МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

**КУРСОВАЯ РАБОТА**

по дисциплине «Алгоритмы и структуры данных»

|  |  |
| --- | --- |
| **Подп.и** |  |
| **Инв** |  |
| **Вза** |  |
| **Подп.и** |  |
| **Инв** |  |

**Компьютерная игра эндшпиль «Король, ладья – Король, 2 коня»**

Руководитель разработки:  
Кандидат технических наук,

Доцент кафедры ИВК  
Шишкин Вадим Викторинович

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

Исполнитель:

студент гр. ИСТбд-23

Аскаров Линар Радикович

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**Содержание**

**Техническое задание……………………………………………………………………..3**

**Пояснительная записка………………………………………………………………..10**

**Руководство программиста……………………………………………………………22**

**Листинг кода…………………………………………………………………………….34**

**Источники, использованные при разработке ……………………………………...53**

**Заключение……………………………………………………………………………...54**

**Приложение 1……………………………………………………………………………55**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

|  |  |
| --- | --- |
| **Подп.и** |  |
| **Инв** |  |
| **Вза** |  |
| **Подп.и** |  |
| **Инв** |  |

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

**Компьютерная игра эндшпиль «Король, ладья – Король, 2 коня»**

Р.02069337. №23/721-Вариант 4

Листов 6

Исполнитель:

студент гр. ИСТбд-23

Аскаров Линар Радикович

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**Введение**

Наименование приложения:

Компьютерная игра эндшпиль «Король, ладья – Король, 2 коня»

Условное обозначение:

"Endgame: King and Rook vs King and Two Knights"

Краткий свод правил игры

Цель игры:

Целью игры является победа одной из сторон путем матования противника. В данной ситуации одна сторона управляет королем и ладьей, а другая — королем и двумя конями.

Правила движения фигур:

* Король: может перемещаться на одну клетку в любом направлении.
* Ладья: может перемещаться на любое количество клеток по горизонтали или вертикали.
* Конь: может перемещаться буквой "Г" — на две клетки в одном направлении и одну клетку в перпендикулярном.

Победа:

* Игрок, управляющий ладьей и королем, выигрывает, если ставит короля противника под шах и мат.
* Игрок, управляющий королем и двумя конями, может избежать мата и стремиться к ничьей.

Ничья:

Игра может закончиться вничью, если одна из сторон не может сделать ход, не попадая под шах.

Общая характеристика функциональных возможностей приложения

Графический интерфейс:

* Интуитивно понятный и удобный интерфейс с шахматной доской и фигурами.
* Возможность выбора цвета фигур для каждого игрока (белые и черные).

Игровой процесс:

* Возможность выбора режима игры: против компьютера или против другого игрока.
* Реализация правил шахмат, включая возможность отмены последнего хода.

Искусственный интеллект:

* Разработка простого алгоритма для управления компьютером, который будет принимать решения на основе базовых стратегий игры.

1. **Основания для разработки**

Учебный план определяет ключевые компетенции и знания, которые должны быть получены студентом в процессе обучения. Он включает в себя изучение основ программирования, проектирования информационных систем, а также принципов разработки программного обеспечения. Эти знания являются основой для реализации проекта, который нацелен на практическое применение теоретических знаний. Распоряжение, изданное факультетом, определяет цели и задачи учебных проектов, а также сроки их выполнения. Оно служит руководством для студентов и преподавателей, указывая на необходимость разработки проектов, которые соответствуют современным требованиям индустрии.

2. **Требования к программе или программному изделию**

2.1. **Функциональное назначение**

Функциональное назначение приложения "Эндшпиль: Король и Ладья против Короля и Двух Коней" заключается в создании интерактивной платформы для обучения и практики в области шахматных эндшпилей, а также в предоставлении пользователям возможности улучшать свои навыки стратегического мышления и принятия решений в условиях ограниченного времени и ресурсов.

2.2. **Требования к функциональным характеристикам**

2.2.1 **Требования к структуре приложения**

• Приложение должно быть организовано в виде модулей, каждый из которых отвечает за определенный функционал (например, модуль игры, модуль интерфейса, модуль ИИ и т.д.).

• Логика игры (правила, алгоритмы) должна быть отделена от графического интерфейса, что упростит поддержку и расширение приложения.

2.2.2 **Требования к составу функций приложения**

Выбор режима игры:

• Игра против компьютера.

• Игра против другого игрока(локально).

Графический интерфейс:

* Интуитивно понятный интерфейс с шахматной доской и элементами управления.

Движение фигур:

* Возможность перемещения фигур по доске с учетом правил шахмат.
* Автоматическая проверка на допустимость хода (например, проверка на шах).

2.2.3 **Требования к организации информационного обеспечения, входных и выходных данных**

Каждая игра должна хранить следующие данные:

* Текущая позиция фигур на доске (например, массив или объект, представляющий каждую клетку и находящиеся на ней фигуры).
* История ходов (массив строк, где каждый элемент представляет ход в формате "e2-e4").

Алгоритм обмена данными:

Инициализация игры:

* При запуске приложения загружается основной интерфейс, инициализируется шахматная доска.

Ввод данных:

* Пользователь нажимает на квадратики, которые обрабатываются приложением (проверка допустимости, выполнение хода и обновление состояния).

Обновление состояния:

* После каждого хода обновляется состояние игры (проверка на шах, мат, ничью) и отображается информация на интерфейсе.

2.3. **Требования к надежности**

• Стабильность: программа должна работать стабильно и не выдавать ошибок при нормальных условиях эксплуатации.

• Отказоустойчивость: программа должна быть способна восстанавливаться после сбоев или ошибок, не теряя данные и сохраняя работоспособность.

• Надежность данных: программа должна обеспечивать целостность и достоверность данных, используемых и обрабатываемых ею.

• Безопасность: программа должна быть защищена от несанкционированного доступа, модификации или уничтожения данных.

2.4. **Требования к информационной и программной совместимости**

Операционная система:

– Windows, macOS, Linux - приложение должно работать на всех этих операционных системах, так как Pygame доступен во всех из них.

Платформа:

– Python 3.x - приложение должно работать на Python версии 3.x.

Инструментальная среда:

– Любая IDE, поддерживающая Python 3.x - например, Visual Studio Code, PyCharm, IDLE, Thonny и т.д.

Библиотеки:

– Pygame - стандартная библиотека для создания компьютерных игр и мультимедиа-приложений в Python.

2.5. **Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

2.6. **Требования к транспортированию и хранению**

2.6.1 **Условия транспортирования**

Требования к условиям транспортирования не предъявляются.

2.6.2 **Условия хранения**

Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы.

2.6.3 **Сроки хранения**

Срок хранения – до окончания срока учебы.

3. **Требования к программной документации**

Определяются заданием на курсовую работу.

4. **Стадии и этапы разработки**

Определяются заданием на курсовую работу.

5. **Порядок контроля и приемки**

Определяются заданием на курсовую работу.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

|  |  |
| --- | --- |
| **Подп.и** |  |
| **Инв** |  |
| **Вза** |  |
| **Подп.и** |  |
| **Инв** |  |

**Компьютерная игра эндшпиль «Король, ладья – Король, 2 коня»**

Р.02069337. №23/721 Вариант 4

Листов 11

Исполнитель:

студент гр. ИСТбд-23

Аскаров Линар Радикович

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**Введение**

В данном приложении реализован шахматный эндшпиль с использованием графической библиотеки Pygame для создания интерактивного интерфейса.

Выбранный подход включает в себя использование алгоритма Минимакс для нахождения оптимальных ходов для компьютера, что позволяет игрокам сразиться как с другими игроками, так и с искусственным интеллектом.

Приложение также включает систему регистрации и аутентификации пользователей, что позволяет сохранять данные о пользователях и их паролях в зашифрованном виде, обеспечивая безопасность.

Краткое описание реализованного приложения:

Приложение предоставляет пользователям возможность играть в шахматы с графическим интерфейсом, где они могут делать ходы, видеть историю ходов и получать информацию о текущем состоянии игры.

Реализованы функции для загрузки изображений шахматных фигур, отображения доски, а также механизмы для управления ходами и проверки условий окончания игры (мат, пат).

Пользователи могут регистрироваться и входить в систему, что позволяет им сохранять свои данные и играть в шахматы в защищенной среде.

**1. Проектная часть**

**1.1. Постановка задачи на разработку приложения**

Задача на разработку приложения определяется заданием на курсовую работу. Приложение должно реализовать игру в шахматный эндшпиль с участием короля, ладьи и двух коней. Игрок играет за белых, а искусственный интеллект — за чёрных. Приложение включает в себя функциональность регистрации и входа пользователей, логику искусственного интеллекта и историю ходов.

**1.2. Математические методы**

Алгоритм Minimax является основным методом, используемым для реализации искусственного интеллекта в шахматах и других играх с нулевой суммой. Он основан на принципе минимизации максимального риска, что позволяет AI принимать оптимальные решения в условиях неопределенности.

Основная идея алгоритма заключается в том, что AI рассматривает все возможные ходы и их последствия, оценивая каждую позицию на доске. Он выбирает ход, который максимизирует его шансы на победу, одновременно минимизируя шансы противника.

1. Алгоритм Minimax:

Алгоритм рекурсивно исследует все возможные ходы, начиная с текущей позиции. Для каждого хода он вызывает себя для оценки последующих позиций, пока не достигнет заданной глубины поиска или конца игры.

На каждом уровне рекурсии происходит оценка позиции с использованием функции оценки, которая учитывает такие факторы, как материальное преимущество, контроль центра, безопасность короля и другие стратегические аспекты.

1. Оценка состояния игры:

Функция оценки используется для количественной оценки позиции на доске, когда достигается максимальная глубина поиска. Она позволяет AI принимать решения, основываясь на текущем состоянии игры, даже если не все возможные ходы были исследованы.

Функция может включать различные параметры, такие как количество фигур, их расположение, активность и другие факторы, влияющие на шансы на победу.

**1.3. Архитектура и алгоритмы**

1.3.1 Архитектура

Приложение состоит из нескольких модулей, каждый из которых отвечает за определённую функциональность:

**Инициализация и загрузка изображений:**

* load\_images(): Загружает изображения шахматных фигур и масштабирует их до нужного размера.

**Управление пользователями:**

* save\_users(users): Сохраняет список пользователей в файл users.txt.
* load\_users(): Загружает список пользователей из файла users.txt.
* encrypt\_password(password): Шифрует пароль.
* decrypt\_password(encrypted\_password): Дешифрует пароль.

**Основной игровой цикл**:

* main(): Основная функция, которая инициализирует Pygame, загружает изображения, создает игровое окно и запускает основной игровой цикл. Внутри цикла обрабатываются события, такие как клики мыши и нажатия клавиш, а также обновляется состояние игры и отрисовывается доска.

**Обработка ходов и анимация:**

* highlightMove(screen, gs, validMoves, sqSelected): Подсвечивает возможные ходы для выбранной фигуры.
* animateMove(move, screen, board, clock): Анимирует перемещение фигуры по доске.

**Отрисовка игрового состояния:**

* drawGameState(screen, gs, validMoves, sqSelected): Отрисовывает текущее состояние игры, включая доску и фигуры.
* drawBoard(screen): Отрисовывает шахматную доску.
* drawPieces(screen, board): Отрисовывает шахматные фигуры на доске.
* drawEndGameText(screen, text): Отображает текст в конце игры (например, "DRAW", "BLACK WIN", "WHITE WIN").
* drawMoveLog(screen, gs, scroll\_offset): Отображает журнал ходов.
* drawScrollBar(screen, scroll\_offset, total\_moves): Отображает полосу прокрутки для журнала ходов.

**Окно входа и регистрации:**

* show\_login\_register\_window(): Создает окно входа и регистрации с использованием Tkinter. Включает в себя две вкладки: "Войти" и "Регистрация". Пользователи могут вводить свои данные и авторизоваться или зарегистрироваться.

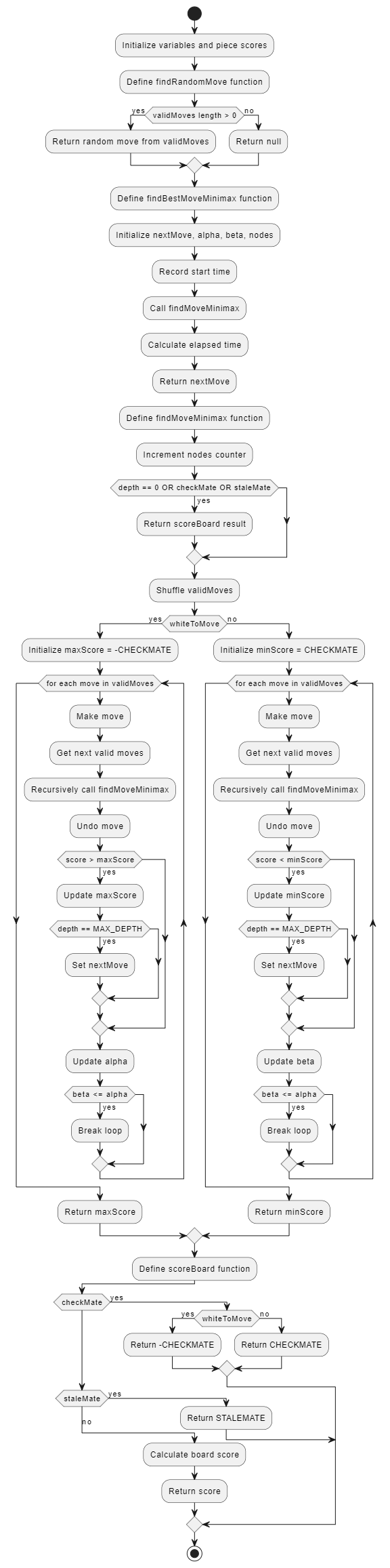
1.3.2 Алгоритм минимакс с альфа-бета отсечениями

Алгоритм минимакс с альфа-бета отсечениями используется для анализа возможных ходов и выбора оптимального хода для ИИ.

Основные шаги алгоритма:

* Увеличить счетчик узлов.
* Если достигнута максимальная глубина или игра завершена (мат или пат), вернуть оценку доски.
* Перемешать список доступных ходов.
* Если ход белых:
* Инициализировать максимальную оценку.
* Для каждого хода в списке доступных ходов:
* Сделать ход.
* Получить список следующих доступных ходов.
* Рекурсивно вызвать findMoveMinimax с уменьшенной глубиной.
* Отменить ход.
* Обновить максимальную оценку и альфа, если найден лучший ход.
* Если альфа больше или равно бета, прервать цикл (альфа-бета отсечение).
* Вернуть максимальную оценку.
* Если ход черных:
* Инициализировать минимальную оценку.
* Для каждого хода в списке доступных ходов:
* Сделать ход.
* Получить список следующих доступных ходов.
* Рекурсивно вызвать findMoveMinimax с уменьшенной глубиной.
* Отменить ход.
* Обновить минимальную оценку и бета, если найден лучший ход.
* Если бета меньше или равно альфа, прервать цикл (альфа-бета отсечение).
* Вернуть минимальную оценку.

Блок-схема алгоритма:



Комментарии:

* Алгоритм Minimax с альфа-бета отсечением позволяет эффективно анализировать большое количество возможных ходов, что особенно важно в шахматах, где количество возможных позиций может быстро возрастать.
* Использование отсечений позволяет исключать менее перспективные ветви дерева решений, что значительно сокращает время, необходимое для принятия решения. Это позволяет AI принимать более обоснованные решения в ограниченные временные рамки, что особенно важно в условиях реального времени.
* Оценка состояния игры основана на разнице в количестве фигур, что является одним из ключевых факторов, влияющих на стратегию игры.
* Алгоритм учитывает материальное преимущество, что позволяет ему принимать решения на основе текущей позиции на доске. Например, если AI имеет больше фигур, он может выбирать более агрессивные стратегии, в то время как в случае недостатка фигур он может сосредоточиться на защите и поиске возможностей для контратаки.
* Такой подход к оценке позволяет AI адаптироваться к различным игровым ситуациям и принимать более стратегически обоснованные решения, что делает игру более интересной и конкурентоспособной для пользователей.

**1.4. Тестирование**

**Mind map**

Для тестирования игры можно привести карту приложения(Приложение 1)

**Чек лист**

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функция | Шаги выполнения | Ожидаемый результат | Статус |
| 1. Регистрация и авторизация пользователей (ChessMain.py) |  |  |  |
| Регистрация нового пользователя | Логин и пароль состоят только из пробелов | Пользователь не регистрируется, выводится ошибка | Пройдено/Не пройдено |
| Вход в систему | Вводится правильный пароль | Вход выполняется, выводится доска с фигурами | Пройдено/Не пройдено |
| Шифрование паролей | Сохранить пароль в открытом виде | Пароль должен быть зашифрован | Пройдено/Не пройдено |
| 3. Логика игры (ChessMain.py) |  |  |  |
| Инициализация игры | Запустить новую игру | Начальная доска инициализирована без ошибок | Пройдено/Не пройдено |
| Обработка ходов игрока | Король ходит через битое поле | Ход не выполняется, т.к игрок видит куда может ходить та или иная фигура | Пройдено/Не пройдено |
| Состояние игры (шах, мат, пат) | Установить позицию мата | Игра завершается с сообщением “White Win» или “Black Win” | Пройдено/Не пройдено |
| 4. Искусственный интеллект (ChessEngine.py) |  |  |  |
| Алгоритм минимакс | ИИ анализирует ход | ИИ находит лучший ход | Пройдено/Не пройдено |
| Альфа-бета отсечение | ИИ рассматривает очень много вариантов | С помощью отсечения сокращается количество исследуемых ходов | Пройдено/Не пройдено |
| Оценка состояния игры | Оценить позицию с разным количеством фигур | Оценка учитывает разницу в фигуры | Пройдено/Не пройдено |

1.4.1 Описание отчета о тестировании

В ходе разработки приложения "Эндшпиль: Король и Ладья против Короля и Двух Коней" был проведен комплексный процесс тестирования, направленный на выявление и устранение ошибок, которые могли бы повлиять на функциональность и стабильность программы.

* + 1. Цель тестирования

Цель тестирования — проверить корректность работы основных функций приложения, включая:

* Регистрацию и вход пользователей.
* Логика игры и проверка правильности алгоритма.

1.4.3 Методика тестирования

Методика тестирования включает:

* Ручное тестирование интерфейса.
* Автоматизированное тестирование логики игры.
* Метод грубой силы

1.4.4 Проведенные тесты

1. Регистрация и авторизация (ChessMain.py)

Цель: Проверить корректность работы регистрации и входа в систему.

Ожидаемый результат:

* При регистрации с логином и паролем, которые целиком состоят из пробелов, выводится ошибка.
* При входе с неправильным паролем вход не выполняется, выводится ошибка.
* Пароль должен быть зашифрован перед сохранением в текстовый файл.

2. Логика игры (ChessMain.py)

Цель: Проверить корректность работы логики игры.

Ожидаемый результат:

* При запуске новой игры начальная доска должна быть корректно инициализирована.
* Фигуры должны ходить только по правилам игры.
* При установке позиции мата или пата игра должна завершаться с сообщением “Мат” или “Пат”.

3. Искусственный интеллект (ChessEngine.py)

Цель: Проверить корректность работы алгоритма ИИ.

Ожидаемый результат:

* ИИ должен выбирать налучший ход, чтобы обыгрывать человека.
* Альфа-бета отсечение должно сокращать количество перебираемых ходов.
* Оценка состояния игры должна учитывать разницу в количестве фигур.

4. Основной интерфейс (ChessMain.py)

Цель: Проверить корректность работы графического интерфейса.

Ожидаемый результат:

* Программа должна запускаться корректно.
* При переключении любого режима игры, она должна работать правильно.

1.4.5 Выводы

Тестирование подтвердило, что приложение функционирует корректно и стабильно. Все основные функции, такие как возможность делать ходы, взаимодействие с AI, а также регистрация и аутентификация пользователей, были проверены и работают без сбоев.

Рекомендации по улучшению включают:

* Улучшить алгоритм искусственного интеллекта, добавив более сложные стратегии и тактики.
* Внедрить дополнительные режимы игры, такие как турниры, обучение или возможность играть с разными уровнями сложности AI.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

**РУКОВОДСТВО ПРОГРАММИСТА**

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

|  |  |
| --- | --- |
| **Подп.и** |  |
| **Инв** |  |
| **Вза** |  |
| **Подп.и** |  |
| **Инв** |  |

**Компьютерная игра эндшпиль «Король, ладья – Король, 2 коня»**

Р.02069337. №23/721-Вариант 4

Листов 11

Исполнитель:

студент гр. ИСТбд-23

Аскаров Линар Радикович

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение "Эндшпиль: Король и Ладья против Короля и Двух Коней" предназначено для реализации шахматного эндшпиля с использованием фигур "король", "конь" и "ладья". Основные функции приложения включают:

* Игра против ИИ: Пользователь может играть против искусственного интеллекта, который использует алгоритм минимакс с альфа-бета отсечениями для выбора оптимальных ходов.
* Регистрация и вход пользователей: Приложение поддерживает регистрацию новых пользователей и вход существующих.
* Оценка состояния игры: Приложение использует простую функцию оценки состояния игры на основе разницы в количестве фигур.
* Графический интерфейс: Приложение предоставляет графический интерфейс для удобства взаимодействия с пользователем.
  1. **Условия, необходимые для использования приложения**

Для использования приложения необходимы следующие условия:

1. Операционная система: Windows.
2. Интерпретатор Python: Версия 3.7 или выше.
3. Библиотеки Python:

* Pygame (для графического интерфейса).
* Tkinter (для графического интерфейса).
* Cryptography (для шифрования паролей).

1. Требования к оборудованию:

* Процессор: x86 или x64.
* ОЗУ: 4 ГБ или больше.
* Графика: Поддержка OpenGL

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество строк кода: примерно 1100 строк.

Количество структур данных: 2 (класс GameState и класс Move).

Количество алгоритмов: 2 (алгоритм минимакс с альфа-бета отсечениями и алгоритм оценки состояния игры).

Используемые библиотеки:

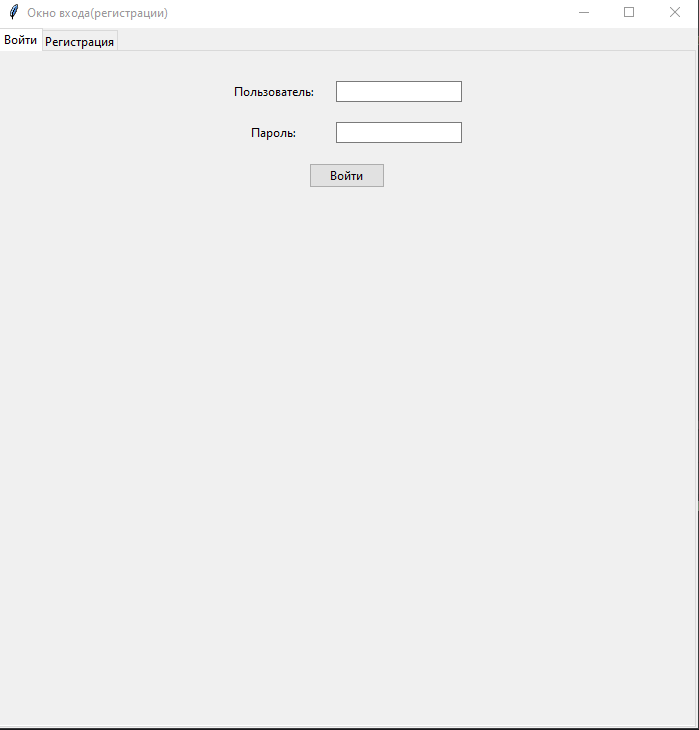
* Pygame: для создания графического интерфейса.
* Cryptography: для зашифрования паролей.
* Tkinter: для создания графического интерфейса.

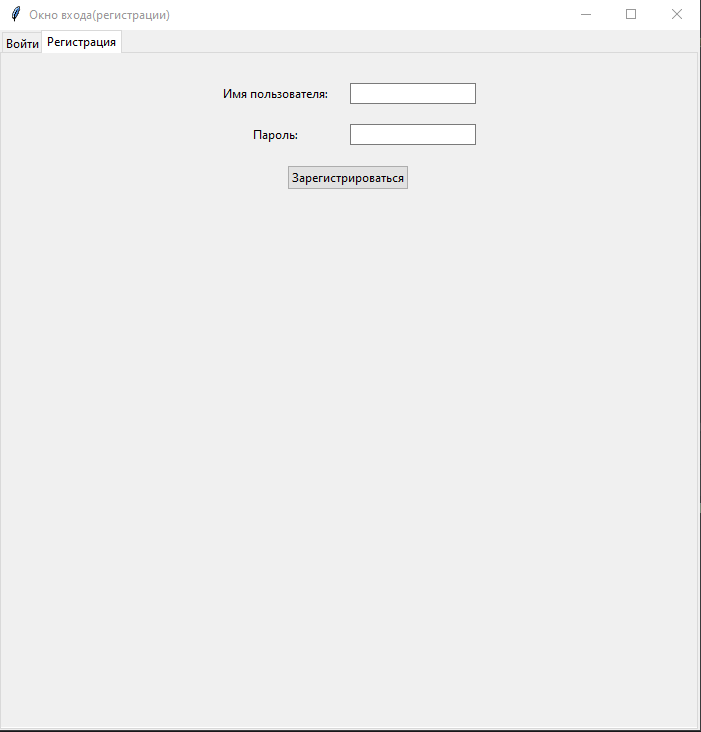
Работа приложения:

* Приложение запускается с помощью файла ChessMain.py.
* Пользователь должен зарегистрироваться или войти в систему.
* После входа пользователь начинает новую игру.
* Во время игры пользователь взаимодействует с графическим интерфейсом, выбирая фигуры и делая ходы.
* ИИ анализируя ходы с помощью алгоритма минимакс, обрабатывает и делает свои ходы.

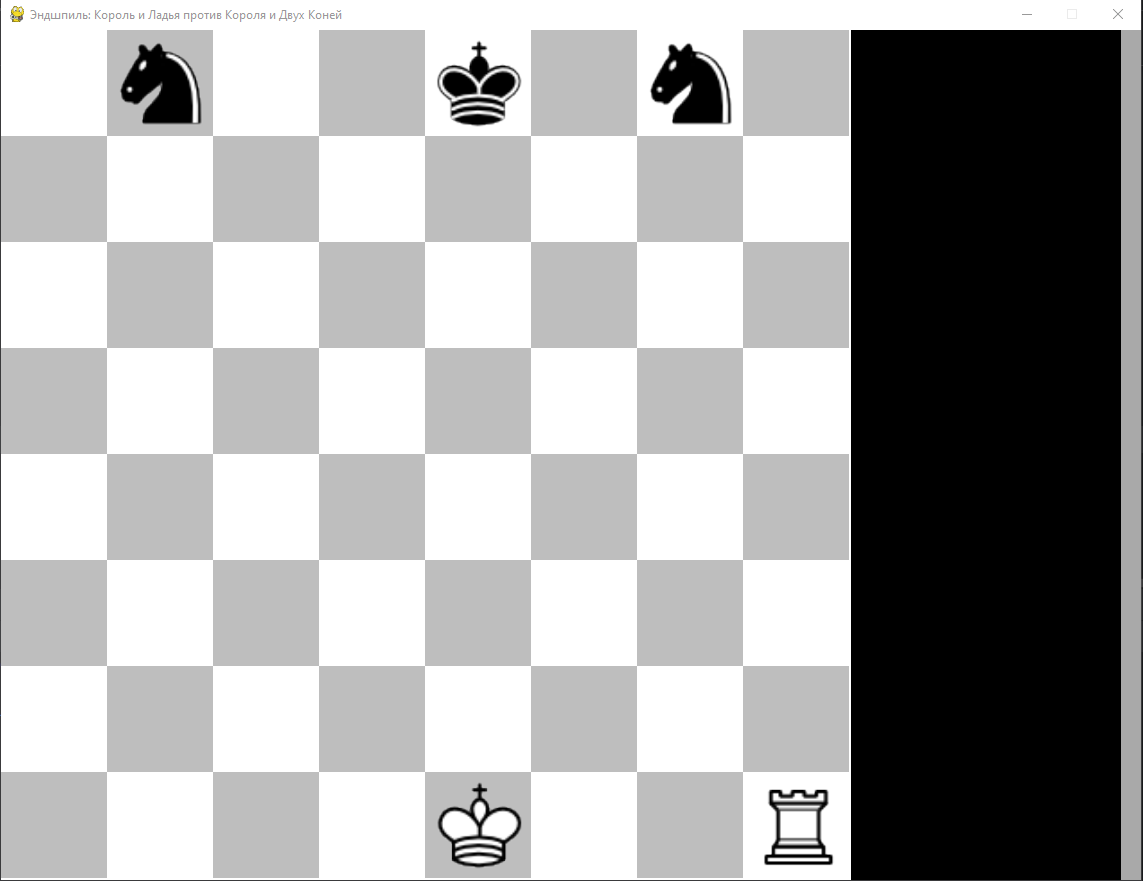
Внешний вид приложения:

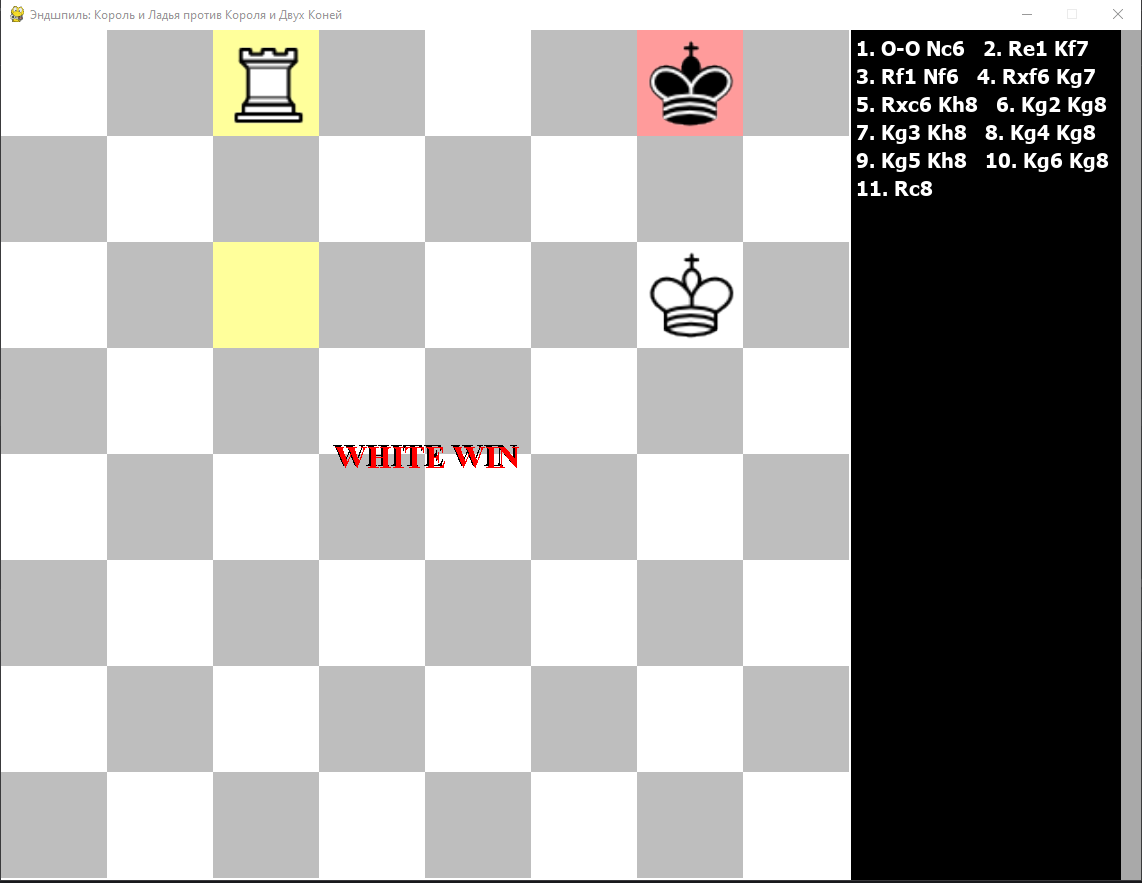
1. Главный экран: Экран входа / регистрации.





1. Главное окно игры: Графическое представление доски с фигурами





Средства контроля корректности ввода/вывода:

* При входе в систему проверяется корректность введенного имени пользователя(логин) и пароля. Если данные не совпадают с сохраненными в базе данных, пользователю выводится сообщение об ошибке.
* Проверка, что ход не выходит за пределы шахматной доски.
* Проверка, что ход не ставит короля под шах. Если ход приводит к шаху, он считается недопустимым, и игроку запрещено ходить на это поле.

**2.2 Особенности реализации приложения**

Структуры данных:

1. Метод main:

* Включает в себя: состояние игры, доску, фигуры, историю ходов

1. Класс GameState:

* Содержит в себе доску в виде массива, логику игры, проверки на шах, мат или пат

1. Users.txt:

* Представляет собой текстовый файл, который хранит информацию о пользователе (имя, пароль (в зашифрованном виде)).

**3. Обращение к программе**

**1. Методы и алгоритмы модуля ChessMain.py**

1.1 Массивы и списки

Доска (board):

Тип: Двумерный массив (список списков).

Описание: Доска представлена в виде двумерного массива размером 8x8, где каждая ячейка содержит строку, представляющую фигуру (например, "wK" для белого короля или "--" для пустой клетки).

Альтернативы: Можно было бы использовать одномерный массив с вычислением индексов, но это усложнило бы логику работы с доской. Двумерный массив обеспечивает более интуитивное представление и упрощает доступ к элементам.

История ходов (moveLog):

Тип: Список.

Описание: Список, хранящий все сделанные ходы в игре. Каждый элемент списка представляет собой объект класса Move, который содержит информацию о начале и конце хода, а также о захваченной фигуре.

Альтернативы: Можно было бы использовать очередь, но список позволяет легко добавлять и удалять элементы, а также удобно итерироваться по всем ходам.

1.2 Словари

Пользователи (users):

Тип: Словарь.

Описание: Словарь, где ключами являются имена пользователей, а значениями — зашифрованные пароли. Это позволяет быстро проверять существование пользователя и получать его пароль.

Альтернативы: Можно было бы использовать список кортежей, но это усложнило бы поиск и проверку уникальности пользователей. Словарь обеспечивает O(1) сложность для операций поиска и вставки.

1.3 **Прямоугольники** (Rect)

Параметры интерфейса:

Тип: Объекты Rect из библиотеки Pygame.

Описание: Используются для определения размеров и положения различных элементов интерфейса, таких как игровая доска, кнопки и полосы прокрутки.

Альтернативы: Можно было бы использовать обычные кортежи для хранения координат и размеров, но использование объектов Rect упрощает работу с графикой и предоставляет встроенные методы для проверки коллизий и взаимодействия.

1.4 **Классы**

Класс Move:

Тип: Пользовательский класс.

Описание: Класс, представляющий ход в игре. Содержит информацию о начальной и конечной позициях, перемещенной фигуре и захваченной фигуре.

Альтернативы: Можно было бы использовать простые структуры данных, такие как кортежи, но использование класса позволяет инкапсулировать логику и методы, связанные с ходом, что делает код более организованным и читаемым.

Заимствование из общедоступных источников

В процессе разработки были использованы некоторые общедоступные решения и алгоритмы, такие как:

Алгоритм Minimax с альфа-бета отсечением:

Источник: Общепринятые алгоритмы для игр с нулевой суммой, описанные в литературе по искусственному интеллекту и шахматам.

Описание: Алгоритм был адаптирован для использования в приложении, что позволило создать эффективный искусственный интеллект для игры в шахматы.

Шифрование паролей с использованием библиотеки cryptography:

Источник: Документация библиотеки cryptography (https://cryptography.io/en/latest/).

Описание: Использование алгоритма шифрования Fernet для безопасного хранения паролей пользователей. Это решение было выбрано для обеспечения безопасности данных и защиты от несанкционированного доступа.

**2. Методы и алгоритмы модуля ChessEngine.py**

2.1 Список для хранения истории ходов

self.moveLog = []

Обоснование выбора:

* Список moveLog используется для хранения всех сделанных ходов, что позволяет реализовать функциональность отмены ходов (undo).
* Списки в Python обеспечивают быструю вставку и удаление элементов, что делает их подходящими для этой задачи.

2.2 Кортежи для представления координат

self.whiteKingLocate = (7, 4)

self.blackKingLocate = (0, 4)

Обоснование выбора:

* Кортежи используются для хранения координат королей, так как они неизменяемы и обеспечивают безопасность данных.
* Кортежи позволяют легко передавать координаты в функции и методы, не беспокоясь о случайном изменении их значений.

2.3 Классы для представления состояния игры и ходов

class GameState():

...

class Move():

...

Обоснование выбора:

* Использование классов позволяет организовать код и разделить логику игры и представление ходов.
* Класс GameState управляет состоянием игры, включая доску, историю ходов и правила, в то время как класс Move отвечает за детали конкретного хода.
* Это улучшает модульность и позволяет легче тестировать и изменять отдельные части кода.

**6. Описания библиотек**

6.1 pygame

* Описание: Библиотека для создания игр и мультимедийных приложений.
* Функции:
* pygame.init (): Инициализация библиотеки.
* pygame.display.set\_mode (): Создание окна.
* pygame.image.load (): Загрузка изображений.
* pygame.transform.scale (): Масштабирование изображений.
* pygame.draw.rect (): Отрисовка прямоугольников.
* pygame.font.SysFont (): Создание шрифта.
* pygame.event.get (): Получение событий.
* pygame.mouse.get\_pos (): Получение позиции мыши.
* Применение: Используется для создания графического интерфейса игры.

6.2 cryptography

* Описание: Библиотека для шифрование паролей.
* Функции:
* cipher\_suite.encrypt(password.encode()).decode():   
  Шифрование пароля.
* cipher\_suite.decrypt(encrypted\_password.encode()).decode(): Расшифрование пароля.
* Применение: Используется для безопасного хранения паролей пользователей.

6.3 math

* Описание: Стандартная библиотека Python для математических операций.
* Функции:
* math.sqrt: Квадратный корень.
* Применение: Используется для анимации фигур.

**4. Сообщения**

1. Сообщение об ошибке входа:

* Текст: “Неверное имя пользователя или пароль”
* Условие: Неправильный логин или пароль при входе.

2. Сообщение об ошибке регистрации:

* Текст: "Имя пользователя уже занято."
* Условие: Попытка регистрации с уже существующим именем пользователя.

3. Сообщение об успешной регистрации:

* Текст: "Вы успешно зарегистрировались!"
* Условие: Успешная регистрация нового пользователя.

4. Сообщение о пате:

* Текст: "StaleMate!"
* Условие: Достигнута Ничья.

5. Сообщение о мате:

* Текст: "White Win" или "Black Win"
* Условие: Король игрока под матом.

**Листинг программы**

# ChessMain.py

import sys  
import pygame as p  
import ChessEngine  
import Minimax  
import math  
import tkinter.messagebox  
from cryptography.fernet import Fernet  
import tkinter as tk  
from tkinter import ttk  
  
WIDTH = 850  
HEIGHT = 850  
MOVE\_LOG\_PANEL\_WIDTH = 290  
MOVE\_LOG\_PANEL\_HEIGHT = HEIGHT  
DIMENSION = 8  
SQ\_SIZE = HEIGHT // DIMENSION  
MAX\_FPS = 240  
IMAGES = {}  
  
*# Генерация ключа для шифрования*key = Fernet.generate\_key()  
cipher\_suite = Fernet(key)  
  
def load\_images():  
 pieces = ["wp", "wR", "wN", "wB", "wQ", "wK", "bp", "bR", "bN", "bB", "bQ", "bK"]  
 for piece in pieces:  
 IMAGES[piece] = p.transform.smoothscale(p.image.load("images/" + piece + ".png").convert\_alpha(), (SQ\_SIZE, SQ\_SIZE))  
  
def save\_users(users):  
 with open('users.txt', 'w', encoding='utf-8') as f:  
 for username, encrypted\_password in users.items():  
 f.write(f"{username},{encrypted\_password}\n")  
  
def load\_users():  
 users = {}  
 try:  
 with open('users.txt', 'r', encoding='utf-8') as f:  
 for line in f:  
 username, encrypted\_password = line.strip().split(',')  
 users[username] = encrypted\_password  
 except FileNotFoundError:  
 pass  
 return users  
  
def encrypt\_password(password):  
 return cipher\_suite.encrypt(password.encode()).decode()  
  
def decrypt\_password(encrypted\_password):  
 return cipher\_suite.decrypt(encrypted\_password.encode()).decode()  
  
def main():  
 p.init()  
 screen = p.display.set\_mode((WIDTH + MOVE\_LOG\_PANEL\_WIDTH, HEIGHT))  
 clock = p.time.Clock()  
 screen.fill(p.Color("white"))  
 gs = ChessEngine.GameState()  
 validMoves = gs.getValidMoves()  
 moveMade = False  
 animate = False  
 gameOver = False  
 playerOne = True  
 playerTwo = False  
 load\_images()  
 sqSelected = ()  
 playerClicks = []  
 scroll\_offset = 0  
 scroll\_bar\_dragging = False  
 scroll\_bar\_rect = p.Rect(WIDTH + MOVE\_LOG\_PANEL\_WIDTH - 20, 0, 20, MOVE\_LOG\_PANEL\_HEIGHT)  
 scroll\_bar\_handle\_rect = p.Rect(WIDTH + MOVE\_LOG\_PANEL\_WIDTH - 20, 0, 20, 50)  
  
 running = True  
 while running:  
 humanTurn = (gs.whiteToMove and playerOne) or (not gs.whiteToMove and playerTwo)  
 for e in p.event.get():  
 if e.type == p.QUIT:  
 running = False  
 elif e.type == p.MOUSEBUTTONDOWN:  
 if not gameOver and humanTurn:  
 location = p.mouse.get\_pos()  
 col = location[0] // SQ\_SIZE  
 row = location[1] // SQ\_SIZE  
 if sqSelected == (row, col) or col >= 8:  
 sqSelected = ()  
 playerClicks = []  
 else:  
 sqSelected = (row, col)  
 playerClicks.append(sqSelected)  
 if len(playerClicks) == 2:  
 move = ChessEngine.Move(playerClicks[0], playerClicks[1], gs.board)  
 if move in validMoves:  
 move = validMoves[validMoves.index(move)]  
 gs.makeMove(move)  
 moveMade = True  
 animate = True  
 sqSelected = ()  
 playerClicks = []  
 print(move.getChessNotation())  
 else:  
 playerClicks = [sqSelected]  
 if scroll\_bar\_rect.collidepoint(e.pos):  
 scroll\_bar\_dragging = True  
 mouse\_x, mouse\_y = e.pos  
 scroll\_offset = (mouse\_y - scroll\_bar\_handle\_rect.y) / (MOVE\_LOG\_PANEL\_HEIGHT - scroll\_bar\_handle\_rect.height) \* len(gs.moveLog)  
 elif e.type == p.MOUSEBUTTONUP:  
 scroll\_bar\_dragging = False  
 elif e.type == p.MOUSEMOTION:  
 if scroll\_bar\_dragging:  
 mouse\_x, mouse\_y = e.pos  
 scroll\_offset = (mouse\_y - scroll\_bar\_handle\_rect.y) / (MOVE\_LOG\_PANEL\_HEIGHT - scroll\_bar\_handle\_rect.height) \* len(gs.moveLog)  
 scroll\_offset = max(0, min(scroll\_offset, len(gs.moveLog) - 1))  
 elif e.type == p.KEYDOWN:  
 if e.key == p.K\_z:  
 gs.undoMove()  
 moveMade = True  
 animate = False  
 gameOver = False  
 playerOne = True  
 playerTwo = True  
 elif e.key == p.K\_r:  
 gs = ChessEngine.GameState()  
 validMoves = gs.getValidMoves()  
 moveMade = False  
 animate = False  
 gameOver = False  
 playerOne = True  
 playerTwo = True  
 sqSelected = ()  
 playerClicks = []  
 elif e.key == p.K\_q:  
 playerOne = False  
 playerTwo = True  
 elif e.key == p.K\_e:  
 playerOne = True  
 playerTwo = False  
 elif e.key == p.K\_UP:  
 scroll\_offset = max(0, scroll\_offset - 1)  
 elif e.key == p.K\_DOWN:  
 scroll\_offset = min(len(gs.moveLog) - 1, scroll\_offset + 1)  
  
 if moveMade:  
 if animate:  
 animateMove(gs.moveLog[-1], screen, gs.board, clock)  
 validMoves = gs.getValidMoves()  
 moveMade = False  
 animate = False  
  
 ''' AI move finder '''  
 if not gameOver and not humanTurn:  
 AIMove = Minimax.findBestMoveMinimax(gs, validMoves)  
 if AIMove is None: *#when begin the game* AIMove = Minimax.findRandomMove(validMoves)  
 gs.makeMove(AIMove)  
 moveMade = True  
 animate = True  
 print(AIMove.getChessNotation())  
  
 drawGameState(screen, gs, validMoves, sqSelected)  
  
 if gs.checkMate or gs.staleMate:  
 gameOver = True  
 if gs.staleMate:  
 gameOver = True  
 drawEndGameText(screen, "DRAW")  
 else:  
 if gs.whiteToMove:  
 drawEndGameText(screen, "BLACK WIN")  
 else:  
 drawEndGameText(screen, "WHITE WIN")  
  
 drawMoveLog(screen, gs, scroll\_offset)  
 drawScrollBar(screen, scroll\_offset, len(gs.moveLog))  
  
 clock.tick(MAX\_FPS)  
 p.display.flip()  
  
def highlightMove(screen, gs, validMoves, sqSelected):  
 sq = p.Surface((SQ\_SIZE, SQ\_SIZE))  
 sq.set\_alpha(100)  
 if sqSelected != ():  
 r, c = sqSelected  
 if gs.board[r][c][0] == ('w' if gs.whiteToMove else 'b'): *#sqSelected is a piece that can be moved  
 #highlight selected square* sq.fill(p.Color("blue"))  
 screen.blit(sq, (c \* SQ\_SIZE, r \* SQ\_SIZE))  
 *#highlight validmoves* sq.fill(p.Color("cyan"))  
 for move in validMoves:  
 if move.startRow == r and move.startCol == c:  
 screen.blit(sq, (move.endCol \* SQ\_SIZE, move.endRow \* SQ\_SIZE))  
  
 if gs.inCheck:  
 if gs.whiteToMove:  
 sq.fill(p.Color("red"))  
 screen.blit(sq, (gs.whiteKingLocate[1] \* SQ\_SIZE, gs.whiteKingLocate[0] \* SQ\_SIZE))  
 else:  
 sq.fill(p.Color("red"))  
 screen.blit(sq, (gs.blackKingLocate[1] \* SQ\_SIZE, gs.blackKingLocate[0] \* SQ\_SIZE))  
  
 if len(gs.moveLog) != 0:  
 sq.fill(p.Color("yellow"))  
 screen.blit(sq, (gs.moveLog[-1].startCol \* SQ\_SIZE, gs.moveLog[-1].startRow \* SQ\_SIZE))  
 screen.blit(sq, (gs.moveLog[-1].endCol \* SQ\_SIZE, gs.moveLog[-1].endRow \* SQ\_SIZE))  
  
def animateMove(move, screen, board, clock):  
 colors = [p.Color("white"), p.Color("grey")]  
 dR = move.endRow - move.startRow  
 dC = move.endCol - move.startCol  
 sqDistance = math.sqrt(abs(move.endRow - move.startRow)\*abs(move.endRow - move.startRow) +  
 abs(move.endCol - move.startCol)\*abs(move.endCol - move.startCol))  
 sqDistance = int(sqDistance)  
 framesPerSquare = 12 // sqDistance  
 frameCount = (abs(dR) + abs(dC)) \* framesPerSquare  
 for frame in range(frameCount + 1):  
 r, c = (move.startRow + dR\*frame/frameCount, move.startCol + dC\*frame/frameCount)  
 drawBoard(screen)  
 drawPieces(screen, board)  
 color = colors[(move.endRow + move.endCol) % 2]  
 endSquare = p.Rect(move.endCol\*SQ\_SIZE, move.endRow\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE)  
 p.draw.rect(screen, color, endSquare)  
 if move.pieceCaptured != "--":  
 if move.isEnpassantMove:  
 enPassantRow = (move.endRow + 1) if move.pieceCaptured[0] == 'b' else (move.endRow - 1)  
 endSquare = p.Rect(move.endCol\*SQ\_SIZE, enPassantRow\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE)  
 screen.blit(IMAGES[move.pieceCaptured], endSquare)  
 if move.pieceMoved != "--":  
 screen.blit(IMAGES[move.pieceMoved], p.Rect(c\*SQ\_SIZE, r\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE))  
 p.display.flip()  
 clock.tick(144)  
  
def drawGameState(screen, gs, validMoves, sqSelected):  
 drawBoard(screen)  
 highlightMove(screen, gs, validMoves, sqSelected)  
 drawPieces(screen, gs.board)  
  
def drawBoard(screen):  
 colors = [p.Color("white"), p.Color("grey")]  
 for r in range(DIMENSION):  
 for c in range(DIMENSION):  
 color = colors[((r + c) % 2)]  
 p.draw.rect(screen, color, p.Rect(c\*SQ\_SIZE, r\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE))  
  
def drawPieces(screen, board):  
 for row in range(DIMENSION):  
 for col in range(DIMENSION):  
 piece = board[row][col]  
 if piece != "--":  
 screen.blit(IMAGES[piece], p.Rect(col\*SQ\_SIZE, row\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE))  
  
def drawEndGameText(screen, text):  
 font = p.font.SysFont("Times New Roman", 32, True, False)  
 textObject = font.render(text, False, p.Color("black"))  
 textLocation = p.Rect(0, 0, WIDTH, HEIGHT).move(WIDTH/2 - textObject.get\_width()/2, HEIGHT/2 - textObject.get\_height()/2)  
 screen.blit(textObject, textLocation)  
 textObject = font.render(text, False, p.Color("red"))  
 screen.blit(textObject, textLocation.move(2, 2))  
  
def drawMoveLog(screen, gs, scroll\_offset):  
 moveLogRect = p.Rect(WIDTH, 0, MOVE\_LOG\_PANEL\_WIDTH, MOVE\_LOG\_PANEL\_HEIGHT)  
 p.draw.rect(screen, p.Color("black"), moveLogRect)  
 moveLog = gs.moveLog  
 moveTexts = []  
 for i in range(0, len(moveLog), 2):  
 moveString = str(i//2 + 1) + ". " + str(moveLog[i]) + " "  
 if i+1 < len(moveLog):  
 moveString += str(moveLog[i+1]) + " "  
 moveTexts.append(moveString)  
  
 padding = 5  
 movesPerRow = 2  
 lineSpacing = 3  
 textY = padding  
 for i in range(0, len(moveTexts), movesPerRow):  
 text = ""  
 font = p.font.SysFont("Tahoma", 20, True, False)  
 for j in range(movesPerRow):  
 if i+j < len(moveTexts):  
 text += moveTexts[i+j]  
 textObject = font.render(text, True, p.Color("white"))  
 textLocation = moveLogRect.move(padding, textY - scroll\_offset \* (textObject.get\_height() + lineSpacing))  
 screen.blit(textObject, textLocation)  
 textY += textObject.get\_height() + lineSpacing  
  
def drawScrollBar(screen, scroll\_offset, total\_moves):  
 scroll\_bar\_rect = p.Rect(WIDTH + MOVE\_LOG\_PANEL\_WIDTH - 20, 0, 20, MOVE\_LOG\_PANEL\_HEIGHT)  
 p.draw.rect(screen, p.Color("grey"), scroll\_bar\_rect)  
 handle\_height = max(20, MOVE\_LOG\_PANEL\_HEIGHT / (total\_moves + 1) \* MOVE\_LOG\_PANEL\_HEIGHT)  
 handle\_y = scroll\_offset / (total\_moves + 1) \* (MOVE\_LOG\_PANEL\_HEIGHT - handle\_height)  
 scroll\_bar\_handle\_rect = p.Rect(WIDTH + MOVE\_LOG\_PANEL\_WIDTH - 20, handle\_y, 20, handle\_height)  
 p.draw.rect(screen, p.Color("darkgrey"), scroll\_bar\_handle\_rect)  
  
def show\_login\_register\_window():  
 root = tk.Tk()  
 root.title("Окно входа(регистрации)")  
 root.geometry('%dx%d+%d+%d' % (700, 700, 570, 240))  
  
 tab\_control = ttk.Notebook(root)  
 tab\_login = ttk.Frame(tab\_control)  
 tab\_register = ttk.Frame(tab\_control)  
  
 tab\_control.add(tab\_login, text='Войти')  
 tab\_control.add(tab\_register, text='Регистрация')  
 tab\_control.pack(expand=1, fill="both")  
  
 *# Login Tab* login\_frame = ttk.Frame(tab\_login)  
 login\_frame.pack(pady=20)  
  
 ttk.Label(login\_frame, text="Пользователь:").grid(row=0, column=0, padx=10, pady=10)  
 ttk.Label(login\_frame, text="Пароль:").grid(row=1, column=0, padx=10, pady=10)  
  
 username\_entry = ttk.Entry(login\_frame)  
 password\_entry = ttk.Entry(login\_frame, show="\*")  
  
 username\_entry.grid(row=0, column=1, padx=10, pady=10)  
 password\_entry.grid(row=1, column=1, padx=10, pady=10)  
  
 def login():  
 username = username\_entry.get()  
 password = password\_entry.get()  
 users = load\_users()  
 if username in users and decrypt\_password(users[username]) == password:  
 root.destroy()  
 main()  
 else:  
 tk.messagebox.showerror("Ошибка", "Неверное имя пользователя или пароль")  
  
 login\_button = ttk.Button(login\_frame, text="Войти", command=login)  
 login\_button.grid(row=2, column=0, columnspan=2, pady=10)  
  
 *# Register Tab* register\_frame = ttk.Frame(tab\_register)  
 register\_frame.pack(pady=20)  
  
 ttk.Label(register\_frame, text="Имя пользователя:").grid(row=0, column=0, padx=10, pady=10)  
 ttk.Label(register\_frame, text="Пароль:").grid(row=1, column=0, padx=10, pady=10)  
  
 register\_username\_entry = ttk.Entry(register\_frame)  
 register\_password\_entry = ttk.Entry(register\_frame, show="\*")  
  
 register\_username\_entry.grid(row=0, column=1, padx=10, pady=10)  
 register\_password\_entry.grid(row=1, column=1, padx=10, pady=10)  
  
 def register():  
 username = register\_username\_entry.get()  
 password = register\_password\_entry.get()  
 users = load\_users()  
 if username and password:  
 if username not in users:  
 users[username] = encrypt\_password(password)  
 save\_users(users)  
 tk.messagebox.showinfo("Успешно", "Вы успешно зарегистрировались!")  
 else:  
 tk.messagebox.showerror("Ошибка", "Имя пользователя уже занято")  
 else:  
 tk.messagebox.showerror("Ошибка", "Оба поля обязательны для заполнения")  
  
 register\_button = ttk.Button(register\_frame, text="Зарегистрироваться", command=register)  
 register\_button.grid(row=2, column=0, columnspan=2, pady=10)  
  
 root.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 show\_login\_register\_window()

# ChessEngine.py

import copy  
import Minimax  
from select import select  
from shutil import move  
  
class GameState():  
 def \_\_init\_\_(self):  
 self.board = [  
 ["--", "bN", "--", "--", "bK", "--", "bN", "--"],  
 ["--", "--", "--", "--", "--", "--", "--", "--"],  
 ["--", "--", "--", "--", "--", "--", "--", "--"],  
 ["--", "--", "--", "--", "--", "--", "--", "--"],  
 ["--", "--", "--", "--", "--", "--", "--", "--"],  
 ["--", "--", "--", "--", "--", "--", "--", "--"],  
 ["--", "--", "--", "--", "--", "--", "--", "--"],  
 ["--", "--", "--", "--", "wK", "--", "--", "wR"]  
 ]  
 self.moveFunctions = {'p': self.getPawnMoves, 'R': self.getRookMoves, 'N': self.getKnightMoves,  
 'B': self.getBishopMoves, 'Q': self.getQueenMoves, 'K': self.getKingMoves}  
 self.whiteToMove = True  
 self.moveLog = []  
 self.whiteKingLocate = (7, 4)  
 self.blackKingLocate = (0, 4)  
 self.inCheck = False  
 self.checkMate = False  
 self.staleMate = False  
 self.enpassantPossible = ()  
 self.enpassantPossibleLog = [self.enpassantPossible]  
 self.pins = []  
 self.checks = []  
 self.currentCastlingRight = castleRight(True, False, True, True)  
 self.castleRightsLog = [castleRight(self.currentCastlingRight.wks, self.currentCastlingRight.wqs,  
 self.currentCastlingRight.bks, self.currentCastlingRight.bqs)]  
  
  
 def makeMove(self, move):  
 self.board[move.startRow][move.startCol] = "--"  
 self.board[move.endRow][move.endCol] = move.pieceMoved  
 self.moveLog.append(move)  
 self.whiteToMove = not self.whiteToMove  
 if move.pieceMoved == "wK":  
 self.whiteKingLocate = (move.endRow, move.endCol)  
 if move.pieceMoved == "bK":  
 self.blackKingLocate = (move.endRow, move.endCol)  
 *# promotion* if move.isPawnPromotion:  
 piecePromote = 'Q' *#input("Promote R, N, B, Q: ")  
 #piecePromote = piecePromote.upper()* self.board[move.endRow][move.endCol] = move.pieceMoved[0] + piecePromote  
 *# Enpassant Move* if move.isEnpassantMove:  
 self.board[move.startRow][move.endCol] = "--" *# capturing the pawn  
 # EnpassantPossible update* if move.pieceMoved[1] == 'p' and (abs(move.startRow - move.endRow) == 2): *# only 2 squares pawn move* self.enpassantPossible = ((move.startRow + move.endRow) // 2, move.endCol)  
 else:  
 self.enpassantPossible = () *# not en passant move* self.enpassantPossibleLog.append(self.enpassantPossible)  
 *# update castling right - whenever king move or rook move* self.updateCastleRight(move)  
 self.castleRightsLog.append(castleRight(self.currentCastlingRight.wks, self.currentCastlingRight.wqs,  
 self.currentCastlingRight.bks, self.currentCastlingRight.bqs))  
 *# castle move* if move.isCastleMove:  
 if move.endCol-move.startCol == 2:  
 self.board[move.endRow][move.endCol-1] = self.board[move.endRow][move.endCol+1]  
 self.board[move.endRow][move.endCol+1] = "--"  
 else:  
 self.board[move.endRow][move.endCol+1] = self.board[move.endRow][move.endCol-2]  
 self.board[move.endRow][move.endCol-2] = "--"  
  
  
  
 def undoMove(self):  
 if len(self.moveLog) != 0:  
 move = self.moveLog.pop()  
 self.board[move.startRow][move.startCol] = move.pieceMoved  
 self.board[move.endRow][move.endCol] = move.pieceCaptured  
 self.whiteToMove = not self.whiteToMove  
 if move.pieceMoved == "wK":  
 self.whiteKingLocate = (move.startRow, move.startCol)  
 if move.pieceMoved == "bK":  
 self.blackKingLocate = (move.startRow, move.startCol)  
 *#undo enpassant move* if move.isEnpassantMove:  
 self.board[move.endRow][move.endCol] = "--"  
 self.board[move.startRow][move.endCol] = move.pieceCaptured  
  
 self.enpassantPossibleLog.pop()  
 self.enpassantPossible = copy.deepcopy(self.enpassantPossibleLog[-1])  
 *#undo castling right* self.castleRightsLog.pop()  
 self.currentCastlingRight = copy.deepcopy(self.castleRightsLog[-1])  
 *#undo castle move* if move.isCastleMove:  
 if move.endCol - move.startCol == 2:  
 self.board[move.endRow][move.endCol + 1] = self.board[move.endRow][move.endCol - 1]  
 self.board[move.endRow][move.endCol - 1] = "--"  
 else:  
 self.board[move.endRow][move.endCol - 2] = self.board[move.endRow][move.endCol + 1]  
 self.board[move.endRow][move.endCol + 1] = "--"  
  
 self.checkMate = False  
 self.staleMate = False  
  
 def updateCastleRight(self, move):  
 if move.pieceMoved == "wK":  
 self.currentCastlingRight.wks = False  
 self.currentCastlingRight.wqs = False  
 elif move.pieceMoved == "bK":  
 self.currentCastlingRight.bks = False  
 self.currentCastlingRight.bqs = False  
 elif move.pieceMoved == "wR":  
 if move.startRow == 7:  
 if move.startCol == 0:  
 self.currentCastlingRight.wqs = False  
 elif move.startCol == 7:  
 self.currentCastlingRight.wks = False  
 elif move.pieceMoved == "bR":  
 if move.startRow == 0:  
 if move.startCol == 0:  
 self.currentCastlingRight.bqs = False  
 elif move.startCol == 7:  
 self.currentCastlingRight.bks = False  
  
 if move.pieceCaptured == 'wR':  
 if move.endRow == 7:  
 if move.endCol == 0:  
 self.currentCastlingRight.wqs = False  
 elif move.endCol == 7:  
 self.currentCastlingRight.wks = False  
 elif move.pieceCaptured == 'bR':  
 if move.endRow == 0:  
 if move.endCol == 0:  
 self.currentCastlingRight.bqs = False  
 elif move.endCol == 7:  
 self.currentCastlingRight.bks = False  
  
 def check\_only\_kings(self, board):  
 *# Список допустимых фигур* allowed\_figures = {"bK", "wK", "--"}  
  
 for row in board:  
 for piece in row:  
 if piece not in allowed\_figures:  
 return False  
 return True  
  
 def getValidMoves(self):  
 moves = []  
 self.inCheck, self.pins, self.checks = self.checkForPinsAndChecks()  
 if self.whiteToMove:  
 kingRow = self.whiteKingLocate[0]  
 kingCol = self.whiteKingLocate[1]  
 else:  
 kingRow = self.blackKingLocate[0]  
 kingCol = self.blackKingLocate[1]  
  
 if self.inCheck:  
 if len(self.checks) == 1:  
 moves = self.getAllPossibleMoves()  
 check = self.checks[0]  
 checkRow = check[0]  
 checkCol = check[1]  
 pieceChecking = self.board[checkRow][checkCol]  
 validSqs = []  
 if pieceChecking[1] == 'N':  
 validSqs = [(checkRow, checkCol)]  
 else:  
 for i in range(1, 8):  
 validSq = (kingRow + check[2] \* i, kingCol + check[3] \* i)  
 validSqs.append(validSq)  
 if validSq[0] == checkRow and validSq[1] == checkCol:  
 break  
 for i in range(len(moves) - 1, -1, -1):  
 if moves[i].pieceMoved[1] != 'K':  
 if not (moves[i].endRow, moves[i].endCol) in validSqs:  
 moves.remove(moves[i])  
 else:  
 self.getKingMoves(kingRow, kingCol, moves)  
 else:  
 moves = self.getAllPossibleMoves()  
  
 if len(moves) == 0 or self.check\_only\_kings(self.board):  
 if self.inCheck:  
 self.checkMate = True  
 else:  
 self.staleMate = True  
  
 return moves  
  
  
 def getAllPossibleMoves(self):  
 moves = []  
 for r in range(8):  
 for c in range(8):  
 turn = self.board[r][c][0]  
 if (turn == 'w' and self.whiteToMove) or (turn == 'b' and not self.whiteToMove):  
 piece = self.board[r][c][1]  
 self.moveFunctions[piece](r, c, moves)  
 return moves  
  
 def getPawnMoves(self, r, c, moves):  
 piecePinned = False  
 pinDirection = ()  
 for i in range(len(self.pins) - 1, -1, -1):  
 if self.pins[i][0] == r and self.pins[i][1] == c:  
 piecePinned = True  
 pinDirection = (self.pins[i][2], self.pins[i][3])  
 self.pins.remove(self.pins[i])  
 break  
  
 if self.whiteToMove:  
 kingRow = self.whiteKingLocate[0]  
 kingCol = self.whiteKingLocate[1]  
 else:  
 kingRow = self.blackKingLocate[0]  
 kingCol = self.blackKingLocate[1]  
 if self.whiteToMove:  
 if self.board[r-1][c] == "--":  
 if not piecePinned or pinDirection == (-1, 0):  
 moves.append(Move((r, c), (r-1, c), self.board))  
 if r == 6:  
 if self.board[4][c] == "--":  
 moves.append(Move((r, c), (4, c), self.board))  
 if c-1 >= 0:  
 if self.board[r-1][c-1][0] == "b":  
 if not piecePinned or pinDirection == (-1, -1):  
 moves.append(Move((r, c), (r-1, c-1), self.board))  
 elif (r-1, c-1) == self.enpassantPossible:  
 attackingPiece = blockingPiece = False  
 if kingRow == r:  
 if kingCol < c:  
 insideRange = range(kingCol + 1, c-1)  
 outsideRange = range(c+1, 8)  
 else:  
 insideRange = range(kingCol - 1, c, -1)  
 outsideRange = range(c-2, -1, -1)  
 for i in insideRange:  
 if self.board[r][i] != "--":  
 blockingPiece = True  
 for i in outsideRange:  
 square = self.board[r][i]  
 if square[0] == 'b' and (square[1] == 'R' or square[1] == 'Q'):  
 attackingPiece = True  
 elif square != "--":  
 blockingPiece = True  
 if not attackingPiece or blockingPiece:  
 moves.append(Move((r, c), (r - 1, c - 1), self.board, isEnpassantMove=True))  
 if c+1 <= 7:  
 if self.board[r-1][c+1][0] == "b":  
 if not piecePinned or pinDirection == (-1, 1):  
 moves.append(Move((r, c), (r-1, c+1), self.board))  
 elif (r-1, c+1) == self.enpassantPossible:  
 attackingPiece = blockingPiece = False  
 if kingRow == r:  
 if kingCol < c:  
 insideRange = range(kingCol + 1, c)  
 outsideRange = range(c+2, 8)  
 else:  
 insideRange = range(kingCol - 1, c+1, -1)  
 outsideRange = range(c-1, -1, -1)  
 for i in insideRange:  
 if self.board[r][i] != "--":  
 blockingPiece = True  
 for i in outsideRange:  
 square = self.board[r][i]  
 if square[0] == 'b' and (square[1] == 'R' or square[1] == 'Q'):  
 attackingPiece = True  
 elif square != "--":  
 blockingPiece = True  
 if not attackingPiece or blockingPiece:  
 moves.append(Move((r, c), (r-1, c+1), self.board, isEnpassantMove=True))  
 else:  
 if self.board[r+1][c] == "--":  
 if not piecePinned or pinDirection == (1, 0):  
 moves.append(Move((r, c), (r+1, c), self.board))  
 if r == 1:  
 if self.board[3][c] == "--":  
 moves.append(Move((r, c), (3, c), self.board))  
 if c-1 >= 0:  
 if self.board[r+1][c-1][0] == "w":  
 if not piecePinned or pinDirection == (1, -1):  
 moves.append(Move((r, c), (r+1, c-1), self.board))  
 elif (r+1, c-1) == self.enpassantPossible:  
 attackingPiece = blockingPiece = False  
 if kingRow == r:  
 if kingCol < c:  
 insideRange = range(kingCol + 1, c-1)  
 outsideRange = range(c+1, 8)  
 else:  
 insideRange = range(kingCol - 1, c, -1)  
 outsideRange = range(c-2, -1, -1)  
 for i in insideRange:  
 if self.board[r][i] != "--":  
 blockingPiece = True  
 for i in outsideRange:  
 square = self.board[r][i]  
 if square[0] == 'w' and (square[1] == 'R' or square[1] == 'Q'):  
 attackingPiece = True  
 elif square != "--":  
 blockingPiece = True  
 if not attackingPiece or blockingPiece:  
 moves.append(Move((r, c), (r+1, c-1), self.board, isEnpassantMove=True))  
  
 if c+1 <= 7:  
 if self.board[r+1][c+1][0] == "w":  
 if not piecePinned or pinDirection == (1, 1):  
 moves.append(Move((r, c), (r+1, c+1), self.board))  
 elif (r+1, c+1) == self.enpassantPossible:  
 attackingPiece = blockingPiece = False  
 if kingRow == r:  
 if kingCol < c:  
 insideRange = range(kingCol + 1, c)  
 outsideRange = range(c+2, 8)  
 else:  
 insideRange = range(kingCol - 1, c+1, -1)  
 outsideRange = range(c-1, -1, -1)  
 for i in insideRange:  
 if self.board[r][i] != "--":  
 blockingPiece = True  
 for i in outsideRange:  
 square = self.board[r][i]  
 if square[0] == 'w' and (square[1] == 'R' or square[1] == 'Q'):  
 attackingPiece = True  
 elif square != "--":  
 blockingPiece = True  
 if not attackingPiece or blockingPiece:  
 moves.append(Move((r, c), (r+1, c+1), self.board, isEnpassantMove=True))  
  
 def getRookMoves(self, r, c, moves):  
 piecePinned = False  
 pinDirection = ()  
 for i in range(len(self.pins) - 1, -1, -1):  
 if self.pins[i][0] == r and self.pins[i][1] == c:  
 piecePinned = True  
 pinDirection = (self.pins[i][2], self.pins[i][3])  
 if self.board[r][c][1] != 'Q':  
 self.pins.remove(self.pins[i])  
 break  
  
 straightDirections = ((-1, 0), (1, 0), (0, -1), (0, 1))  
 enemyColor = 'b' if self.whiteToMove else 'w'  
 for d in straightDirections:  
 for i in range(1, 8):  
 endRow = r + d[0] \* i  
 endCol = c + d[1] \* i  
 if (0 <= endRow < 8) and (0 <= endCol < 8):  
 if not piecePinned or pinDirection == d or pinDirection == (-d[0], -d[1]):  
 endPiece = self.board[endRow][endCol]  
 if endPiece == "--":  
 moves.append(Move((r, c), (endRow, endCol), self.board))  
 elif endPiece[0] == enemyColor:  
 moves.append(Move((r, c), (endRow, endCol), self.board))  
 break  
 else:  
 break  
 else:  
 break  
  
 def getKnightMoves(self, r, c, moves):  
 piecePinned = False  
 for i in range(len(self.pins) - 1, -1, -1):  
 if self.pins[i][0] == r and self.pins[i][1] == c:  
 piecePinned = True  
 self.pins.remove(self.pins[i])  
 break  
  
 knightDirections = ((-2, -1), (-1, -2), (1, -2), (2, -1), (2, 1), (1, 2), (-1, 2), (-2, 1))  
 allyColor = 'w' if self.whiteToMove else 'b'  
 for d in knightDirections:  
 endRow = r + d[0]  
 endCol = c + d[1]  
 if (0 <= endRow < 8) and (0 <= endCol < 8):  
 if not piecePinned:  
 endPiece = self.board[endRow][endCol]  
 if endPiece[0] != allyColor:  
 moves.append(Move((r, c), (endRow, endCol), self.board))  
  
 def getBishopMoves(self, r, c, moves):  
 piecePinned = False  
 pinDirection = ()  
 for i in range(len(self.pins) - 1, -1, -1):  
 if self.pins[i][0] == r and self.pins[i][1] == c:  
 piecePinned = True  
 pinDirection = (self.pins[i][2], self.pins[i][3])  
 self.pins.remove(self.pins[i])  
 break  
  
 diagonalDirections = ((-1, -1), (1, 1), (1, -1), (-1, 1))  
 enemyColor = 'b' if self.whiteToMove else 'w'  
 for d in diagonalDirections:  
 for i in range(1, 8):  
 endRow = r + d[0] \* i  
 endCol = c + d[1] \* i  
 if (0 <= endRow < 8) and (0 <= endCol < 8):  
 if not piecePinned or pinDirection == d or pinDirection == (-d[0], -d[1]):  
 endPiece = self.board[endRow][endCol]  
 if endPiece == "--":  
 moves.append(Move((r, c), (endRow, endCol), self.board))  
 elif endPiece[0] == enemyColor:  
 moves.append(Move((r, c), (endRow, endCol), self.board))  
 break  
 else:  
 break  
 else:  
 break  
  
 def getQueenMoves(self, r, c, moves):  
 self.getRookMoves(r, c, moves)  
 self.getBishopMoves(r, c, moves)  
  
 def getKingMoves(self, r, c, moves):  
 kingDirections = ((-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (1, 1), (1, -1), (-1, 1))  
 allyColor = 'w' if self.whiteToMove else 'b'  
 for i in range(8):  
 endRow = r + kingDirections[i][0]  
 endCol = c + kingDirections[i][1]  
 if (0 <= endRow < 8) and (0 <= endCol < 8):  
 endPiece = self.board[endRow][endCol]  
 if endPiece[0] != allyColor:  
 *# place king on the end of square to check* if allyColor == 'w':  
 self.whiteKingLocate = (endRow, endCol)  
 else:  
 self.blackKingLocate = (endRow, endCol)  
 inCheck, pins, checks = self.checkForPinsAndChecks()  
 if not inCheck:  
 moves.append(Move((r, c), (endRow, endCol), self.board))  
 *# place king back* if allyColor == 'w':  
 self.whiteKingLocate = (r, c)  
 else:  
 self.blackKingLocate = (r, c)  
 self.getCastleMoves(r, c, moves, allyColor)  
  
 def checkForPinsAndChecks(self):  
 pins = []  
 checks = []  
 inCheck = False  
 if self.whiteToMove:  
 enemyColor = 'b'  
 allyColor = 'w'  
 startRow = self.whiteKingLocate[0]  
 startCol = self.whiteKingLocate[1]  
 else:  
 enemyColor = 'w'  
 allyColor = 'b'  
 startRow = self.blackKingLocate[0]  
 startCol = self.blackKingLocate[1]  
  
 directions = ((-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, 1), (1, -1))  
 for j in range(len(directions)):  
 d = directions[j]  
 possiblePins = ()  
 for i in range(1, 8):  
 endRow = startRow + d[0] \* i  
 endCol = startCol + d[1] \* i  
 if 0 <= endRow < 8 and 0 <= endCol < 8:  
 endPiece = self.board[endRow][endCol]  
 if endPiece[0] == allyColor and endPiece[1] != 'K':  
 if possiblePins == ():  
 possiblePins = (endRow, endCol, d[0], d[1])  
 else:  
 break  
 elif endPiece[0] == enemyColor:  
 pieceType = endPiece[1]  
 if (0 <= j <= 3 and pieceType == 'R') or \  
 (4 <= j <= 7 and pieceType == 'B') or \  
 (i == 1 and pieceType == 'p' and ((enemyColor == 'b' and 4 <= j <= 5) or (enemyColor == 'w' and 6 <= j <= 7))) or \  
 (pieceType == 'Q') or (i == 1 and pieceType == 'K'):  
 if possiblePins == ():  
 inCheck = True  
 checks.append((endRow, endCol, d[0], d[1]))  
 break  
 else:  
 pins.append(possiblePins)  
 break  
 else:  
 break  
 else:  
 break  
  
 knightDirections = ((-2, -1), (-1, -2), (1, -2), (2, -1), (2, 1), (1, 2), (-1, 2), (-2, 1))  
 for m in knightDirections:  
 endRow = startRow + m[0]  
 endCol = startCol + m[1]  
 if 0 <= endRow < 8 and 0 <= endCol < 8:  
 endPiece = self.board[endRow][endCol]  
 if (endPiece[0] == enemyColor) and (endPiece[1] == 'N'):  
 inCheck = True  
 checks.append((endRow, endCol, m[0], m[1]))  
 break  
  
 return inCheck, pins, checks  
  
 def getCastleMoves(self, r, c, moves, allyColor=""):  
 if self.inCheck:  
 return *#can't castle while be checked* if (self.whiteToMove and self.currentCastlingRight.wks) or (not self.whiteToMove and self.currentCastlingRight.bks):  
 self.getKingsideCastleMove(r, c, moves, allyColor)  
 if (self.whiteToMove and self.currentCastlingRight.wqs) or (not self.whiteToMove and self.currentCastlingRight.bqs):  
 self.getQueensideCastleMove(r, c, moves, allyColor)  
  
 def getKingsideCastleMove(self, r, c, moves, allyColor=""):  
 if self.board[r][c+1] == "--" and self.board[r][c+2] == "--":  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c+1)  
 else:  
 self.blackKingLocate = (r, c+1)  
 inCheck1, pins, checks = self.checkForPinsAndChecks()  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c+2)  
 else:  
 self.blackKingLocate = (r, c+2)  
 inCheck2, pins, checks = self.checkForPinsAndChecks()  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c)  
 else:  
 self.blackKingLocate = (r, c)  
  
 if not inCheck1 and not inCheck2:  
 moves.append(Move((r, c), (r, c+2), self.board, isCastleMove=True))  
  
 def getQueensideCastleMove(self, r, c, moves, allyColor=""):  
 if self.board[r][c - 1] == "--" and self.board[r][c - 2] == "--" and self.board[r][c-3] == "--":  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c-1)  
 else:  
 self.blackKingLocate = (r, c-1)  
 inCheck1, pins, checks = self.checkForPinsAndChecks()  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c-2)  
 else:  
 self.blackKingLocate = (r, c-2)  
 inCheck2, pins, checks = self.checkForPinsAndChecks()  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c-3)  
 else:  
 self.blackKingLocate = (r, c-3)  
 inCheck3, pins, checks = self.checkForPinsAndChecks()  
 if allyColor == 'w':  
 self.whiteKingLocate = (r, c)  
 else:  
 self.blackKingLocate = (r, c)  
  
 if not inCheck1 and not inCheck2 and not inCheck3:  
 moves.append(Move((r, c), (r, c-2), self.board, isCastleMove=True))  
  
class castleRight():  
 def \_\_init\_\_(self, wks, wqs, bks, bqs):  
 self.wks = wks  
 self.bks = bks  
 self.wqs = wqs  
 self.bqs = bqs  
  
  
  
class Move():  
 ranksToRows = {"1": 7, "2": 6, "3": 5, "4": 4, "5": 3, "6": 2, "7": 1, "8": 0}  
 rowsToRanks = {v: k for k, v in ranksToRows.items()}  
 filesToCols = {"a": 0, "b": 1, "c": 2, "d": 3, "e": 4, "f": 5, "g": 6, "h": 7}  
 colsToFiles = {v: k for k, v in filesToCols.items()}  
  
 def \_\_init\_\_(self, startSq, endSq, board, isEnpassantMove=False, isCastleMove=False):  
 self.startRow = startSq[0]  
 self.startCol = startSq[1]  
 self.endRow = endSq[0]  
 self.endCol = endSq[1]  
 self.pieceMoved = board[self.startRow][self.startCol]  
 self.pieceCaptured = board[self.endRow][self.endCol]  
 self.isPawnPromotion = (self.pieceMoved == "wp" and self.endRow == 0) or (self.pieceMoved == "bp" and self.endRow == 7)  
 self.isEnpassantMove = isEnpassantMove  
 if self.isEnpassantMove:  
 self.pieceCaptured = "wp" if self.pieceMoved == "bp" else "bp"  
 self.isCastleMove = isCastleMove  
 self.isCapture = (self.pieceCaptured != "--")  
 self.moveID = self.startCol \* 1000 + self.startRow \* 100 + self.endCol \* 10 + self.endRow  
  
 def \_\_eq\_\_(self, other):  
 if isinstance(other, Move):  
 return self.moveID == other.moveID  
 return False  
  
 def getChessNotation(self):  
 return self.getFileRank(self.startRow, self.startCol) + self.getFileRank(self.endRow, self.endCol)  
 def getFileRank(self, row, col):  
 return str(self.colsToFiles[col] + self.rowsToRanks[row])  
  
 def \_\_str\_\_(self):  
 *#castleMove:* if self.isCastleMove:  
 return "O-O" if self.endCol == 6 else "O-O-O"  
 endSquare = self.getFileRank(self.endRow, self.endCol)  
 *#pawn move* if self.pieceMoved[1] == 'p':  
 if self.isCapture:  
 return self.colsToFiles[self.startCol] + "x" + endSquare  
 else:  
 return endSquare  
  
 *#piece moves* moveString = self.pieceMoved[1]  
 if self.isCapture:  
 moveString += "x"  
 return moveString + endSquare

# Minimax.py

import random  
import time  
  
  
pieceScore = {"K": 0, "Q": 90, "R": 50, "B": 35, "N": 30, "p": 10}  
  
knightScore = [[1, 1, 1, 1, 1, 1, 1, 1],  
 [1, 2, 2, 2, 2, 2, 2, 1],  
 [1, 2, 3, 3, 3, 3, 2, 1],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [1, 2, 3, 3, 3, 3, 2, 1],  
 [1, 2, 2, 2, 2, 2, 2, 1],  
 [1, 1, 1, 1, 1, 1, 1, 1]]  
  
bishopScore = [[4, 3, 2, 1, 1, 2, 3, 4],  
 [3, 4, 3, 2, 2, 3, 4, 3],  
 [2, 3, 4, 3, 3, 4, 3, 2],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [2, 3, 4, 3, 3, 4, 3, 2],  
 [3, 4, 3, 2, 2, 3, 4, 3],  
 [4, 3, 2, 1, 1, 2, 3, 4]]  
  
queenScore = [[1, 1, 1, 3, 1, 1, 1, 1],  
 [1, 2, 3, 3, 3, 1, 1, 1],  
 [1, 4, 3, 3, 3, 4, 2, 1],  
 [1, 2, 3, 3, 3, 2, 2, 1],  
 [1, 2, 3, 3, 3, 2, 2, 1],  
 [1, 4, 3, 3, 3, 4, 2, 1],  
 [1, 2, 3, 3, 3, 1, 1, 1],  
 [1, 1, 1, 3, 1, 1, 1, 1]]  
  
rookScore = [[4, 3, 4, 4, 4, 4, 3, 4],  
 [4, 4, 4, 4, 4, 4, 4, 4],  
 [1, 1, 2, 3, 3, 2, 1, 1],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [1, 1, 2, 3, 3, 2, 1, 1],  
 [4, 4, 4, 4, 4, 4, 4, 4],  
 [4, 3, 4, 4, 4, 4, 3, 4]]  
  
whitePawnScore = [[8, 8, 8, 8, 8, 8, 8, 8],  
 [8, 8, 8, 8, 8, 8, 8, 8],  
 [5, 6, 6, 7, 7, 6, 6, 5],  
 [2, 3, 3, 5, 5, 3, 3, 2],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [1, 2, 3, 3, 3, 3, 2, 1],  
 [1, 1, 1, 0, 0, 1, 1, 1],  
 [0, 0, 0, 0, 0, 0, 0, 0]]  
  
blackPawnScore = [[0, 0, 0, 0, 0, 0, 0, 0],  
 [1, 1, 1, 0, 0, 1, 1, 1],  
 [1, 2, 3, 3, 3, 3, 2, 1],  
 [1, 2, 3, 4, 4, 3, 2, 1],  
 [2, 3, 3, 5, 5, 3, 3, 2],  
 [5, 6, 6, 7, 7, 6, 6, 5],  
 [8, 8, 8, 8, 8, 8, 8, 8],  
 [8, 8, 8, 8, 8, 8, 8, 8]]  
  
piecePosScores = {'N': knightScore, 'B': bishopScore, 'Q': queenScore, 'R': rookScore, "wp": whitePawnScore, "bp": blackPawnScore}  
  
  
CHECKMATE = 100000  
STALEMATE = 0  
MAX\_DEPTH = 4  
  
''' Find random move for AI '''  
def findRandomMove(validMoves):  
 if len(validMoves) > 0:  
 return validMoves[random.randint(0, len(validMoves)-1)]  
  
  
def findBestMoveMinimax(gs, validMoves):  
 global nextMove  
 global nodes  
 nextMove = None  
 alpha = -CHECKMATE  
 beta = CHECKMATE  
 nodes = 0  
 start\_time = time.time()  
 findMoveMinimax(gs, validMoves, MAX\_DEPTH, alpha, beta, gs.whiteToMove)  
 end\_time = time.time()  
 elapsed\_time = end\_time - start\_time  
 print ("elapsed\_time:{0}".format(elapsed\_time) + "[sec]")  
 print(nodes)  
 return nextMove  
  
  
def findMoveMinimax(gs, validMoves, depth, alpha, beta, whiteToMove):  
 global nextMove  
 global nodes  
 nodes += 1  
 if depth == 0 or gs.checkMate or gs.staleMate:  
 return scoreBoard(gs)  
 random.shuffle(validMoves)  
 if whiteToMove:  
 maxScore = -CHECKMATE  
 random.shuffle(validMoves)  
 for move in validMoves:  
 gs.makeMove(move)  
 nextMoves = gs.getValidMoves()  
 score = findMoveMinimax(gs, nextMoves, depth - 1, alpha, beta, False)  
 gs.undoMove()  
 if score > maxScore:  
 maxScore = score  
 if depth == MAX\_DEPTH:  
 nextMove = move  
 alpha = max(alpha, score)  
 if beta <= alpha:  
 break  
 return maxScore  
 else:  
 minScore = CHECKMATE  
 random.shuffle(validMoves)  
 for move in validMoves:  
 gs.makeMove(move)  
 nextMoves = gs.getValidMoves()  
 score = findMoveMinimax(gs, nextMoves, depth - 1, alpha, beta, True)  
 gs.undoMove()  
 if score < minScore:  
 minScore = score  
 if depth == MAX\_DEPTH:  
 nextMove = move  
 beta = min(beta, score)  
 if beta <= alpha:  
 break  
 return minScore  
  
def scoreBoard(gs):  
 if gs.checkMate:  
 if gs.whiteToMove:  
 return -CHECKMATE *#black win* else:  
 return CHECKMATE *#white win* elif gs.staleMate:  
 return STALEMATE  
 *# if not checkmate or stalemate:* score = 0  
 for row in range(8):  
 for col in range(8):  
 square = gs.board[row][col]  
 if square != "--":  
 piecePosScore = 0  
 if square[1] != "K":  
 if square[1] == "p":  
 piecePosScore = piecePosScores[square][row][col]  
 else:  
 piecePosScore = piecePosScores[square[1]][row][col]  
 if square[0] == 'w':  
 score += pieceScore[square[1]] + piecePosScore  
 elif square[0] == 'b':  
 score -= pieceScore[square[1]] + piecePosScore  
 return score

**2. Источники, использованные при разработке**

1. В.В. Шишкин, Д.С. Афонин РАЗРАБОТКА ЛОГИЧЕСКИХ КОМПЬЮТЕРНЫХ ИГР С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ В СРЕДЕ ПИТОН
2. ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». – М.: Издательство стандартов, 1990.
3. Pygame Documentation. [Электронный ресурс]. - Режим доступа: https://www.pygame.org/docs/
4. Python Documentation. [Электронный ресурс]. - Режим доступа: https://docs.python.org/3/
5. Python 3. Самое необходимое / Н.А. Прохоренок, В.А. Дронов. – 2-е изд.,перераб. и доп. – СПб.: БХВ-Петербург,2020. – 608 с.: ил. – (Самое необходимое)
6. Руководство по Pygame [Электронный ресурс]. – Режим доступа: https://pythonist.ru/pygame-tutorial/
7. Pygame [Электронный ресурс]. – Режим доступа: https://egoroffartem.pythonanywhere.com/course/pygame/
8. Образовательная платформа Stepik [Электронный ресурс]. – Режим доступа: https://stepik.org/learn

**Заключение**

Разработанное приложение "Эндшпиль: Король и Ладья против Короля и Двух Коней" успешно реализует функционал шахматной игры с возможностью игры против компьютера и против человека. Приложение демонстрирует интуитивно понятный интерфейс и высокую степень интерактивности, что делает его доступным для игроков с различным уровнем подготовки.

В ходе разработки были достигнуты ключевые цели, такие как создание графического интерфейса, реализация логики игры, а также внедрение системы регистрации и аутентификации пользователей. Использование алгоритма Минимакс для AI позволяет обеспечить интересный и конкурентоспособный игровой процесс.

В будущем приложение может быть расширено дополнительными функциями, такими как:

Поддержка различных шахматных режимов (например, турниры, обучение).

Улучшение AI с использованием более сложных алгоритмов.

Внедрение сетевой игры для возможности сражений между пользователями через интернет.

Добавление статистики и рейтингов игроков для повышения мотивации и вовлеченности.

**Приложение 1**

