкретний шлях до мату.

Мінімакс, або прямий перебір варіантів, у такому разі не працює. Навіть КМС без проблем знайде на дошці мат у 3 ходи в миттельшпилі, коли на дошці ще багато фігур. А програмі для цього потрібно буде перебрати понад 750 млн. півходів. Навіть якщо програма перебирає 1 млн варіантів за секунду, щоб знайти мат у 3 ходи, їй знадобиться до 750 секунд, або 12,5 хвилин. І це глибина в три ходи. У стратегічних позиціях, де розвиток гри йде з урахуванням п'ять чи десять ходів уперед, такі програми взагалі будуть марними. Тому для аналізу позиції використовується алгоритм під назвою «альфа-бета-відсікання» [2] (рис. 1.2).

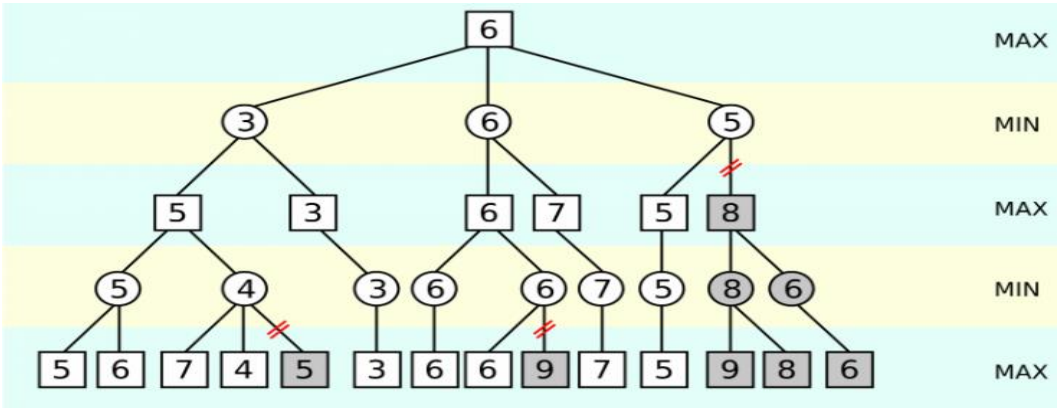

Рисунок 1.2 – «альфа-бета-відсікання»

Система аналізує початкові варіанти ходів і одночасно відсікає ті їх, які ведуть до миттєвого погіршення оцінки.

Програма відкидає ті варіанти, в яких вона відразу програє матеріал або включають комбінації з боку суперника, в ході яких вона виграє матеріал чи партію.

Це дозволяє скоротити кількість робочих ліній на порядки, зосереджуючи обчислювальні ресурси лише з тих гілках дерева, які у перспективі ведуть поліпшення позиції.

1.2 Стратегія, тактика та стадії гри

Шахова стратегія складається з постановки і досягнення довгострокових позиційних переваг під час гри – наприклад, питання де розмістити різні фігури – а тактика зосереджується на безпосередніх маневрах. Ці дві частини процесу шахової гри не можна повністю відокремити одну від одної, тому що стратегічні цілі переважно досягаються за рахунок тактики, тоді як тактичні можливості виникають з попередньої стратегії гри. Шахова партія зазвичай ділиться на три етапи: дебют, зазвичай перші 10 ходів, коли гравці пересувають свої фігури на вигідні для подальшої боротьби позиції; потім міттельшпіль; і зрештою ендшпіль, коли більшість

фігур вже відсутні, королі зазвичай беруть активнішу участь у боротьбі, а просування пішаків часто є вирішальним.

1.2.1 Основи тактики

Тактика загалом зосереджується на короткострокових діях – настільки короткострокових, що їх може розрахувати наперед людина або комп’ютер. Можлива глибина розрахунку залежить від уміння гравця. У спокійній позиції, з великою кількістю можливостей з обох сторін, глибокий розрахунок є складнішим і не може бути практичним, тоді як у «тактичний» позиції з обмеженою кількістю форсованих варіантів сильні гравці можуть розраховувати довгі послідовності ходів. Прості одно або двоходові тактичні дії – загрози, обміни матеріалу, подвійні атаки, можна об’єднувати у більш складні комбінації, послідовності тактичних маневрів, які часто вимушені з точки зору одного або обох гравців. Теоретики описують багато елементарних тактичних прийомів та типових маневрів, наприклад, зв’язування, вилка, лінійний удар, батарея, відкритий напад (особливо відкритий шах), проміжний хід, відволікання, заманювання, жертва, знищення захисту, перевантаження і перекриття. Форсований варіант, який містить жертву і, як правило, приносить суттєву вигоду, називається комбінацією. Блискучі комбінації – такі, як у безсмертній партії – вважаються шедеврами і ними захоплюються любителі шахів. Зазвичай шахові задачі, направлені на розвиток ігрових навичок, показують гравцям позиції, які містять комбінацію, а гравець має її знайти [3].

1.2.2 Основи стратегії

Стратегія пов’язана з оцінкою шахових позицій і з встановленням цілей та довгострокових планів на подальшу гру. В ході оцінки гравці повинні враховувати множина факторів, таких як цінність фігур на шахівниці, контроль центру і централізація, пішакова структура, безпека короля, а також контроль ключових полів або групи полів (наприклад, діагоналей, відкритих вертикалей і темних або світлих полів). Основний крок в оцінці позиції полягає в підрахунку відносної цінності шахових фігур, своїх і суперника. Очки, які використовують для цього, ґрунтуються на досвіді; зазвичай пішака вважають вартим одного очка, коней і слонів близько трьох очок кожен, туру близько п’яти очок (різниця вартості між турою і слоном або конем називається якістю), а ферзя близько дев’яти очок. Король є ціннішим, ніж всі інші фігури разом, оскільки мат йому означає поразку. Але в практичному сенсі в ендшпілі король як бойова фігура зазвичай сильніший за слона або коня, але слабший, ніж тура. Ця базова оцінка змінюється під впливом інших факторів, таких як положення фігури (наприклад, просунуті пішаки зазвичай цінніші, ніж коли вони на своїх початкових полях), координацією між фігурами (наприклад, пара слонів зазвичай скоординована краще, ніж слон і кінь), а також типом позиції (наприклад, коні зазвичай переважають слонів у закритих позиціях з багатьма пішаками, тоді як слони сильніші у відкритих позиціях.

Ще одним важливим фактором в оцінці шахової позиції є пішакова структура (іноді її називають пішаковим скелетом), або конфігурація пішаків на шахівниці. Пішаки є найменш мобільними шаховими фігурами, пішакова структура відносно статична і багато в чому визначає стратегічний характер позиції. Такі слабкості в пішаковій структурі, як ізольовані, подвоєні пішаки, відсталі пішаки, а також дірки після утворення часто стають постійними. Тому необхідно дотримуватися обережності, щоб уникнути цих недоліків,

якщо вони не компенсуються іншими надбаннями (наприклад, можливістю розгорнути атаку) [4].

1.2.3 Стадії гри

Шаховий дебют – це низка початкових ходів у грі («дебютних ходів»). Визнані послідовності дебютних ходів мають свої назви, такі як Іспанська партія або Сицилійський захист. Вони каталогізовані в довідкових працях, таких як Енциклопедія шахових дебютів. Існують десятки різних дебютів, характер яких варіюється в широкому діапазоні, від спокійної позиційної гри (наприклад, Дебют Реті) до дуже агресивних (Латиський гамбіт). Для деяких дебютних ліній точну послідовність ходів, яку вважають найкращою для обох суперників, проаналізовано до понад 30 ходів. Професійні гравці витрачають роки на вивчення дебютів і продовжують це робити упродовж усієї своєї кар'єри разом з розвитком дебютної теорії. Основні стратегічні цілі схожі для більшості дебютів:

– розвиток: метод розміщення фігур (особливо слонів і коней) на корисні поля, звідки вони матимуть оптимальний вплив на гру;

– контроль над центром: контроль над центральними полями дозволяє фігурам переміститись у будь-яку частину шахівниці відносно легко, а також обмежує можливість дій опонента;

– безпека короля: дуже важливо зберегти короля в безпеці від загрози мату. Вчасно зроблена рокіровка часто сприяє цьому;

– пішакова структура: гравці прагнуть уникнути утворення таких пішакових слабкостей, як ізольовані, здвоєні, відсталі пішаки і наявність пішакових островків, а також спричинити появу таких слабкостей у позиції супротивника.

Більшість гравців і теоретиків вважають, що білі завдяки праву першого ходу починають гру з невеликою перевагою. Це спочатку дає білим

ініціативу. Чорні зазвичай прагнуть нейтралізувати перевагу білих і домогтися рівності, або ж шукають динамічної контргри в нерівноважній позиції [5].

Міттельшпіль – стадія гри, яка розпочинається після дебюту. Чітку межу між дебютом і мітельшпілем провести неможливо, але зазвичай міттельшпіль розпочинається коли більшість фігур вже розвинулись. (Аналогічно, немає чіткого переходу з мітельшпіля в ендшпіль). Оскільки дебютна теорія вже закінчилася, то гравці повинні сформувати плани, взявши до уваги особливості позиції, і водночас враховувати тактичні можливості позиції. Міттельшпіль – це стадія, в якій відбувається більшість комбінацій. Комбінації – це серії тактичних ходів для досягнення якоїсь вигоди. Комбінації в міттельшпілі часто пов’язані з нападом на короля супротивника. Деякі типові схеми мають власні назви, наприклад, Мат Бодена або комбінація Ласкер проти Бауера. Специфічні плани або стратегічні теми часто виникають з певних груп дебютів, які призводять до певних типів пішакової структури. Прикладом є атака меншості, коли гравець, маючи менше пішаків на ферзевому фланзі, атакує на ньому суперника, у якого там більше пішаків. Таким чином, вивчення дебютів пов’язане з підготовкою планів, які є типовими для того чи іншого мітельшпілю. Ще одне важливе стратегічне питання мітельшпілю полягає в тому, як і коли розміняти матеріал і перейти в ендшпіль, тобто спростити позицію. Незначну матеріальну перевагу зазвичай можна реалізувати лише в ендшпілі, тож сильніша сторона повинна вибрати правильний спосіб переходу в ендшпіль. Не кожен розмін матеріалу годиться для цієї мети. Наприклад, якщо суперники мають слони різного кольору, то перехід в ендшпіль зі слонами і пішаком зазвичай вигідний для слабшої сторони, оскільки ендшпіль з різнокольоровими слонами в переважній більшості веде до нічиєї навіть за явності зайвого пішака, часом навіть двох [6].

Ендшпіль – стадія гри, коли на шахівниці залишилося мало фігур. Існує три основні стратегічні відмінності ендшпілю від попередніх стадій гри:

– більшої ваги набувають пішаки. Ендшпілі часто обертаються навколо прагнення замінити пішака на сильнішу фігуру шляхом просування його на останню горизонталь;

– Король, який у мітельшпілі потребував захисту від мату, стає в ендшпілі сильною фігурою. Його часто переміщують в центр шахівниці, де він може захистити своїх пішаків, атакувати пішаків суперника і перешкоджати ходам короля супротивника [7].

Цугцванг змушує гравця робити невигідний для нього хід. Він набагато частіше трапляється в ендшпілі, ніж на іншій стадії гри. На діаграмі наведено приклад обопільного цугцвангу. Якщо хід чорних, то 1…Kb7 дозволяючи білим просунути пішака після 2.Kd7; якщо черга білих, то гра завершується внічию, або через пат внаслідок 1.Kc6 або через втрату пішака після будь-якого іншого ходу [8].

Ендшпіль можна класифікувати за типом фігур, що залишилися на шахівниці. Базові мати – позиції, в яких один гравець має тільки короля, а його суперник крім короля має одну або дві фігури і спроможний поставити мат завдяки взаємодії фігур з королем. Наприклад, король і пішак містить лише королів і пішаків з обох боків і задача полягає в тому, щоб провести пішака на останню горизонталь, де він перетворюється на фігуру. Інші, складніші ендшпілі класифікуються в залежності від фігур, які залишились на шахівниці окрім королів, наприклад «Тура та пішак проти тури».

1.3 Історія гри «Шахи»

Шахи – абстрактна стратегічна гра на спеціальній дошці, що має назву шахівниця й поділена на 64 світлі та темні клітини (поля), між 16 світлими (білими) і 16 темними (чорними) фігурами за встановленими для них правилами пересування. У цю гру грають мільйони людей по всьому світі. Походить від стародавньої індійської гри чатуранґа, яка, крім того, є

ймовірним предком східних стратегічних ігор сянці, чангі і сьоґі. Шахи потрапили до Європи в 9 столітті, внаслідок завоювань Омейядів в Іспанії. Фігури набули своєї нинішньої сили в Іспанії наприкінці 15 століття; правила стандартизовано в 19 столітті. Як одна з найпоширеніших спортивних ігор сучасності, поєднує в собі елементи мистецтва (уяви), науки (логічно-точний розрахунок) і спорту.

Шахи, як ми їх знаємо сьогодні, народилися з індійської гри чатуранга до 600-х років нашої ери. Протягом наступних століть ця гра поширилася по Азії та Європі, а в 16 столітті перетворилася на те, що ми знаємо як шахи. Одним з перших майстрів гри був іспанський священик на ім'я Руй Лопес. Хоча він не винайшов дебют, названий на його честь, він проаналізував це в книзі, яку опублікував у 1561 році. Теорія шахів тоді була настільки примітивною, що Лопес відстоював стратегію гри із сонцем в очах вашого суперника [9].

Форма чатуранги або шатрандж потрапила в Європу через Персію, Візантійську імперію і, можливо, найважливіше з усіх, розширювану Аравійську імперію. Найдавнішу зареєстровану гру, знайдену в рукописі 10-го століття, грали між багдадським істориком, який, як вважають, був улюбленцем трьох послідовних халіфів, та учнем.

Мусульмани принесли шахи в Північну Африку, Сицилію та Іспанію до 10 століття. Приблизно в той же час східні слов'яни поширили його в Київську Русь. Вікінги рознесли гру аж до Ісландії та Англії і вважаються відповідальними за найвідомішу колекцію шахових фігур, 78 фігурок із слонової кістки різних наборів, які були знайдені на острові Льюїс у Зовнішніх Гебридських островах у 1831 році та датовані 11-м. або 12 ст.

Ігри в шахи та кості періодично заборонялися королями та релігійними лідерами. Наприклад, король Людовик IX заборонив гру у Франції в 1254 році. Однак популярності гри сприяла її соціальна прихильність: шаховий набір часто асоціювався з багатством, знаннями та владою. Він був фаворитом королів Англії Генріха I, Генріха II, Іоанна і Річарда I, Філіпа II і

Альфонсо X (Мудрого) і Івана IV (Грозного) Росії. Вона була відома як королівська гра ще в 15 столітті.

З часів чатуранги зовнішній вигляд виробів чергувався між простим і багато прикрашеним. Простий дизайн предметів до 600 р. н.е. поступово привів до фігуративних наборів із зображенням тварин, воїнів і дворян. Але мусульманські набори 9 – 12 століть часто були нерепрезентативними і виготовлялися з простої глини або різьбленого каменю відповідно до ісламської заборони зображень живих істот. Вважається, що повернення до більш простих, символічних фігур шатрандж підсилило популярність гри, полегшивши виготовлення наборів і перенаправивши увагу гравців зі складних частин на саму гру.

Стилізовані набори, часто прикрашені дорогоцінним і напівдорогоцінним камінням, повернулися в моду, коли гра поширилася в Європі та Росії. Ігрові дошки, які мали однотонні квадрати в мусульманському світі, почали мати чергування чорних і білих, або червоних і білих, квадратів до 1000 року нашої ери і часто виготовлялися з тонкого дерева або мармуру. Петро I (Великий) Росії мав спеціальні похідні дошки з м'якої шкіри, які він носив під час військових дій. Король став найбільшою фігурою і придбав корону, а іноді і витончений трон і булаву. Близьке ототожнення лицаря з конем сходить до чатуранги. Пішка, як найнижча за владою та соціальним становищем, традиційно була найменшою і найменш репрезентативною з фігур. Королева зросла у розмірах після 1475 року, коли її повноваження розширилися, і вона перетворилася з чоловіка-порадника на дружину короля. Єпископ був відомий під різними іменами – «дурень» французькою і «слон» російською, наприклад, – і не був загальновизнаним відмітною митрою до 19 століття. Зображення грака також значно варіюється. У Росії його зазвичай представляли як вітрильний корабель до 20 століття. В інших місцях це був воїн у колісниці чи башті замку. Стандарт для сучасних наборів був встановлений приблизно в 1835 році за простим дизайном англійця Натаніеля Кука. Після того, як він був запатентований в

1849 році, його схвалив Говард Стонтон, тоді найкращий гравець світу; через широке просування Стонтона, згодом він став відомий як шаблон Стонтона. Сьогодні на міжнародний конкурс допускаються лише набори, засновані на дизайні Staunton.

Гра не містить прихованої інформації. Кожен гравець починає гру маючи 16 фігур: одного короля, одного ферзя, дві тури, двох коней, двох слонів, а також вісім пішаків. Кожен з шести типів фігур рухається по-різному, найпотужнішим є ферзь, а найслабшим – пішак. Мета гри – поставити мат королю суперника, помістивши його під неминучу загрозу взяття. З цією метою гравці використовують свої фігури, щоб атакувати і брати фігури супротивника, водночас підтримуючи одна одну. Зазвичай трапляються розміни однієї фігури на еквівалентну фігуру супротивника, але гравці можуть і знаходити можливості для розміну однієї фігури на дві, або ж отримувати кращу позицію. Перемогти у грі можна не лише поставивши мат, але й змусивши супротивника здатися, або ж, для гри з обмеженням часу, якщо у суперника час закінчиться. Також існує кілька випадків, коли гра може закінчитися внічию.

Теорія шахів рухалася швидкими темпами до середини 18 століття. У 1749 році на сцену вийшов французький майстер Франсуа-Андре Філідор зі своєю книгою під назвою Analyze du jeu des Échecs. У цій книзі висвітлювалися деякі нові вступні ідеї (включаючи захист, який все ще носить його ім’я), а також містив знаменитий захист Філідора в ендшпілях з ладою та пішаком – техніка эндшпилю, яка використовується й донині. Відоме твердження Філідора про те, що «Пішки — це душа шахів», вперше було представлено світу в цій книзі.

Шахи продовжували набирати популярність у всьому світі, а в середині 19 століття відбулася стандартизація шахових наборів. До 1850-х років шахові набори взагалі не були однорідними. У 1849 році Jaques of London (виробник ігор та іграшок) представив новий стиль виробів, створених Натаніелем Куком. Ці ж твори були схвалені Говардом Стонтоном,

найсильнішим гравцем свого часу. Цей новий стиль виробів, відомий як візерунок Стонтона, миттєво став популярним і використовувався на турнірах і клубах по всьому світу. Фігури Стонтона та незначні їх варіації досі вважаються стандартом для турнірних шахових наборів.

Введення шахових годинників до змагальної гри у 19 столітті. До того, як шахові годинники стали нормою, одна гра могла тривати до 14 годин! Зі стандартизацією шахових наборів і введенням шахових годин було встановлено обладнання, необхідне для сучасних матчів і турнірів.

Шахи самі по собі дуже розвивалися протягом 1800-х років. Найвідомішими іграми цього періоду були химерні атакуючі ігри - сильні оборонні ідеї ще не були навчені. Якщо гравець не жертвував своїми фігурами праворуч і ліворуч, намагаючись жорстоко поставити мат своєму супернику, то це була не весела гра! Саме в цю атакуючу епоху в шахах на сцену вийшов американський гравець Пол Морфі [10].

Чемпіонат світу став більш формалізованим після того, як Морфі пішов у відставку, а Андерсен зазнав поразки від Вільгельма Стейніца з Праги в матчі 1866 року. Стейніц був першим, хто претендував на повноваження визначати, як проводити матч за титул. Він встановив ряд правил і фінансових умов, за яких він буде захищати свій статус найкращого гравця світу, а в 1886 році він погодився зіграти з Йоганном Цукертортом з Австрії в першому матчі, спеціально призначеному для чемпіонату світу. Стейніц залишав за собою право визначати, чий виклик він прийме, коли і як часто він буде захищати свій титул.

Починаючи з другої половини 20-го століття, запрограмовані на гру в шахи комп’ютери грають зі зростальним успіхом, за силою гри набагато обійшовши найсильніших гравців серед людей. Починаючи з 1990-х років, комп’ютерний аналіз робить значний внесок у теорію шахів, особливо в ендшпілі. Комп’ютер IBM Deep Blue став першою машиною, яка здолала чинного чемпіона світу з шахів у матчі, коли він переміг Гаррі Каспарова в 1997 році. Поява сильних шахових рушіїв, які працюють на портативних

пристроях, призвела до все більшої заклопотаності щодо читерства на турнірах.

Стародавнє походження шахів не викликає сумнівів. Ігри з фішками на дошці були відомі ще в 3 – 4-му тисячоліттях до н. е. У перші п’ятсот років нової ери в Індії, за думкою багатьох істориків з’явилась найдавніша форма шахів – військова гра чатуранґа [11] буквально чотири роди військ – піхота, кіннота, слони і колісниці, представлені фігурами, які згодом розвинулися відповідно в сучасних пішака, коня, слона і туру. Грали в чатуранґу четверо гравців. Хоча чатуранґа й була складною грою, що вимагала точного розрахунку, пересування фігур визначали кидком грального кості [12]. Звідти гра поширилася на схід і захід уздовж Шовкового шляху.

Найбільш ранні свідчення про шахи зустрічаються в сусідній Сасанідській Персії близько 600 року н. е., де гра стала відомою під назвою чатранг. Мусульманський світ підхопив чатранг після ісламського завоювання Персії, після чого гра стала називатися шатрандж, а фігури здебільшого зберегли свої перські назви. На відміну від чатуранги в шатранж вже грали двоє, а не четверо суперників і ходи визначали самі гравці, а не те як випадуть кубики [12]. Найстаріші археологічні шахові артефакти, фігури зі слонової кістки, розкопано у Стародавнього Афросіабі (нинішній Самарканд в Узбекистані). Їх створено близько 760 року, а деякі, можливо, раніше. У 8–9 ст. шатрандж набув значного поширення в Арабському Халіфаті. Під час арабського періоду розвитку шахів відбувся винахід шахової нотації – способу запису партій. З’явилась велика кількість літератури з теорії дебютів, стратегії, тактики, різних типів ендшпілю [11].

Найдавніший відомий шаховий посібник написаний арабською мовою і припадає на 840–850 роки. Його написав Аль-Адлі Ар-Румі відомий арабський шахіст, під назвою «Кітаб аш-шатрандж» (книга з шахів). Цей рукопис втрачено, але посилання на нього є в пізніших працях. Поширення шахів на схід, у Китай та південно-східну Азію, ще менш задокументоване, ніж на захід. Перша китайська згадка про шахи, під назвою Сян Ці, міститься

в збірці Нариси про дива зі світу пітьмиі припадає приблизно на 800 рік. Крім того, деякі дослідники стверджують, що шахи розвинулись з китайських шахів або одного з їхніх попередників, хоча це твердження ставлять під сумнів [13].

Гра досягла Західної Європи і Русі принаймні трьома маршрутами, найстаріший припадає на 9 століття. Близько 1000 року вона поширилась по всій Європі [14, 15]. На Піренейський півострів її принесли мусульмани в 10 столітті, про що написано в знаменитому рукописі 13-го століття Трактат Альфонса Мудрого, який охоплює шатрандж, нарди і гральні кісточки. До країн західної Європи шахи потрапили не раніше 11 ст. Саме на цей час припадає перша згадка про шахи в літературі («Каталонський заповіт», 1010 року). Але гра, напевно, була відома й на 1-2 ст. раніше, про що свідчать знахідки шахових фігур, датовані 9-м ст. У Західній Європі, найвірогідніше, навчилися грі в шахи від арабів Іберійського півострова, але можливе також проникнення шахів через Аквітанію, Прованс, а також південь Апеннінського півострова. У Скандинавії та на Британських островах шахи розповсюджували вікінги, про що свідчить знахідка кількох комплектів шахових фігур на о. Льюїс, що датовані 1200 року.

Близько 1200 року правила шатранджа почали змінюватись у Південній Європі, і близько 1475 року кілька великих змін зробили гру такою, якою вона є сьогодні. Ці сучасні основні правила пересування фігур з’явились в Італії та Іспанії [16]. Спочатку католицька церква вважала шахи азартною грою, та спробувала заборонити їх, але вже в 1400 році всі заборони було знято. З 11–12 ст. шахи вважались одним із найбільш поширених розваг феодальних вельмож. Шахи навіть увійшли до програми лицарського виховання. Шахи згадуються в низці відомих літературних творів того часу («Пісня про Роланда», «Трістан та Ізольда», «Роман про Розу» та ін.). Потрапивши до Європи, шахи зазнали деяких змін, а саме шахівниця стала двоколірною (до цього всі клітини шахівниці були білого кольору). Вперше така шахівниця згадується вже в 11 ст. У другій половині 15 ст. у Західній

Європі з метою прискорення темпу гри слон та королева стали набагато сильнішими. Також було остаточно затверджено право ходу пішаком із початкової позиції на дві клітини, відповідно сучасні шахи стали називатись «ферзевими шахами» або «шахами божевільного ферзя». З'явилась рокіровка, похідна від «стрибка короля», зазвичай у поєднанні з ходом пішака або тури, щоб перемістити короля в безпечне місце. Ці нові правила швидко поширилися по всій Західній Європі.

Правила, що стосуються пата остаточно оформились на початку 19 століття. Також у 19 столітті домовились, що першими ходять білі (раніше першими могли ходити білі або чорні). Нарешті стандартизовано правила рокіровки – варіанти правила рокіровки зберігалася в Італії до кінця 19 століття. Підсумкову стандартну гру іноді називають західними шахами [17] або міжнародними шахами [18], особливо в Азії де більш поширені інші ігри шахового сімейства, такі як сянці. Починаючи з 19 століття, зміни в правилах мали тільки технічний характер, наприклад, встановлення правильного порядку в процедурі проголошення нічиєї повторенням.

Романтична епоха характеризується гамбітами у дебютах (жертвами пішаків або навіть фігур), зухвалими атаками і нахабними жертвами. Тогочасні майстри розіграли багато складних і красивих, але некоректних послідовностей ходів, званих «комбінаціями». У гру грали більше заради мистецтва, ніж теорії. Глибоке переконання, що достоїнство шахів полягає в генії гравців, а не закладено в положенні на шахівниці, пронизувало шахову практику. У 18 столітті центр європейського шахового життя перемістився з країн південної Європи до Франції. Двома найважливішими французькими майстрами були Франсуа-Андре Данікан Філідор, музикант за професією, який виявив важливість пішаків у шаховій стратегії, а пізніше Луї Шарль де Лабурдонне, який у 1834 році виграв знамениту серію матчів проти ірландського майстра Олександра Макдоннелла. Центрами шахової діяльності в цей період були кав'ярні у великих європейських містах, такі як Режанс у Парижі в Лондоні [19]. Упродовж 19-го століття шахове життя

швидко розвивалося. З’являлося багато шахових клубів, шахових книг і шахових журналів. Відбувалися матчі за листуванням між містами, наприклад, 1824 року Лондонський шаховий клуб грав проти Единбурзького шахового клубу [20]. Шахові задачі стали звичайною частиною газет 19-го століття; Бернгард Горвіц, Йозеф Клінг і Семюель Лойд складали деякі найвпливовіші композиції. 1843 року фон дер Лаза опублікував спільний з Більгером перший вичерпний підручник з шахової теорії. У 19 столітті шахам іноді дорікали, що це порожня трата часу [21].

Популярність шахів протягом останніх двох століть була тісно пов’язана з змаганням, як правило, у формі поєдинків двох гравців, за звання чемпіона світу. Назва була неофіційною до 1886 року, але широкий інтерес глядачів до гри почався більше ніж 50 років тому. Першою великою міжнародною подією була серія з шести матчів, проведених у 1834 році між провідними французькими та британськими гравцями, Луї-Шарлем де ла Бурдоне з Парижа та Олександром Макдоннелом з Лондона, які закінчилися перемогою Бурдоне. Вперше про велику шахову подію широко повідомляли в газетах і аналізували в книгах. Після смерті Бурдоне в 1840 році його змінив Стонтон після іншого матчу, який привернув міжнародну увагу, поразки Стонтона від П'єра-Шарля Фурньє де Сент-Амана з Франції в 1843 році. Цей матч також допоміг представити ідею змагання ставок, оскільки Стонтон виграв 100 фунтів стерлінгів внесли прихильники двох гравців. Стонтон використав свою позицію неофіційного чемпіона світу, щоб популяризувати набір шаблонів Стонтона, пропагувати єдиний набір правил і організувати перший міжнародний турнір, який відбувся в Лондоні в 1851 році. Карл Ернст Адольф Андерсен, німецький шкільний вчитель, був натхненний Матч Бурдоне-Макдоннел перетворився з складання задач на турнірне змагання, і він виграв лондонський турнір і з ним визнання неофіційним чемпіоном. Лондонський турнір, у свою чергу, надихнув американських гравців на організацію першого національного чемпіонату, Першого американського шахового конгресу, у Нью-Йорку в 1857 році, що

поклало початок першому шаховому захопленню в Західній півкулі. Переможець, Пол Морфі з Нового Орлеана, був визнаний неофіційним чемпіоном світу після перемоги над Андерсеном у 1858 році.

1.4 Правила гри

Сучасні правила та зовнішній вигляд творів розвивалися повільно, з широкими регіональними варіаціями. До 1300 року, наприклад, пішак набув здатності рухатися на два поля під час першого ходу, а не лише по одному, як це було в шатранджі. Але це правило не здобуло загального визнання в Європі більше 300 років. Шахи досягли найбільшого прогресу після двох важливих змін у правилах, які стали популярними після 1475 року. До того часу порадник обмежувався переміщенням одного поля по діагоналі. І оскільки пішак, який досяг восьмого рангу, міг стати лише порадником, просування пішака було відносно незначним фактором під час гри. Але згідно з новими правилами порадник змінив стать і отримав значно збільшену мобільність, щоб стати найпотужнішою фігурою на дошці – сучасною королевою. Це та підвищена цінність просування пішака додали новий динамічний елемент у шахи. Крім того, частина чатуранги під назвою слон, яка була обмежена стрибком у два квадрати по діагоналі в шатранджі, стала єпископом, більш ніж вдвічі збільшивши свій діапазон. Поки ці зміни не відбулися, мат був відносно рідкісним, і частіше партію вирішували оголенням короля. З новими повноваженнями королеви і єпископа окопна війна середньовічних шахів була замінена грою, в якій мат можна було поставити всього за два ходи. Останні дві головні зміни в правилах – рокірування та захоплення прохідним – знадобилося більше часу, щоб отримати визнання. Обидва правила були відомі в 15 столітті, але мали обмежене використання до 18 століття. Незначні зміни в інших правилах тривали до кінця 19 століття; наприклад, у багатьох частинах Європи навіть у

середині 19 століття було неприйнятно підвищувати пішака до ферзя, якщо гравець все ще мав початкового ферзя.

Гра у шахи ведеться двома гравцями за дошкою, що має 64 клітини чорного та білого кольору, фігурами білого та чорного кольорів. Шахова дошка зазвичай називається шахівниця. Шахівниця розміщується таким чином, що її перша горизонталь знаходиться біля гравця, що грає білими. Кожен з гравців має 16 шахових фігур: вісім пішаків, дві тури, два коня, два слона, ферзь та король. Гравці роблять ходи по черзі, один за одним. Під час кожного ходу гравець може перемістити лише одну фігуру. Виключенням є рокіровка – спільний хід турою та королем. Гру розпочинає той з гравців, що грає фігурами білого кольору. При переміщенні своєї фігури на клітину зайняту фігурою суперника, фігура суперника знімається з дошки. Такий хід називають взяття. Правила пересування фігур:

– Пішак пересувається лише вперед по вертикалі. Перший хід пішак – білий з другої, а чорний з сьомої горизонталі – може зробити одразу на дві клітини, перестрибнувши через третю або шосту горизонталь. З усіх інших горизонталей пішак може пересуватися лише на одну клітину вперед. Крім цього пішак може взяти будь-яку фігуру супротивника, що знаходиться попереду нього на відстані однієї клітини по діагоналі. Якщо білий пішак досягнув восьмої горизонталі, або чорний – першої то він може перетворитися на будь-яку фігуру окрім короля того ж кольору що й пішак;

– Тура пересувається на будь-яку кількість клітин по вертикалі, або горизонталі;

– Кінь єдина фігура у грі, яка може перестрибувати через інші фігури – свої та суперника. Хід конем схожий на літеру «Г». Кінь пересувається на одну клітину по вертикалі та дві по горизонталі, або навпаки – на дві по вертикалі, та одну по горизонталі у будь-якому напрямку;

– Слон пересувається на будь-яку кількість клітин по діагоналі;

– Король пересувається на одну клітину по вертикалі, горизонталі або діагоналі;

– Ферзь пересувається на будь-яку кількість клітин по вертикалі, горизонталі або діагоналі. Додаткові правила та виключення при виконанні ходів;

– не дозволяється робити хід, після якого король гравця, що ходив, опиняється під атакою фігур суперника;

– не можна робити хід на клітину, що зайнята власною фігурою;

– фігури не можуть перестрибувати через власні фігури та фігури суперника(єдиний виняток – кінь);

– спеціальний хід рокірування виконується королем та турою. Цей хід можливий за виконання двох умов: король та тура до цього жодного разу не ходили, король та тура після виконання рокірування не повинні знаходитися під атакою фігур суперника. Рокірування виконується наступним чином: король пересувається по горизонталі на дві клітини. Тура після цього встановлюється на клітину через яку перестрибнув король під час пересування;

– взяття на проході – правило згідно якого, якщо пішак робить хід на дві клітини та при цьому перестрибує через клітину, яку атакує пішак супротивника, він може бути вбитий пішаком супротивника. При цьому пішак супротивника пересувається на клітину, через яку перестрибнув пішак. Деякі шахові терміни:

– шах становище під час якого король того гравця, чия черга робити хід знаходиться під атакою фігур суперника;

– мат, шах від якого не має захисту. Гравець, який поставив мат королю суперника виграє партію;

– пат становище, коли жодна з фігур того гравця, чия черга робити хід не може зробити хід дозволений правилами. Після встановлення на шахівниці пату партія оголошується нічиєю. Партія оголошується виграною одним із суперників якщо:

– оголошено мат королю суперника;

– суперник визнав себе переможеним;

– технічна перемога (суперник не з’явився на гру на домовлений час; суперник розпочав гру, але відмовився грати; суперник грубо порушив правила гри) Партія оголошується нічиєю якщо:

– на шахівниці виникло патове становище;

– за взаємною згодою гравців;

– при наявності «вічного шаху»;

– на дошці залишилися лише королі і, можливо, у одного або обох гравців є одна легка фігура (слон або кінь);

– має місце трикратне повторення позиції. Правило «доторкнувся – ходи»

– гравець, що при своєму ходу доторкнувся до своєї фігури повинен нею на цьому ході ходити, а до чужої – взяти;

– якщо це неможливо, дотик не має ніяких наслідків;

– якщо гравець хоче поправити фігуру на дошці, він має попередньо сказати «поправляю» і лише після цього поправити її. У такому випадку доторк не має ніяких наслідків;

– рокірування слід починати з короля. Якщо при рокіруванні гравець спершу торкнувся до тури, то має ходити нею, а право рокірування втрачається [22].

1.5 Постановка задачі

Об’єктом роботи є розробка гри «Шахи».

Метою кваліфікаційної роботи є розроблення гри «Шахи» з використанням простого штучного інтелекту для гри проти комп’ютера.

Нижче наведені вимоги користувача до даних і функціональності програми:

– користувач може вибрати режим гри : «Проти комп’ютера»;

– користувач може вибрати режим гри «Один на один».

Для досягнення мети кваліфікаційної роботи необхідно вирішити наступні завдання:

– вивчити такі теоретичні питання: теорію програмування на мові Python, теорію навчання комп’ютера у цю гру на мові Python, теорію о роботі з графічними інтерфейсами мови Python, теорію о створенні дизайну сторінок у Photoshop;

– ознайомитися з наступними інструментами: PyCharm;

– розробити дизайн головного меню, ігрового поля, сторінки допомоги та інше;

– застосувати отримані теоретичні та практичні знання;

– розробити програмний код для гри «Шахи»;

– розробити програмний код для штучного інтелекту;

– провести тестування гри;

– відобразити процес і результати розробки гри в пояснювальній записці до кваліфікаційної роботи.

Для розробки використані такі інструменти:

– Photoshop – середовище для розробки дизайну гри;

– PyCharm – середовище розробки програмного забезпечення;

– Python – мова програмування;

– PyGame – бібліотека для розробки ігор.

2 ОПИС ТЕХНОЛОГІЙ ТА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ БАЗОВОГО ФУНКЦІОНАЛУ

2.1 Опис використаних технологій

Python – інтерпретована об’єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв’язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об’єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [23].

Pygame – набір крос-платформових модулів для мови програмування Python, створений для розробки відеоігор. Включає в себе бібліотеки комп’ютерної графіки і звуку на базі SDL [24].

PyCharm – інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веброзробку на Django. PyCharm працює під операційними системами Windows, Mac OS X і Linux [25].

Adobe Photoshop – графічний редактор, розроблений і поширюваний фірмою Adobe Systems. Цей продукт є лідером ринку в галузі комерційних засобів редагування растрових зображень і найвідомішим продуктом фірми Adobe. Часто цю програму називають просто Photoshop (Фотошоп). У наш час Photoshop доступний на платформах Mac OS X/Mac OS і Microsoft Windows. Ранні версії редактора були портовані під SGI IRIX, але офіційна підтримка була припинена, починаючи з третьої версії продукту. Для версії

CS і CS6 можливий запуск під Linux за допомогою альтернативи Windows API – Wine [26].

Artificial intelligence – штучний інтелект (ШІ) дає можливість машинам вчитися на досвіді, пристосовуватися до нових даних і виконувати завдання, схожі на людину. Більшість прикладів штучного інтелекту, про які ви чуєте сьогодні – від комп'ютерів для гри в шахи до самокерованих автомобілів – значною мірою покладаються на глибоке навчання та обробку природної мови. Використовуючи ці технології, комп'ютери можна навчити виконувати конкретні завдання, обробляючи великі обсяги даних і розпізнаючи закономірності в даних [27].

2.2 Програмна реалізація базового функціоналу

Для того щоб створити гру потрібно визначитися що нам для цього потрібно, розробити дизайн, та написати код, потрібно розробити:

– дизайн дошки (рис. 2.1);

– дизайн фігур чорних (рис. 2.2);

– дизайн фігур білих (рис. 2.3);

– ігрову логіку;

– логіку фігур;

– логіку дошки;

– валідацію ходів;

– ігрове меню;

– вибір режиму.

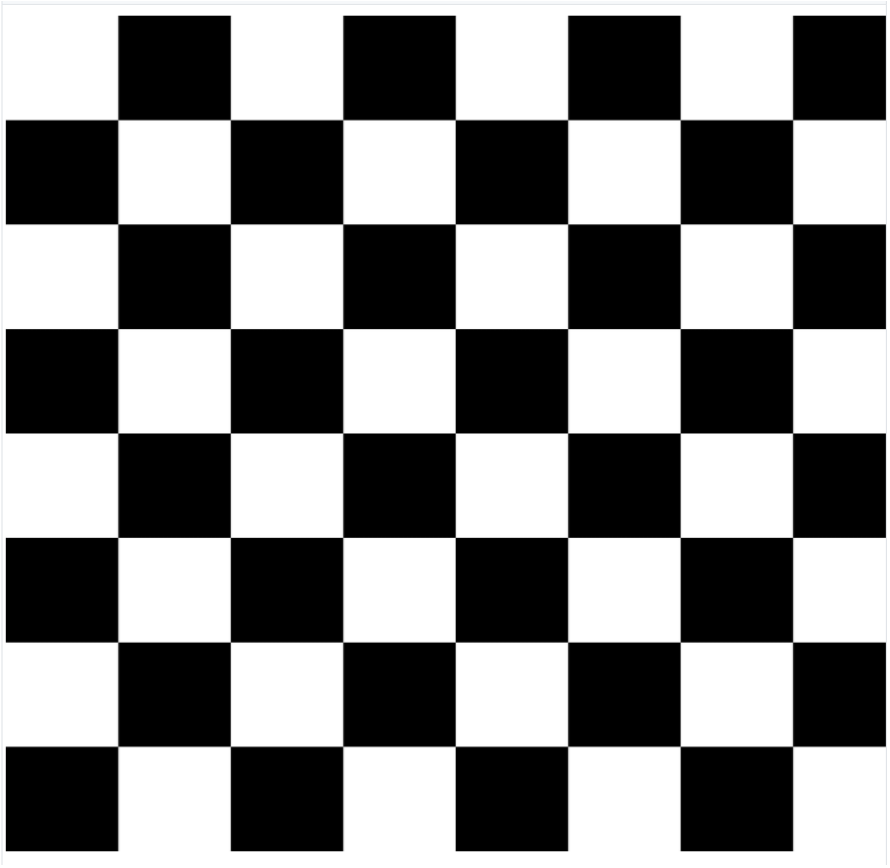

Рисунок 2.1 – Дизайн дошки

Рисунок 2.2 – Дизайн чорних фігур

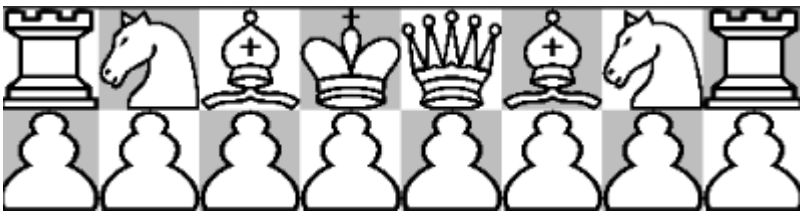

Рисунок 2.3 – Дизайн білих фігур

Для початку, потрібно створити основні змінні:

– фігури (рис. 2.4);

– дошка (рис. 2.5).

```
class Player:
    PLAYER_1 = 'white'
    PLAYER_2 = 'black'
    EMPTY = -9
    PIECES = ['white_r', 'white_n', 'white_b', 'white_q', 'white_k', 'white_p',
              'black_r', 'black_n', 'black_b', 'black_q', 'black_k', 'black_p']
```

Рисунок 2.4 – Змінні гравця та положення фігур

```
r \ c     0          1          2          3          4          5          6          7
0   [(r=0, c=0), (r=0, c=1), (r=0, c=2), (r=0, c=3), (r=0, c=4), (r=0, c=5), (r=0, c=6), (r=0, c=7)]
1   [(r=1, c=0), (r=1, c=1), (r=1, c=2), (r=1, c=3), (r=1, c=4), (r=1, c=5), (r=1, c=6), (r=1, c=7)]
2   [(r=2, c=0), (r=2, c=1), (r=2, c=2), (r=2, c=3), (r=2, c=4), (r=2, c=5), (r=2, c=6), (r=2, c=7)]
3   [(r=3, c=0), (r=3, c=1), (r=3, c=2), (r=3, c=3), (r=3, c=4), (r=3, c=5), (r=3, c=6), (r=3, c=7)]
4   [(r=4, c=0), (r=4, c=1), (r=4, c=2), (r=4, c=3), (r=4, c=4), (r=4, c=5), (r=4, c=6), (r=4, c=7)]
5   [(r=5, c=0), (r=5, c=1), (r=5, c=2), (r=5, c=3), (r=5, c=4), (r=5, c=5), (r=5, c=6), (r=5, c=7)]
6   [(r=6, c=0), (r=6, c=1), (r=6, c=2), (r=6, c=3), (r=6, c=4), (r=6, c=5), (r=6, c=6), (r=6, c=7)]
7   [(r=7, c=0), (r=7, c=1), (r=7, c=2), (r=7, c=3), (r=7, c=4), (r=7, c=5), (r=7, c=6), (r=7, c=7)]
```

Рисунок 2.5 – Дошка

Потім потрібно створити ігрове середовище – шахову дошку, яка буде зберегати стан гри.

Шахівниця — ігрове поле в шахах, шашках і деяких інших стратегічних настільних іграх. Традиційна шахівниця є полем 8×8 (усього 64) темних і світлих клітинок (полів), що чергуються. Поле «a1» традиційно чорне. У різних варіантах кількість полів може варіюватися, з радикальніших відступів — можлива зміна форми полів. Незмінна риса шахівниці — двоколірність [28].

У традиційному настільному (не комп'ютерному й не портативному) форм-факторі шахівниця дерев'яна. Зазвичай це коробка, що складається навпіл, в якій зберігаються шахові фігури.

Проініціалізувати білі та чорні фігури [29], проініціалізувати дошку, написати логіку та валідацію ходів фігур, прописати усі можливі ходи фігур.

Створено класс який буде відповідати за стан гри, мати всю ігрову логіку та назвемо його «game_state» та головну ініціалізуючу функцію «__init__ » (рис. 2.6) у якій буде реалізована ігрова логіка, фігури, дошка, валідація ходів, можливі ходи.

```
class game_state:
    # Initialize 2D array to represent the chess board
    def __init__(self):
        # The board is a 2D array
        # TODO: Change to a numpy format later
        self.white_captives = []
        self.black_captives = []
        self.move_log = []
        self.white_turn = True
        self.can_en_passant_bool = False
        self._en_passant_previous = (-1, -1)
        self.checkmate = False
        self.stalemate = False

        self._is_check = False
        self._white_king_location = [0, 3]
        self._black_king_location = [7, 3]

        self.white_king_can_castle = [True, True,
                                      True]  # Has king not moved, has Rook1(col=0) not moved, has Rook2(col=7) not moved
        self.black_king_can_castle = [True, True, True]
```

Рисунок 2.6 – Ініціалізація классу «game_state» та функції «__init__ »

Тепер потрібно ініціалізувати білі фігури (рис. 2.7) та чорні (рис. 2.8).

```
# Initialize White pieces
white_rook_1 = Rook('r', 0, 0, Player.PLAYER_1)
white_rook_2 = Rook('r', 0, 7, Player.PLAYER_1)
white_knight_1 = Knight('n', 0, 1, Player.PLAYER_1)
white_knight_2 = Knight('n', 0, 6, Player.PLAYER_1)
white_bishop_1 = Bishop('b', 0, 2, Player.PLAYER_1)
white_bishop_2 = Bishop('b', 0, 5, Player.PLAYER_1)
white_queen = Queen('q', 0, 4, Player.PLAYER_1)
white_king = King('k', 0, 3, Player.PLAYER_1)
white_pawn_1 = Pawn('p', 1, 0, Player.PLAYER_1)
white_pawn_2 = Pawn('p', 1, 1, Player.PLAYER_1)
white_pawn_3 = Pawn('p', 1, 2, Player.PLAYER_1)
white_pawn_4 = Pawn('p', 1, 3, Player.PLAYER_1)
white_pawn_5 = Pawn('p', 1, 4, Player.PLAYER_1)
white_pawn_6 = Pawn('p', 1, 5, Player.PLAYER_1)
white_pawn_7 = Pawn('p', 1, 6, Player.PLAYER_1)
white_pawn_8 = Pawn('p', 1, 7, Player.PLAYER_1)
self.white_pieces = [white_rook_1, white_rook_2, white_knight_1, white_knight_2, white_bishop_1, white_bishop_2,
                     white_queen, white_king, white_pawn_1, white_pawn_2, white_pawn_3, white_pawn_4,
                     white_pawn_5,
                     white_pawn_6, white_pawn_7, white_pawn_8]
```

Рисунок 2.7 – Ініціалізація білих фігур

```
# Initialize Black Pieces
black_rook_1 = Rook('r', 7, 0, Player.PLAYER_2)
black_rook_2 = Rook('r', 7, 7, Player.PLAYER_2)
black_knight_1 = Knight('n', 7, 1, Player.PLAYER_2)
black_knight_2 = Knight('n', 7, 6, Player.PLAYER_2)
black_bishop_1 = Bishop('b', 7, 2, Player.PLAYER_2)
black_bishop_2 = Bishop('b', 7, 5, Player.PLAYER_2)
black_queen = Queen('q', 7, 4, Player.PLAYER_2)
black_king = King('k', 7, 3, Player.PLAYER_2)
black_pawn_1 = Pawn('p', 6, 0, Player.PLAYER_2)
black_pawn_2 = Pawn('p', 6, 1, Player.PLAYER_2)
black_pawn_3 = Pawn('p', 6, 2, Player.PLAYER_2)
black_pawn_4 = Pawn('p', 6, 3, Player.PLAYER_2)
black_pawn_5 = Pawn('p', 6, 4, Player.PLAYER_2)
black_pawn_6 = Pawn('p', 6, 5, Player.PLAYER_2)
black_pawn_7 = Pawn('p', 6, 6, Player.PLAYER_2)
black_pawn_8 = Pawn('p', 6, 7, Player.PLAYER_2)
self.black_pieces = [black_rook_1, black_rook_2, black_knight_1, black_knight_2, black_bishop_1, black_bishop_2,
                     black_queen, black_king, black_pawn_1, black_pawn_2, black_pawn_3, black_pawn_4,
                     black_pawn_5,
                     black_pawn_6, black_pawn_7, black_pawn_8]
```

Рисунок 2.8 – Ініціалізація чорних фігур

Після цього потрібно розробити усі можливі ходи для фігур щоб вони могли переміщятися, для цього ініціалізована функція (рис. 2.9)

```
# Move a piece
def move_piece(self, starting_square, ending_square, is_ai):
    current_square_row = starting_square[0]  # The integer row value of the starting square
    current_square_col = starting_square[1]  # The integer col value of the starting square
    next_square_row = ending_square[0]  # The integer row value of the ending square
    next_square_col = ending_square[1]  # The integer col value of the ending square

    if self.is_valid_piece(current_square_row, current_square_col) and \
            (((self.whose_turn() and self.get_piece(current_square_row, current_square_col).is_player(
                Player.PLAYER_1)) or
              (not self.whose_turn() and self.get_piece(current_square_row, current_square_col).is_player(
                  Player.PLAYER_2)))):

        # The chess piece at the starting square
        moving_piece = self.get_piece(current_square_row, current_square_col)

        valid_moves = self.get_valid_moves(starting_square)

        temp = True

        if ending_square in valid_moves:
            moved_to_piece = self.get_piece(next_square_row, next_square_col)
            if moving_piece.get_name() is "k":
                if moving_piece.is_player(Player.PLAYER_1):
                    if moved_to_piece == Player.EMPTY and next_square_col == 1 and self.king_can_castle_left(
                            moving_piece.get_player()):
                        move = chess_move(starting_square, ending_square, self, self._is_check)
                        move.castling_move((0, 0), (0, 2), self)
                        self.move_log.append(move)

                        # move rook
                        self.get_piece(0, 0).change_col_number(2)

                        self.board[0][2] = self.board[0][0]
                        self.board[0][0] = Player.EMPTY

                        self.white_king_can_castle[0] = False
                        self.white_king_can_castle[1] = False

                    elif moved_to_piece == Player.EMPTY and next_square_col == 5 and self.king_can_castle_right(
                            moving_piece.get_player()):
                        move = chess_move(starting_square, ending_square, self, self._is_check)
                        move.castling_move((0, 7), (0, 4), self)
                        self.move_log.append(move)
                        # move rook
                        self.get_piece(0, 7).change_col_number(4)

                        self.board[0][4] = self.board[0][7]
                        self.board[0][7] = Player.EMPTY

                        self.white_king_can_castle[0] = False
                        self.white_king_can_castle[2] = False
                    else:
                        move = chess_move(starting_square, ending_square, self, self._is_check)
                        self.move_log.append(move)
                        self.white_king_can_castle[0] = False
                    self._white_king_location = (next_square_row, next_square_col)
                else:
```

Рисунок 2.9 – Функція «move_piece» (частина 1)

```
        else:
            if moved_to_piece == Player.EMPTY and next_square_col == 1 and self.king_can_castle_left(
                    moving_piece.get_player()):
                move = chess_move(starting_square, ending_square, self, self._is_check)
                move.castling_move((7, 0), (7, 2), self)
                self.move_log.append(move)

                self.get_piece(7, 0).change_col_number(2)
                # move rook
                self.board[7][2] = self.board[7][0]
                self.board[7][0] = Player.EMPTY

                self.black_king_can_castle[0] = False
                self.black_king_can_castle[1] = False
            elif moved_to_piece == Player.EMPTY and next_square_col == 5 and self.king_can_castle_right(
                    moving_piece.get_player()):
                move = chess_move(starting_square, ending_square, self, self._is_check)
                move.castling_move((7, 7), (7, 4), self)
                self.move_log.append(move)

                self.get_piece(0, 7).change_col_number(4)

                # move rook
                self.board[7][4] = self.board[7][7]
                self.board[7][7] = Player.EMPTY

                self.black_king_can_castle[0] = False
                self.black_king_can_castle[2] = False
            else:
                move = chess_move(starting_square, ending_square, self, self._is_check)
                self.move_log.append(move)
                self.black_king_can_castle[0] = False
            self._black_king_location = (next_square_row, next_square_col)
            # self.can_en_passant_bool = False  WHAT IS THIS
    elif moving_piece.get_name() is "r":
        if moving_piece.is_player(Player.PLAYER_1) and current_square_col == 0:
            self.white_king_can_castle[1] = False
        elif moving_piece.is_player(Player.PLAYER_1) and current_square_col == 7:
            self.white_king_can_castle[2] = False
        elif moving_piece.is_player(Player.PLAYER_2) and current_square_col == 0:
            self.white_king_can_castle[1] = False
        elif moving_piece.is_player(Player.PLAYER_2) and current_square_col == 7:
            self.white_king_can_castle[2] = False
        self.move_log.append(chess_move(starting_square, ending_square, self, self._is_check))
        self.can_en_passant_bool = False
    # Add move class here
    elif moving_piece.get_name() is "p":
        # Promoting white pawn
        if moving_piece.is_player(Player.PLAYER_1) and next_square_row == 7:
            # print("promoting white pawn")
            if is_ai:
                self.promote_pawn_ai(starting_square, moving_piece, ending_square)
            else:
                self.promote_pawn(starting_square, moving_piece, ending_square)
            temp = False
        # Promoting black pawn
        elif moving_piece.is_player(Player.PLAYER_2) and next_square_row == 0:
            # print("promoting black pawn")
            if is_ai:
                self.promote_pawn_ai(starting_square, moving_piece, ending_square)
```

Рисунок 2.9 – Функція «move_piece» (частина 2)

```
                    self.promote_pawn(starting_square, moving_piece, ending_square)
                temp = False
            # Promoting black pawn
            elif moving_piece.is_player(Player.PLAYER_2) and next_square_row == 0:
                # print("promoting black pawn")
                if is_ai:
                    self.promote_pawn_ai(starting_square, moving_piece, ending_square)
                else:
                    self.promote_pawn(starting_square, moving_piece, ending_square)
                temp = False
            # Moving pawn forward by two
            # Problem with Pawn en passant ai
            elif abs(next_square_row - current_square_row) == 2 and current_square_col == next_square_col:
                # print("move pawn forward")
                self.move_log.append(chess_move(starting_square, ending_square, self, self._is_check))
                # self.can_en_passant_bool = True
                self._en_passant_previous = (next_square_row, next_square_col)
            # en passant
            elif abs(next_square_row - current_square_row) == 1 and abs(
                    current_square_col - next_square_col) == 1 and \
                    self.can_en_passant(current_square_row, current_square_col):
                # print("en passant")
                if moving_piece.is_player(Player.PLAYER_1):
                    move = chess_move(starting_square, ending_square, self, self._is_check)
                    move.en_passant_move(self.board[next_square_row - 1][next_square_col],
                                         (next_square_row - 1, next_square_col))
                    self.move_log.append(move)
                    self.board[next_square_row - 1][next_square_col] = Player.EMPTY
                else:
                    move = chess_move(starting_square, ending_square, self, self._is_check)
                    move.en_passant_move(self.board[next_square_row + 1][next_square_col],
                                         (next_square_row + 1, next_square_col))
                    self.move_log.append(move)
                    self.board[next_square_row + 1][next_square_col] = Player.EMPTY
            # moving forward by one or taking a piece
            else:
                self.move_log.append(chess_move(starting_square, ending_square, self, self._is_check))
                self.can_en_passant_bool = False
        else:
            self.move_log.append(chess_move(starting_square, ending_square, self, self._is_check))
            self.can_en_passant_bool = False

        if temp:
            moving_piece.change_row_number(next_square_row)
            moving_piece.change_col_number(next_square_col)
            self.board[next_square_row][next_square_col] = self.board[current_square_row][current_square_col]
            self.board[current_square_row][current_square_col] = Player.EMPTY

        self.white_turn = not self.white_turn

    else:
        pass
```

Рисунок 2.9 – Функція «move_piece» (частина 3)

Була написана функція для можливих ходів фігур, також проініціалізовано класс «chess_move», який буде давати можливість переміщати фігури (рис. 2.10).

```
class chess_move():
    def __init__(self, starting_square, ending_square, game_state, in_check):
        self.starting_square_row = starting_square[0]
        self.starting_square_col = starting_square[1]
        self.moving_piece = game_state.get_piece(self.starting_square_row, self.starting_square_col)
        self.in_check = in_check

        self.ending_square_row = ending_square[0]
        self.ending_square_col = ending_square[1]
        if game_state.is_valid_piece(self.ending_square_row, self.ending_square_col):
            self.removed_piece = game_state.get_piece(self.ending_square_row, self.ending_square_col)
        else:
            self.removed_piece = Player.EMPTY

        self.castled = False
        self.rook_starting_square = None
        self.rook_ending_square = None
        self.moving_rook = None

        self.pawn_promoted = False
        self.replacement_piece = None

        self.en_passaned = False
        self.en_passant_eaten_piece = None
        self.en_passant_eaten_square = None

    def castling_move(self, rook_starting_square, rook_ending_square, game_state):
        self.castled = True
        self.rook_starting_square = rook_starting_square
        self.rook_ending_square = rook_ending_square
        self.moving_rook = game_state.get_piece(rook_starting_square[0], rook_starting_square[1])

    def pawn_promotion_move(self, new_piece):
        self.pawn_promoted = True
        self.replacement_piece = new_piece

    def en_passant_move(self, eaten_piece, eaten_piece_square):
        self.en_passaned = True
        self.en_passant_eaten_piece = eaten_piece
        self.en_passant_eaten_square = eaten_piece_square

    def get_moving_piece(self):
        return self.moving_piece
```

Рисунок 2.10 – Клас «chess_move»

Також не слід забувати про валідацію ходів, у кожної фігури є своя траєкторія (рис. 2.11).

```
        def get_piece(self, row, col):
            if (0 <= row < 8) and (0 <= col < 8):
                return self.board[row][col]

        def is_valid_piece(self, row, col):
            evaluated_piece = self.get_piece(row, col)
            return (evaluated_piece is not None) and (evaluated_piece != Player.EMPTY)


def get_valid_moves(self, starting_square):
    '''
    remove pins from valid moves (unless the pinned piece move can get rid of a check and checks is empty
    remove move from valid moves if the move falls within a check piece's valid move
    if the moving piece is a king, the ending square cannot be in a check
    '''

    current_row = starting_square[0]
    current_col = starting_square[1]

    if self.is_valid_piece(current_row, current_col):
        valid_moves = []
        moving_piece = self.get_piece(current_row, current_col)
        if self.get_piece(current_row, current_col).is_player(Player.PLAYER_1):
            king_location = self._white_king_location
        else:
            king_location = self._black_king_location
        group = self.check_for_check(king_location, moving_piece.get_player())
        checking_pieces = group[0]
        pinned_pieces = group[1]
        pinned_checks = group[2]
        initial_valid_piece_moves = moving_piece.get_valid_piece_moves(self)

        # immediate check
        if checking_pieces:
            for move in initial_valid_piece_moves:
                can_move = True
                for piece in checking_pieces:
                    if moving_piece.get_name() is "k":
                        temp = self.board[current_row][current_col]
                        self.board[current_row][current_col] = Player.EMPTY
                        temp2 = self.board[move[0]][move[1]]
                        self.board[move[0]][move[1]] = temp
                        if not self.check_for_check(move, moving_piece.get_player())[0]:
                            pass
```

Рисунок 2.11 – Валідація ходів (частина 1)

```
                else:
                    can_move = False
                self.board[current_row][current_col] = temp
                self.board[move[0]][move[1]] = temp2
            elif move == piece and len(checking_pieces) == 1 and moving_piece.get_name() is not "k" and \
                    (current_row, current_col) not in pinned_pieces:
                pass
            elif move != piece and len(checking_pieces) == 1 and moving_piece.get_name() is not "k" and \
                    (current_row, current_col) not in pinned_pieces:
                temp = self.board[move[0]][move[1]]
                self.board[move[0]][move[1]] = moving_piece
                self.board[current_row][current_col] = Player.EMPTY
                if self.check_for_check(king_location, moving_piece.get_player())[0]:
                    can_move = False
                self.board[current_row][current_col] = moving_piece
                self.board[move[0]][move[1]] = temp
            else:
                can_move = False
        if can_move:
            valid_moves.append(move)
    self._is_check = True
# pinned checks
elif pinned_pieces and moving_piece.get_name() is not "k":
    if starting_square not in pinned_pieces:
        for move in initial_valid_piece_moves:
            valid_moves.append(move)
    elif starting_square in pinned_pieces:
        for move in initial_valid_piece_moves:

            temp = self.board[move[0]][move[1]]
            self.board[move[0]][move[1]] = moving_piece
            self.board[current_row][current_col] = Player.EMPTY
            if not self.check_for_check(king_location, moving_piece.get_player())[0]:
                valid_moves.append(move)
            self.board[current_row][current_col] = moving_piece
            self.board[move[0]][move[1]] = temp

    else:
        if moving_piece.get_name() is "k":
            for move in initial_valid_piece_moves:
                temp = self.board[current_row][current_col]
                temp2 = self.board[move[0]][move[1]]
                self.board[current_row][current_col] = Player.EMPTY
                self.board[move[0]][move[1]] = temp
                if not self.check_for_check(move, moving_piece.get_player())[0]:
                    valid_moves.append(move)
                self.board[current_row][current_col] = temp
                self.board[move[0]][move[1]] = temp2
        else:
            for move in initial_valid_piece_moves:
                valid_moves.append(move)
    # if not valid_moves:
    #     if self._is_check:
    #         self.checkmate = True
    #     else:
    #         self.stalemate = True
    # else:
    #     self.checkmate = False
    #     self.stalemate = False
    return valid_moves
else:
    return None
```

Рисунок 2.11 – Валідація ходів (частина 2)

Було ініціалізовано функцію яка дасть можливість вибирати гравцю проти кого грати, та буде розпочинати партію або закінчувати її (рис. 2.12).

```
def main():
    # Check for the number of players and the color of the AI
    human_player = ""
    while True:
        try:
            number_of_players = input("How many players (1 or 2)?\n")
            if int(number_of_players) == 1:
                number_of_players = 1
                while True:
                    human_player = input("What color do you want to play (w or b)?\n")
                    if human_player is "w" or human_player is "b":
                        break
                    else:
                        print("Enter w or b.\n")
                break
            elif int(number_of_players) == 2:
                number_of_players = 2
                break
            else:
                print("Enter 1 or 2.\n")
        except ValueError:
            print("Enter 1 or 2.")

    py.init()
    screen = py.display.set_mode((WIDTH, HEIGHT))
    clock = py.time.Clock()
    game_state = chess_engine.game_state()
    load_images()
    running = True
    square_selected = ()  # keeps track of the last selected square
    player_clicks = []  # keeps track of player clicks (two tuples)
    valid_moves = []
    game_over = False

    ai = ai_engine.chess_ai()
    game_state = chess_engine.game_state()
    if human_player is 'b':
        ai_move = ai.minimax_black(game_state, 3, -100000, 100000, True, Player.PLAYER_1)
        game_state.move_piece(ai_move[0], ai_move[1], True)
```

Рисунок 2.12 – функція «Main()» (частина 1)

```
while running:
    for e in py.event.get():
        if e.type == py.QUIT:
            running = False
        elif e.type == py.MOUSEBUTTONDOWN:
            if not game_over:
                location = py.mouse.get_pos()
                col = location[0] // SQ_SIZE
                row = location[1] // SQ_SIZE
                if square_selected == (row, col):
                    square_selected = ()
                    player_clicks = []
                else:
                    square_selected = (row, col)
                    player_clicks.append(square_selected)
                if len(player_clicks) == 2:
                    # this if is useless right now
                    if (player_clicks[1][0], player_clicks[1][1]) not in valid_moves:
                        square_selected = ()
                        player_clicks = []
                        valid_moves = []
                    else:
                        game_state.move_piece((player_clicks[0][0], player_clicks[0][1]),
                                              (player_clicks[1][0], player_clicks[1][1]), False)
                        square_selected = ()
                        player_clicks = []
                        valid_moves = []

                        if human_player is 'w':
                            ai_move = ai.minimax_white(game_state, 3, -100000, 100000, True, Player.PLAYER_2)
                            game_state.move_piece(ai_move[0], ai_move[1], True)
                        elif human_player is 'b':
                            ai_move = ai.minimax_black(game_state, 3, -100000, 100000, True, Player.PLAYER_1)
                            game_state.move_piece(ai_move[0], ai_move[1], True)
                else:
                    valid_moves = game_state.get_valid_moves((row, col))
                    if valid_moves is None:
                        valid_moves = []
```

Рисунок 2.12 – функція «Main()» (частина 2)

```
            elif e.type == py.KEYDOWN:
                if e.key == py.K_r:
                    game_over = False
                    game_state = chess_engine.game_state()
                    valid_moves = []
                    square_selected = ()
                    player_clicks = []
                    valid_moves = []
                elif e.key == py.K_u:
                    game_state.undo_move()
                    print(len(game_state.move_log))

        draw_game_state(screen, game_state, valid_moves, square_selected)

        endgame = game_state.checkmate_stalemate_checker()
        if endgame == 0:
            game_over = True
            draw_text(screen, "Black wins.")
        elif endgame == 1:
            game_over = True
            draw_text(screen, "White wins.")
        elif endgame == 2:
            game_over = True
            draw_text(screen, "Stalemate.")

        clock.tick(MAX_FPS)
        py.display.flip()


def draw_text(screen, text):
    font = py.font.SysFont("Helvitca", 32, True, False)
    text_object = font.render(text, False, py.Color("Black"))
    text_location = py.Rect(0, 0, WIDTH, HEIGHT).move(WIDTH / 2 - text_object.get_width() / 2,
                                                      HEIGHT / 2 - text_object.get_height() / 2)
    screen.blit(text_object, text_location)


if __name__ == "__main__":
    main()
```

Рисунок 2.12 – функція «Main()» (частина 3)

Коли було закінчене написання програмного коду, логіки гри, та розробку дизайна, можна запустити гру та насолоджуватися шахами у режимі «один на один» (рис. 2.13, рис. 2.14).

Рисунок 2.13 – Гравець 1 грає білими

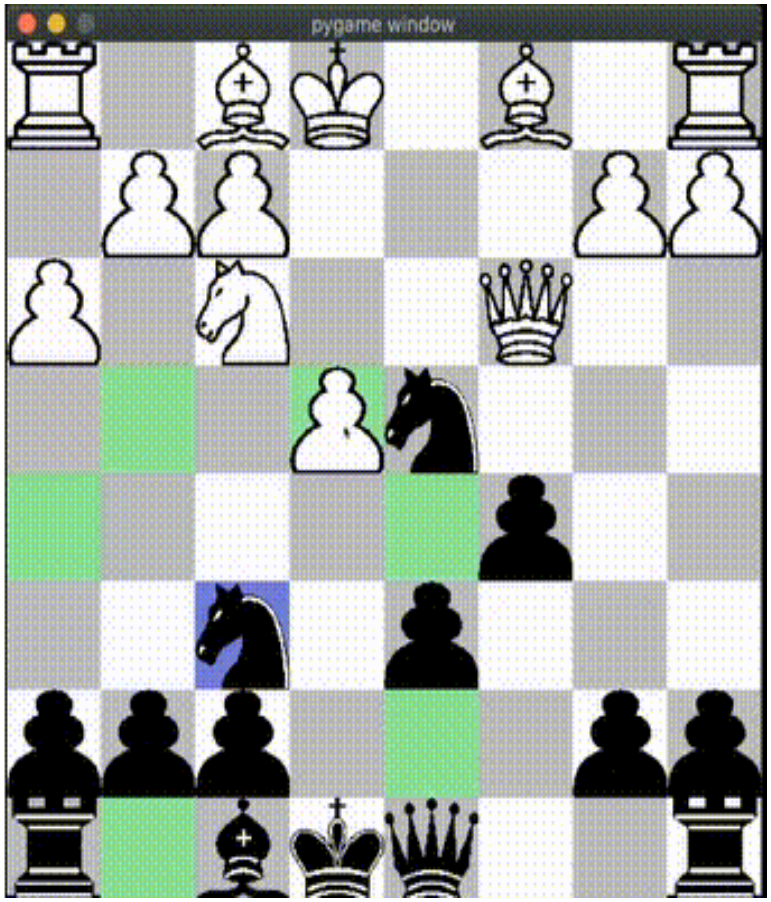

Рисунок 2.14 – Гравець 2 грає чорними

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Алгоритм «мінімакса» для штучного інтелекту

«Мінімаксний» алгоритм спирається на систематичний пошук, чи точніше сказати – на грубу силу та просту функцію оцінки. Припустимо, що кожного разу, вирішуючи наступний хід, шукаємо все дерево, аж до листя. Фактично ми розглядали всі можливі результати і щоразу могли б визначити найкращий можливий хід.

Однак для нетривіальних ігор ця практика не застосовується. Навіть пошук на певну глибину іноді займає неприйнятну кількість часу. Тому «Minimax» застосовує пошук до досить низької глибини дерева за допомогою відповідної евристики та добре розробленої, але простої оцінної функції.

При такому підході ми втрачаємо впевненість у пошуку найкращого можливого ходу, але в більшості випадків рішення, яке приймає «мінімакс», набагато краще, ніж у будь-якої людини.

Тепер давайте розглянемо функцію оцінки, про яку ми вже згадували. Щоб визначити хороший (не обов’язково найкращий) хід для певного гравця, ми маємо якимось чином оцінити вузли (позиції), щоб мати можливість порівнювати один з одним за якістю.

Функція оцінки-це статичне число, яке відповідно до характеристик самої гри присвоюється кожному вузлу (позиції).

Важливо відзначити, що функція оцінки не повинна покладатися на пошук попередніх вузлів, ні на пошук наступних. Він повинен просто аналізувати стан гри та обставини, в яких перебувають обидва гравці.

Необхідно, щоб оцінна функція містила якнайбільше релевантної інформації, але з іншого боку – оскільки вона обчислюється багато разів – вона має бути простою.

В іграх з нульовою сумою значення оцінної функції має протилежне значення - що краще для першого гравця, то гірше для другого, і навпаки.

Отже, значення для симетричних позицій (якщо гравці змінюються ролями) має відрізнятися лише за знаком.

Загальноприйнятою практикою є зміна оцінок листя шляхом віднімання глибини цього точного аркуша, щоб з усіх ходів, що ведуть до перемоги, алгоритм міг вибрати той, який робить це за найменше кроків (або вибирає хід, який відкладає програш, якщо він неминучий).

Ось проста ілюстрація кроків «Мінімаксу». У разі ми шукаємо мінімальне значення. Зелений шар викликає метод «Max()» на вузлах у дочірніх вузлах, а червоний шар викликає метод «Main()» на дочірніх вузлах (рис. 3.1, рис. 3.2).

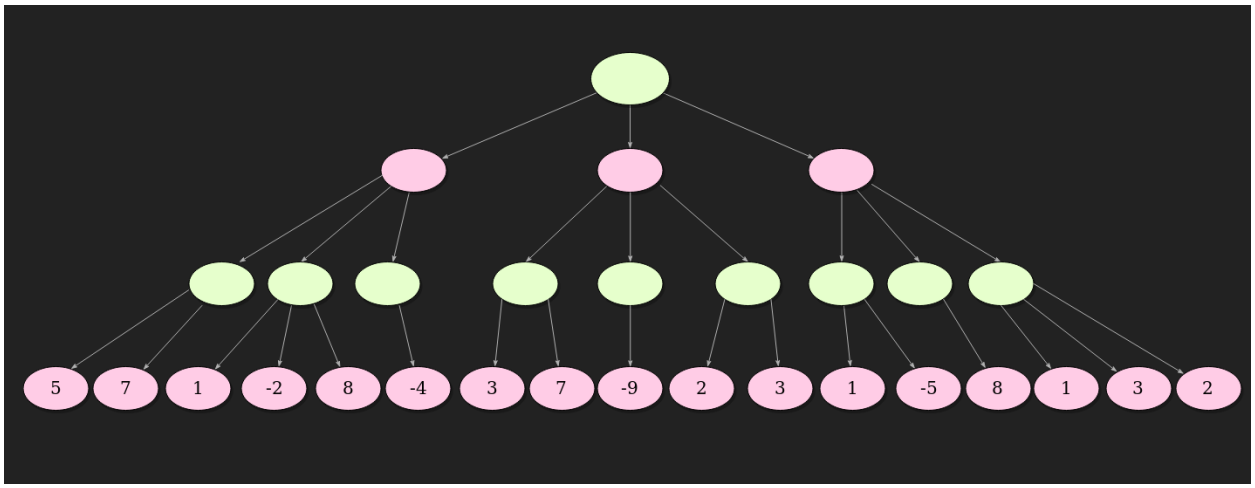

Рисунок 3.1 – Оцінка листків

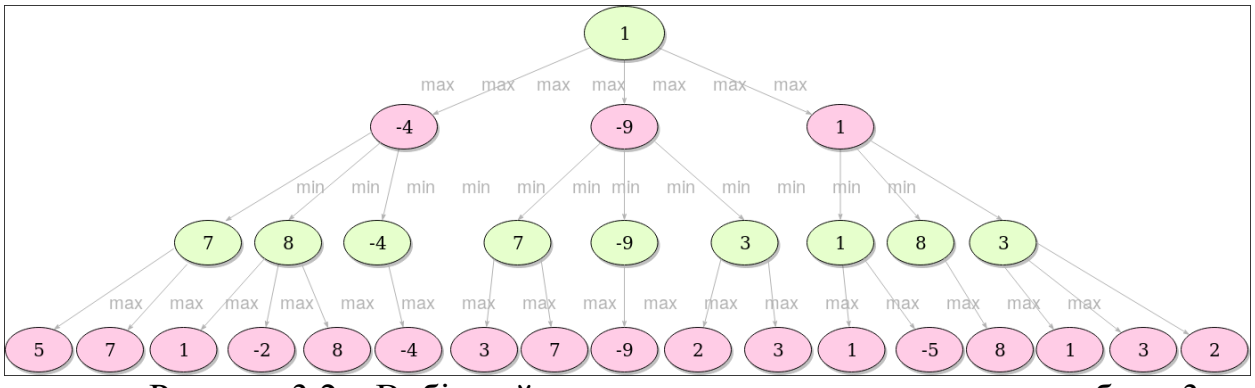

Рисунок 3.2 – Вибір найкращого ходу з використанням глибини 3

Ідея полягає в тому, щоб знайти найкращий можливий хід для даного вузла, глибини та функції оцінки.

У цьому прикладі ми припустили, що зелений гравець шукає позитивні значення, тоді як рожевий гравець шукає негативні. Алгоритм насамперед оцінює лише вузли на заданій глибині, а решта процедури є рекурсивною. Значення інших вузлів є максимальними значеннями їх відповідних дочірніх елементів, якщо це перебіг зеленого гравця, або, аналогічно, мінімальним значенням, якщо це перебіг рожевого гравця. Значення кожному вузлі представляє наступний найкращий хід з урахуванням даної інформації.

Під час пошуку в ігровому дереві ми розглядаємо лише вузли на фіксованій (заданій) глибині, а чи не ті, що були до чи після. Це часто називають ефектом горизонту [30].

3.2 Програмна реалізація штучного інтелекту

Для того щоб була можливість грати проти комп’ютера потрібно розробити штучній інтелект.

Було створено класс «chess_ai» (рис. 3.3) у якому описано логіку штучного інтелекту та «Мінімаксний» алгоритм.

```
class chess_ai:
    '''
    call minimax with alpha beta pruning
    evaluate board
    get the value of each piece
    '''
```

Рисунок 3.3 – Оголошення класу «chess_ai»

Ініціалізована функція «minimax_white» для білих фігур та «minimax_black» для чорних щоб прописати логіку (рис. 3.4, рис. 3.5).

```
def minimax_white(self, game_state, depth, alpha, beta, maximizing_player, player_color):
    csc = game_state.checkmate_stalemate_checker()
    if maximizing_player:
        if csc == 0:
            return 5000000
        elif csc == 1:
            return -5000000
        elif csc == 2:
            return 100
    elif not maximizing_player:
        if csc == 1:
            return 5000000
        elif csc == 0:
            return -5000000
        elif csc == 2:
            return 100

    if depth <= 0 or csc != 3:
        return self.evaluate_board(game_state, Player,PLAYER_1)

    if maximizing_player:
        max_evaluation = -10000000
        all_possible_moves = game_state.get_all_legal_moves("black")
        for move_pair in all_possible_moves:
            game_state.move_piece(move_pair[0], move_pair[1], True)
            evaluation = self.minimax_white(game_state, depth - 1, alpha, beta, False, "white")
            game_state.undo_move()

            if max_evaluation < evaluation:
                max_evaluation = evaluation
                best_possible_move = move_pair
            alpha = max(alpha, evaluation)
            if beta <= alpha:
                break
        if depth == 3:
            return best_possible_move
        else:
            return max_evaluation
    else:
        min_evaluation = 10000000
        all_possible_moves = game_state.get_all_legal_moves("white")
        for move_pair in all_possible_moves:
            game_state.move_piece(move_pair[0], move_pair[1], True)
            evaluation = self.minimax_white(game_state, depth - 1, alpha, beta, True, "black")
            game_state.undo_move()

            if min_evaluation > evaluation:
                min_evaluation = evaluation
                best_possible_move = move_pair
            beta = min(beta, evaluation)
            if beta <= alpha:
                break
        if depth == 3:
            return best_possible_move
        else:
            return min_evaluation
```

Рисунок 3.4 – функція «minimax_white» для білих фігур

```
def minimax_black(self, game_state, depth, alpha, beta, maximizing_player, player_color):
    csc = game_state.checkmate_stalemate_checker()
    if maximizing_player:
        if csc == 1:
            return 5000000
        elif csc == 0:
            return -5000000
        elif csc == 2:
            return 100
    elif not maximizing_player:
        if csc == 0:
            return 5000000
        elif csc == 1:
            return -5000000
        elif csc == 2:
            return 100

    if depth <= 0 or csc != 3:
        return self.evaluate_board(game_state, Player.PLAYER_2)

    if maximizing_player:
        max_evaluation = -10000000
        all_possible_moves = game_state.get_all_legal_moves("white")
        for move_pair in all_possible_moves:
            game_state.move_piece(move_pair[0], move_pair[1], True)
            evaluation = self.minimax_black(game_state, depth - 1, alpha, beta, False, "black")
            game_state.undo_move()

            if max_evaluation < evaluation:
                max_evaluation = evaluation
                best_possible_move = move_pair
            alpha = max(alpha, evaluation)
            if beta <= alpha:
                break
        if depth == 3:
            return best_possible_move
        else:
            return max_evaluation
    else:
        min_evaluation = 10000000
        all_possible_moves = game_state.get_all_legal_moves("black")
        for move_pair in all_possible_moves:
            game_state.move_piece(move_pair[0], move_pair[1], True)
            evaluation = self.minimax_black(game_state, depth - 1, alpha, beta, True, "white")
            game_state.undo_move()

            if min_evaluation > evaluation:
                min_evaluation = evaluation
                best_possible_move = move_pair
            beta = min(beta, evaluation)
            if beta <= alpha:
                break
        if depth == 3:
            return best_possible_move
        else:
            return min_evaluation
```

Рисунок 3.5 – Функція «minimax_black» для чорних фігур

Тепер потрібно прописати логіку ходів фігур для штучного інтелекту та функцію яка буде визначати стан гри(положення фігур), це потрібно щоб штучний інтелект мав можливість робити різні ходи різними фігурами, та міг визначати положення всіх фігур на дошці, для цього написано дві функції «evaluate_board» та «get_piece_value» зображені на рисунку 3.6 та 3.7.

```
def evaluate_board(self, game_state, player):
    evaluation_score = 0
    for row in range(0, 8):
        for col in range(0, 8):
            if game_state.is_valid_piece(row, col):
                evaluated_piece = game_state.get_piece(row, col)
                evaluation_score += self.get_piece_value(evaluated_piece, player)
    return evaluation_score
```

Рисунок 3.6 – функція «evaluate_board»

```
def get_piece_value(self, piece, player):
    if player is Player.PLAYER_1:
        if piece.is_player("black"):
            if piece.get_name() is "k":
                return 1000
            elif piece.get_name() is "q":
                return 100
            elif piece.get_name() is "r":
                return 50
            elif piece.get_name() is "b":
                return 30
            elif piece.get_name() is "n":
                return 30
            elif piece.get_name() is "p":
                return 10
        else:
            if piece.get_name() is "k":
                return -1000
            elif piece.get_name() is "q":
                return -100
            elif piece.get_name() is "r":
                return -50
            elif piece.get_name() is "b":
                return -30
            elif piece.get_name() is "n":
                return -30
            elif piece.get_name() is "p":
                return -10
    else:
        if piece.is_player("white"):
            if piece.get_name() is "k":
                return 1000
            elif piece.get_name() is "q":
                return 100
            elif piece.get_name() is "r":
                return 50
            elif piece.get_name() is "b":
                return 30
            elif piece.get_name() is "n":
                return 30
            elif piece.get_name() is "p":
                return 10
        else:
            if piece.get_name() is "k":
                return -1000
            elif piece.get_name() is "q":
                return -100
            elif piece.get_name() is "r":
                return -50
            elif piece.get_name() is "b":
                return -30
            elif piece.get_name() is "n":
                return -30
            elif piece.get_name() is "p":
                return -10
```

Рисунок 3.7 – функція «get_piece_value»

Був написан клас «chess_ai» з реалізацією алгоритма, навчили комп’ютер робити ходи фігурами, визначати місцеположення фігур тепер можемо грати «Проти комп’ютера» (рис. 3.8, рис. 3.9)

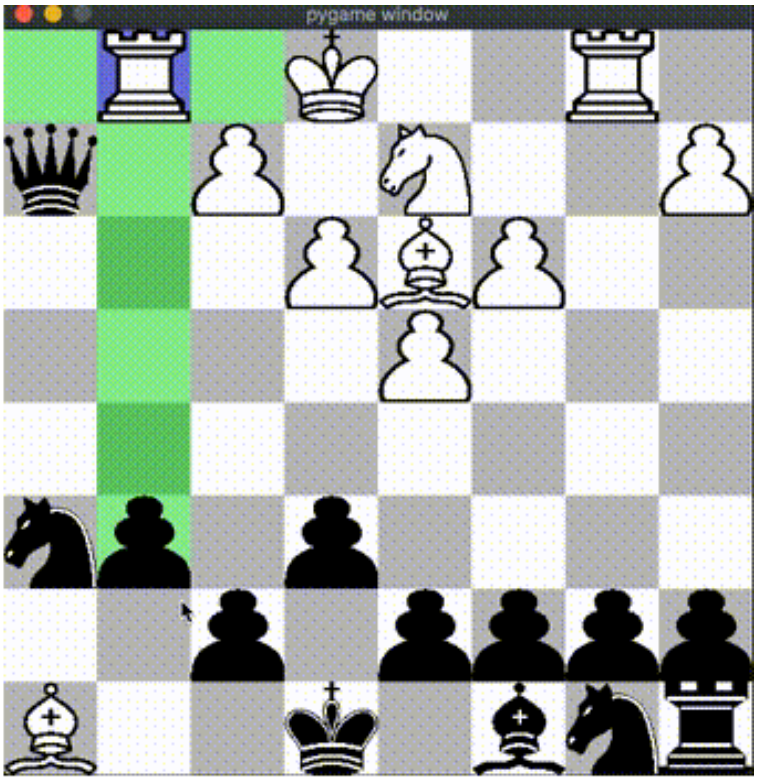

Рисунок 3.8 – Гравець робить хід «Проти комп’ютера»

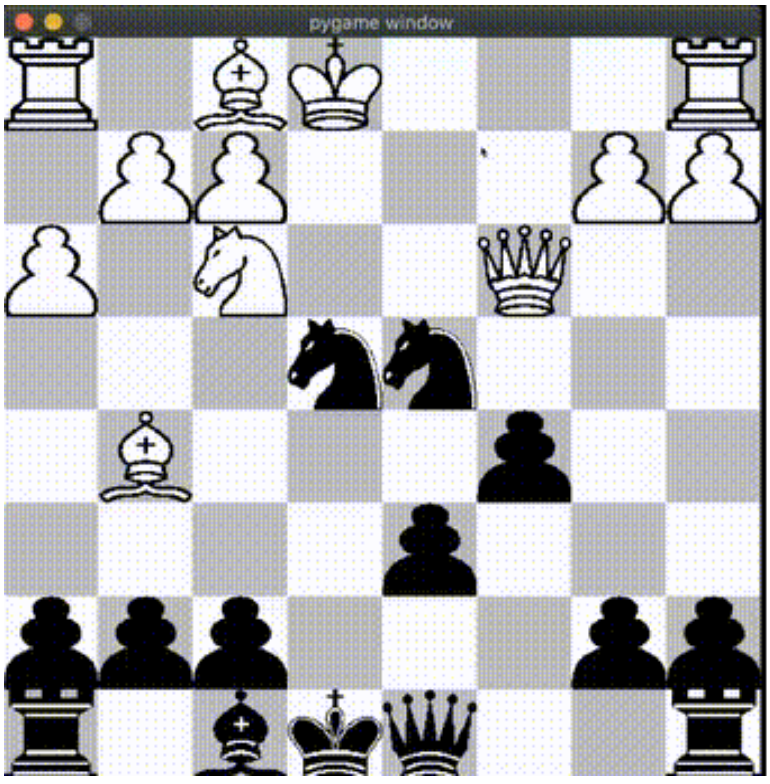

Рисунок 3.9 – Хід гри «Проти комп’ютера»

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблено і реалізовано гру «Шахи» з можливістю грати проти комп’ютера.

Було вивчено такі теоретичні питання: теорію програмування на мові Python, теорію проектування штучного інтелекту, теорію розробки простих ігор з штучним інтелектом.

Ця гра була і буде актуальна завжди.

Можливе покращення гри полягає у додаванні нового функціоналу, починаючи від онлайн режиму закінчуючи таблицею рекордів.

Усі можливі тестування гри були проведені, на даний момент ніяких помилок не виявлено.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. How a chess engine works: from mechanical search to "smart" choices. URL: https://habr.com/ru/company/skillfactory/blog/544040/ (дата звернення 24.04.2022).
2. Alpha-beta clipping algorithm and its examples with images. URL: https://habr.com/ru/company/skillfactory/blog/544040/ (дата звернення 24.04.2022).
3. Chess tactics. URL: https://uk.wikipedia.org/wiki/Тактика_(шахи) (дата звернення 24.04.2022).
4. Chess strategy. URL: https://uk.wikipedia.org/wiki/Стратегія_(шахи) (дата звернення 24.04.2022).
5. Chess debut. URL: https://uk.wikipedia.org/wiki/Шаховий_дебют (дата звернення 24.04.2022).
6. Chess blizzard. URL: https://uk.wikipedia.org/wiki/Мітельшпіль (дата звернення 24.04.2022).
7. Chess endgame. URL: https://uk.wikipedia.org/wiki/Ендшпіль (дата звернення 24.04.2022).
8. Chess zugzwang. URL: https://uk.wikipedia.org/wiki/Цугцванг (дата звернення 24.04.2022).
9. How old is the game of chess. URL: http://thaichess.narod.ru/index/naskolko_stara_shakhmatnaja_igra/0-49 (дата звернення 24.04.2022).
10. History of Chess From Early Stages to Magnus. URL: https://www.chess.com/article/view/history-of-chess (дата звернення 24.04.2022).
11. Chess rules and history. URL: https://web.archive.org/web/20181122172255/http://encyclopaedia.biga.ru/enc/sport/SHAHMATI.html (дата звернення 24.04.2022).

12. Chess lessons. URL: https://web.archive.org/web/20210120030446/https:/edufuture.biz/index.php?title=Шахи_уроки (дата звернення 24.04.2022).
13. A story well told is not necessarily true: a critical assessment of David H. Li's The Genealogy of Chess. URL: http://www.banaschak.net/schach/ligenealogyofchess.htm (дата звернення 24.04.2022).
14. David Vincent Gooper British chess player. URL: https://uk.wikipedia.org/wiki/Девід_Вінсент_Гупер (дата звернення 24.04.2022).
15. Kenneth Wild English chess journalist and historian. URL: https://uk.wikipedia.org/wiki/Кеннет_Вайлд (дата звернення 24.04.2022).
16. Valencia Spain: The Cradle of European Chess. URL: http://history.chess.free.fr/papers/Calvo%201998.pdf (дата звернення 24.04.2022).
17. XiangQi – an alternate to Western Chess. URL: https://en.chessbase.com/post/xiangqi-an-alternate-to-western-che (дата звернення 24.04.2022).
18. Kramnik plays Makruk Thai. URL: http://www.chessvariants.org/oriental.dir/thaikramnik.html (дата звернення 24.04.2022).
19. Chess History and Reminiscences. URL: https://web.archive.org/web/20090924125500/http://www.gutenberg.org/etext/4902 (дата звернення 24.04.2022).
20. London Chess Club. URL: https://www.chessgames.com/perl/chessplayer?pid=80740 (дата звернення 24.04.2022).

21. That Time When People Thought Playing Chess Would Make You Violent. URL: https://www.techdirt.com/2014/06/20/that-time-when-people-thought-playing-chess-would-make-you-violent/ (дата звернення 24.04.2022).
22. Chess rules. URL: https://uk.wikipedia.org/wiki/Правила_шахів (дата звернення 24.04.2022).
23. Python programming language. URL: https://uk.wikipedia.org/wiki/Python (дата звернення 24.04.2022).
24. Pygame set of cross-platform modules. URL: https://uk.wikipedia.org/wiki/Pygame (дата звернення 24.04.2022).
25. PyCharm is an integrated development environment. URL: https://uk.wikipedia.org/wiki/PyCharm (дата звернення 24.04.2022).
26. Adobe Photoshop. URL: https://ru.wikipedia.org/wiki/Adobe_Photoshop (дата звернення 24.04.2022).
27. Artificial intelligence. URL: https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html (дата звернення 24.04.2022).
28. Chessboard history. URL: https://uk.wikipedia.org/wiki/Шахівниця (дата звернення 24.04.2022).
29. Chess pieces history. URL: https://uk.wikipedia.org/wiki/Фігура_(шахи) (дата звернення 24.04.2022).
30. Minimax with Alpha Beta cropping in Python. URL: https://pythobyte.com/minimax-and-alpha-beta-pruning-in-python-fe960495/ (дата звернення 24.04.2022).