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, пешка – Король, пешка»**

Руководитель разработки:  
Кандидат технических наук,

Доцент кафедры ИВК  
Шишкин Вадим Викторинович

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

Исполнитель:

студент гр. ИСТбд-23

Никольский Максим Сергеевич

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**Содержание**

Оглавление

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc185984538)

[**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА** 10](#_Toc185984539)

[**РУКОВОДСТВО ПРОГРАММИСТА** 26](#_Toc185984540)

[**Листинг программы** 47](#_Toc185984541)

[**2. Источники, использованные при разработке** 73](#_Toc185984542)

[**Приложение 1** 74](#_Toc185984543)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

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

**Компьютерная игра эндшпиль «Король, пешка – Король, пешка»**

Р.02069337. №23/748-Вариант 17

Листов 6

Исполнитель:

студент гр. ИСТбд-23

Никольский Максим Сергеевич

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**Введение**

**Наименование и условное обозначение приложения:**

*Название:* Chess Endgame

*Условное обозначение:* CE

**Реализуемая игра:**

*Название игры:* Эндшпиль «Король, пешка – Король, пешка»

**Краткий свод правил игры:**

Эндшпиль «Король, пешка – Король, пешка» является одной из классических позиций в шахматах, где у каждой стороны остаются король и одна пешка. Цель игрока состоит в том, чтобы продвинуть свою пешку до последней горизонтали доски (до поля превращения), защитить её от короля противника и превратить пешку в ферзя или другую фигуру, а затем поставить мат королю соперника.

1. **Основание для разработки**

Основание: учебный план направления 09.03.02 "Информационные системы и технологии" и распоряжение по факультету.

1. **Требования к программе или программному изделию**
   1. Функциональное назначение

**Функциональное назначение приложения:**

Приложение Chess Endgame (CE) предназначено для автоматизации процесса обучения и тренировки шахматного эндшпиля «Король, пешка – Король, пешка». Оно позволяет пользователям играть против искусственного интеллекта, анализировать свои партии и изучать теорию эндшпиля.

**Перечень автоматизируемых процессов:**

1. *Генерация ходов ИИ:*

* Расчет возможных ходов для ИИ с использованием алгоритмов поиска решений (Minimax).
* Оценка позиции на доске и выбор лучшего хода.

1. *Взаимодействие с пользователем:*

* Прием и обработка входных данных от пользователя (ходов).
* Проверка корректности введенных ходов.

1. *Графический интерфейс:*

* Отображение шахматной доски и фигур.
* Анимация движений фигур.
* Индикация текущего хода (чья очередь ходить).

1. *Авторизация:*

* Запрашивание учетных данных пользователя.
* Шифрование и проверка введенного пароля.

1. *База данных:*

* Хранение информации о сыгранных партиях.

**Группы пользователей:**

1. Новички:

* Пользователи, начинающие изучение шахмат и желающих освоить базовые принципы эндшпиля. Нуждаются в обучении основам игры.

1. Средний уровень:

* Шахматисты, имеющие опыт игры, но желающие улучшить навыки в эндшпиле. Используют приложение для тренировок и анализа своих партий.

1. Продвинутые игроки:

* Опытные шахматисты, стремящиеся углубить знания в области эндшпиля. Применяют приложение для анализа сложных позиций и подготовки к соревнованиям.
  1. Требования к функциональным характеристикам
     1. Требования к структуре приложения

1. Модульная структура

* Модули должны быть независимыми и взаимозаменяемыми. Каждый модуль должен выполнять одну конкретную задачу и взаимодействовать с другими модулями через четко определенные интерфейсы.
* Минимальная зависимость между модулями. Минимизировать количество зависимостей между модулями, чтобы изменения в одном модуле минимально влияли на другие.
* Разделение ответственности. Разделять функционал на отдельные модули, чтобы каждый модуль отвечал за строго определённую часть приложения.

1. Основные модули

* Логика игры (Game Logic):

Содержит классы и методы, отвечающие за управление состоянием игры, проверку допустимости ходов, реализацию правил шахматного эндшпиля. Должен включать в себя классы для представления шахматной доски, фигур, а также алгоритмы для генерации и оценки ходов.

* Искусственный интеллект (AI Engine):

Включает алгоритмы для расчета ходов ИИ.

* Пользовательский интерфейс (UI):

Отвечает за отображение игровой доски, фигур и других элементов интерфейса. Управляет взаимодействием с пользователем, включая обработку кликов мыши и нажатий клавиш.

* Авторизация и шифрование:

Реализует механизмы авторизации пользователя и шифрования данных.

Предоставляет API для проверки подлинности пользователя и защиты конфиденциальной информации.

* Анализ партий (Analysis):

Может включать в себя базу данных для хранения информации о сыгранных партиях.

2.2.2 Требования к составу функций приложения

1. Игра против ИИ:

* Алгоритм Minimax: Реализация алгоритма для поиска оптимального хода ИИ.
* Графическая визуализация: Отображение шахматной доски и фигур, анимация ходов.

2. Авторизация и защита данных:

* Шифрование и аутентификация: Механизмы для безопасного входа в приложение и защиты личных данных пользователя.

1. Анализ партий:

* Статистика: Сбор и представление статистических данных о сыгранных партиях.

4. Сохранение прогресса:

* Автоматическое сохранение: Возможность сохранить текущую позицию на доске для продолжения игры позже.
* Загрузка сохранённых партий: Возможность возобновления прерванной игры.
  + 1. Требования к организации информационного обеспечения, входных и выходных данных

Управление осуществляется мышью.

* 1. Требования к надежности

Приложение должно стабильно функционировать при нормальном режиме эксплуатации, избегая неожиданных сбоев и аварийных завершений. Перезагрузка - как способ восстановления.

* 1. Требования к информационной и программной совместимости

1. *Операционная система:*

* Windows 10 или выше
* macOS Catalina (10.15) или выше
* Ubuntu 20.04 LTS или выше

1. *Библиотеки:*

* pygame 2.0+
* math (встроена в стандартную библиотеку Python)
* bcrypt 3.2.2+
* sqlite3 (встроена в стандартную библиотеку Python)
* os (встроена в стандартную библиотеку Python)
* json (встроена в стандартную библиотеку Python)
* datetime (встроена в стандартную библиотеку Python)
* copy (встроена в стандартную библиотеку Python)
* sys (встроена в стандартную библиотеку Python)

1. *База данных:*

* SQLite 3.33+

1. *Язык программирования:*

* Python 3.8+

1. *Среда разработки:*

* PyCharm Community Edition (2021.1 или выше)
  1. Требования к маркировке и упаковке

Определяются заданием на курсовую работу.

* 1. Требования к транспортированию и хранению
     1. Условия транспортирования

Требования к условиям транспортирования не предъявляются.

* + 1. Условия хранения

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

* + 1. Сроки хранения

Срок хранения – до окончания срока учебы.

1. **Требования к программной документации**

Определяются заданием на курсовую работу.

1. **Стадии и этапы разработки**

Определяются заданием на курсовую работу.

1. **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, пешка – Король, пешка»**

Р.02069337. №23-748 Вариант 17

Листов 42

Исполнитель:

студент гр. ИСТбд-23

Никольский Максим Сергеевич

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**Введение**

В данной курсовой работе разрабатывается приложение "Компьютерный эндшпиль: Король, пешка - Король, пешка". Это приложение представляет собой реализацию шахматного эндшпиля, в котором игрок играет против искусственного интеллекта (ИИ) с использованием фигур "король" и "пешка". Цель игры — продемонстрировать возможности ИИ в решении шахматных задач, а также предоставить пользователю возможность играть в эндшпиль с минимальным набором фигур.

Выбранный подход к реализации приложения основан на использовании алгоритма минимакс с альфа-бета отсечениями для анализа возможных ходов и оценки текущего состояния игры. Такой подход позволяет ИИ находить оптимальные ходы, минимизируя количество рассматриваемых вариантов.

Реализованное приложение включает в себя графический интерфейс, поддержку регистрации и входа пользователей, а также возможность сохранения и загрузки игр. Приложение разработано на языке Python с использованием библиотеки Pygame для графического интерфейса и SQLite для хранения данных.

**1. Проектная часть**

**1.1. Постановка задачи на разработку приложения**

Задача на разработку приложения определяется заданием на курсовую работу. Детализируется в разработанном техническом задании. Приложение должно реализовать игру в шахматный эндшпиль с участием короля и пешки. Игрок играет за белых, а искусственный интеллект — за чёрных. Приложение должно включать функциональность регистрации и входа пользователей, сохранения и загрузки игр, а также экспорта игры в формате PGN.

**1.2. Математические методы**

Для реализации искусственного интеллекта используется алгоритм Minimax с альфа-бета отсечением. Этот алгоритм позволяет оценить наилучший ход для искусственного интеллекта на основе оценки текущего состояния игры. Оценка состояния игры производится на основе разницы в количестве фигур на доске.

1. Алгоритм Minimax:

* Генерируются все возможные ходы для текущего игрока.
* Для каждого хода оценивается состояние игры после хода.
* Альфа-бета отсечение сокращает количество рассматриваемых ходов, исключая те, которые не могут повлиять на результат.
* На каждом уровне рекурсии алгоритм выбирает ход, который максимизирует или минимизирует оценку в зависимости от того, чей ход (белых или чёрных).

1. Оценка состояния игры:

Для оценки текущего состояния игры используется простая функция, которая учитывает разницу в количестве фигур и их ценность. Каждая фигура имеет определённую ценность:

* Король: 1000
* Ферзь: 9
* Ладья: 5
* Слон: 3
* Конь: 3
* Пешка: 1.

Это позволяет алгоритму принимать решения на основе текущей позиции.

Выбор данных методов обоснован их эффективностью в решении задач анализа игровых ситуаций, а также их широкой распространённостью в шахматных программах.

**1.3. Архитектура и алгоритмы**

1.3.1 Архитектура

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

1. auth.py: Отвечает за аутентификацию пользователей (регистрация и вход). Использует библиотеку bcrypt для хеширования паролей.

* Функции модуля:
* register (username, password): Регистрация нового пользователя.
* Проверяет длину имени пользователя и пароля.
* Хеширует пароль с использованием bcrypt.
* Сохраняет пользователя в базе данных.
* login (username, password): Вход пользователя в систему.
* Проверяет наличие пользователя в базе данных.
* Сравнивает хешированный пароль с введённым паролем.
* Возвращает результат входа.

1. database.py: Обрабатывает взаимодействие с базой данных SQLite, где хранятся пользователи и игры.

* Функции модуля:
* initialize\_db (): Инициализация базы данных.
* Создаёт таблицы users и games, если они не существуют.
* save\_user (username, password\_hash): Сохранение пользователя в базе данных.
* Вставляет данные пользователя в таблицу users.
* get\_user (username): Получение данных пользователя.
* Возвращает данные пользователя по имени.
* create\_new\_game (white\_player, black\_player): Создание новой игры.
* Вставляет новую игру в таблицу games.
* update\_game (game\_id, moves, result, end\_time, status): Обновление состояния игры.
* Обновляет данные игры в таблице games.
* get\_games\_by\_user (username, status): Получение списка игр пользователя.
* Возвращает список игр пользователя по его имени и статусу.
* get\_game\_by\_id (game\_id): Получение игры по её ID.
* Возвращает данные игры по её идентификатору.

1. game.py: Реализует логику игры, включая управление доской, ходы фигур и проверку состояния игры.

* Класс Move:
* Свойства:
* start\_row, start\_col: Начальная позиция хода.
* end\_row, end\_col: Конечная позиция хода.
* piece\_moved: Фигура, которая совершает ход.
* piece\_captured: Фигура, которая была захвачена.
* is\_pawn\_promotion: Флаг, указывающий на превращение пешки.
* promotion\_choice: Выбор фигуры для превращения пешки.
* Методы:
* \_\_eq\_\_ (self, other): Сравнение двух ходов.
* get\_chess\_notation (self): Возвращает шахматную нотацию хода.
* to\_dict (self): Сериализация хода в словарь.
* from\_dict (cls, move\_dict): Десериализация хода из словаря.
* Класс Game:
* Свойства:
* white\_player, black\_player: Имена игроков.
* game\_id: Идентификатор игры.
* board: Текущее состояние доски.
* white\_to\_move: Флаг, указывающий, чей ход (белых или чёрных).
* move\_log: История ходов.
* selected\_square: Выбранная клетка.
* valid\_moves: Допустимые ходы.
* checkmate, stalemate: Флаги, указывающие на мат или пат.
* en\_passant\_possible: Возможность взятия на проходе.
* promotion\_choice: Выбор фигуры для превращения пешки.
* start\_time, end\_time: Время начала и окончания игры.
* result: Результат игры.
* Методы:
* \_\_init\_\_ (self, white\_player, black\_player, game\_id): Инициализация игры.
* create\_initial\_board (self): Создание начальной доски.
* load\_game (self, game\_id): Загрузка игры из базы данных.
* reconstruct\_board (self): Восстановление доски из истории ходов.
* make\_move (self, move, update\_state): Выполнение хода.
* undo\_move (self): Отмена последнего хода.
* get\_valid\_moves (self): Получение всех допустимых ходов.
* get\_all\_possible\_moves (self): Получение всех возможных ходов.
* in\_check (self, white): Проверка, находится ли король под шахом.
* check\_game\_state (self): Проверка состояния игры (шах, мат, пат).
* save\_current\_game (self): Сохранение текущего состояния игры.
* save\_game\_completion (self): Сохранение завершённой игры.
* export\_pgn (self): Экспорт игры в формате PGN.
* draw (self, win, images, selected\_square, valid\_moves): Отрисовка игры.
* draw\_board (self, win, selected\_square, valid\_moves): Отрисовка доски.
* draw\_pieces (self, win, images): Отрисовка фигур.
* draw\_game\_state (self, win): Отрисовка состояния игры.
* is\_move\_valid (self, move): Проверка, является ли ход допустимым.
* get\_piece\_moves (self, r, c): Получение всех ходов для фигуры на заданной позиции.

1. ai.py: Содержит алгоритм Minimax для искусственного интеллекта.

* Функции модуля:
* find\_best\_move (game, depth): Поиск наилучшего хода для искусственного интеллекта.
* minimax (game, depth, alpha, beta, is\_maximizing): Рекурсивный алгоритм Minimax с альфа-бета отсечением.
* evaluate\_game (game): Оценка текущего состояния игры.

1. main.py: Главный модуль, который управляет запуском приложения и взаимодействием с пользователем.

* Функции модуля:
* auth\_screen (): Экран аутентификации.
* login\_prompt (): Ввод данных для входа.
* register\_prompt (): Ввод данных для регистрации.
* select\_mode (username): Выбор режима игры.
* view\_games (username): Просмотр списка игр.
* resume\_game (game): Возобновление игры.
* game\_screen (mode, white\_player, black\_player): Экран игры.
* main (): Главная функция, запускающая приложение.

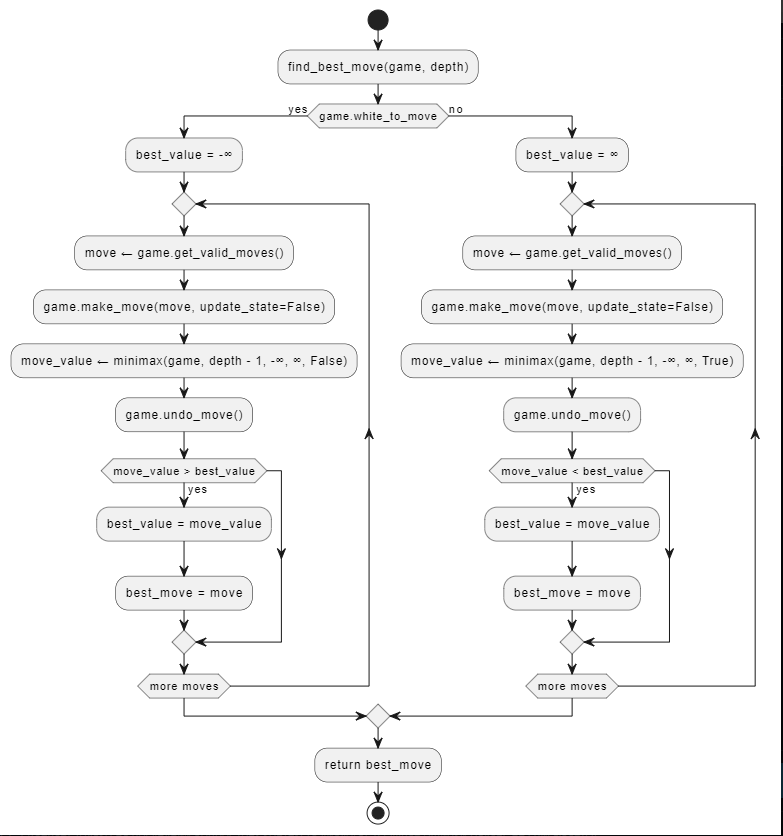
1. settings.py: Хранит основные настройки приложения, такие как размеры окна, цвета и путь к ресурсам.

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:

* Инициализация: устанавливается начальное состояние игры и глубина поиска.
* Рекурсивный анализ ходов: для каждого возможного хода выполняется рекурсивный анализ, где оцениваются возможные ответы противника.
* Оценка состояния игры: для каждого состояния игры вычисляется оценка с использованием функции evaluate\_game.
* Альфа-бета отсечения: если текущая оценка хуже уже найденной, дальнейший анализ ветви прекращается.
* Выбор хода: выбирается ход с наилучшей оценкой.

Блок-схема алгоритма:



Комментарии:

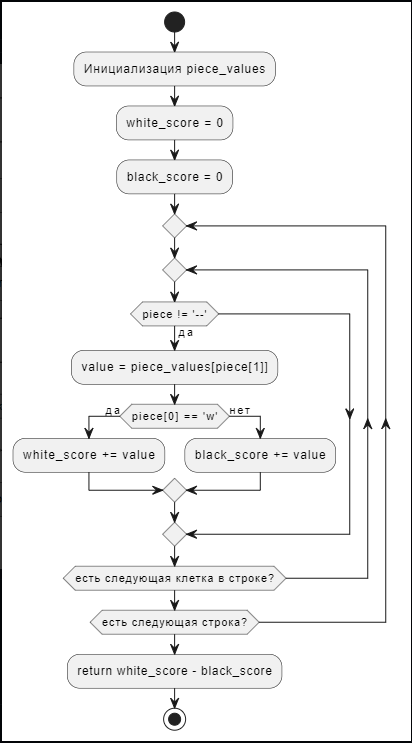
* Алгоритм позволяет эффективно анализировать большое количество возможных ходов, используя отсечения для уменьшения количества рассматриваемых вариантов.
* Оценка состояния игры основана на разнице в количестве фигур, что позволяет алгоритму принимать решения на основе текущей позиции.

1.3.3 Алгоритм оценки состояния игры

Алгоритм оценки состояния игры используется для вычисления текущего состояния игры на основе разницы в количестве фигур. Основные шаги алгоритма:

* Инициализация: устанавливаются начальные значения для оценки.
* Анализ доски: для каждой фигуры на доске вычисляется её ценность.
* Вычисление разницы: разница в ценности фигур используется как оценка текущего состояния игры.

Блок-схема алгоритма:



Комментарии:

* Алгоритм прост в реализации и позволяет быстро оценивать текущее состояние игры.
* Ценность фигур может быть настроена для улучшения качества игры.

**1.4. Тестирование**

**Mind map**

Для тестирования игры можно организовать карту разума, чтобы структурировать процесс (Приложение 1)

**Чек лист**

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функция | Шаги выполнения | Ожидаемый результат | Статус |
| 1. Аутентификация и авторизация (auth.py) |  |  |  |
| Регистрация нового пользователя | Ввести имя < 3 символов, пароль < 6 символов | Пользователь не регистрируется, выводится ошибка | Пройдено/Не пройдено |
| Вход в систему | Ввести неправильный пароль | Вход не выполняется, выводится ошибка | Пройдено/Не пройдено |
| Хеширование паролей | Сохранить пароль в открытом виде | Пароль должен быть захеширован | Пройдено/Не пройдено |
| 2. База данных (database.py) |  |  |  |
| Инициализация базы данных | Запустить программу | База данных создана с таблицами users и games | Пройдено/Не пройдено |
| Сохранение игры | Завершить игру | Игра сохранена в базе данных | Пройдено/Не пройдено |
| Получение данных | Найти пользователя по имени | Данные пользователя корректно получены | Пройдено/Не пройдено |
| 3. Логика игры (game.py) |  |  |  |
| Инициализация игры | Запустить новую игру | Начальная доска корректно инициализирована | Пройдено/Не пройдено |
| Обработка ходов игрока | Пешка ходит назад | Ход не выполняется, выводится ошибка | Пройдено/Не пройдено |
| Состояние игры (шах, мат, пат) | Установить позицию мата | Игра завершается с сообщением "Мат" | Пройдено/Не пройдено |
| 4. Искусственный интеллект (ai.py) |  |  |  |
| Алгоритм минимакс | ИИ делает ход | ИИ выбирает лучший ход | Пройдено/Не пройдено |
| Альфа-бета отсечение | ИИ рассматривает слишком много ходов | Отсечение сокращает количество рассматриваемых ходов | Пройдено/Не пройдено |
| Оценка состояния игры | Оценить позицию с разным количеством фигур | Оценка учитывает разницу в фигуры | Пройдено/Не пройдено |
| 5. Основной интерфейс (main.py) |  |  |  |
| Запуск программы | Запустить программу | Программа запускается корректно | Пройдено/Не пройдено |
| Выбор режима игры | Выбрать режим "Человек против ИИ" | Режим выбран, игра начинается | Пройдено/Не пройдено |
| Пауза и сохранение | Нажать кнопку "Пауза" и выйти | Игра сохранена, программа завершается | Пройдено/Не пройдено |
| 6. Настройки (settings.py) |  |  |  |
| Настройки цвета и размера | Запустить программу | Цвета и размеры доски соответствуют заданным | Пройдено/Не пройдено |
| Полноэкранный режим | Запустить программу | Программа работает в полноэкранном режиме | Пройдено/Не пройдено |

1.4.1 Описание отчета о тестировании

Отчет о тестировании предназначен для проверки корректности работы приложения и его соответствия требованиям технического задания.

* + 1. Цель тестирования

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

* Регистрацию и вход пользователей.
* Логику игры и ходы ИИ.
* Сохранение и загрузку игр.

1.4.3 Методика тестирования

Методика тестирования включает:

* Ручное тестирование интерфейса.
* Автоматизированное тестирование логики игры.
* Проверка сохранения и загрузки игр.

1.4.4 Проведенные тесты

1. Аутентификация и авторизация (auth.py)

Цель: Проверить корректность работы регистрации и входа в систему.

Ожидаемый результат:

* При регистрации с именем < 3 символов и паролем < 6 символов пользователь не регистрируется, выводится ошибка.
* При входе с неправильным паролем вход не выполняется, выводится ошибка.
* Пароль должен быть захеширован перед сохранением в базу данных.

2. База данных (database.py)

Цель: Проверить корректность работы базы данных.

Ожидаемый результат:

* При запуске программы база данных должна быть инициализирована с таблицами users и games.
* После завершения игры она должна быть сохранена в базе данных.
* При поиске пользователя по имени данные должны быть корректно получены.

3. Логика игры (game.py)

Цель: Проверить корректность работы логики игры.

Ожидаемый результат:

* При запуске новой игры начальная доска должна быть корректно инициализирована.
* Пешка не должна ходить назад или на занятую клетку, выводится ошибка.
* При установке позиции мата игра должна завершаться с сообщением "Мат".

4. Искусственный интеллект (ai.py)

Цель: Проверить корректность работы алгоритма ИИ.

Ожидаемый результат:

* ИИ должен выбирать лучший ход.
* Альфа-бета отсечение должно сокращать количество рассматриваемых ходов.
* Оценка состояния игры должна учитывать разницу в количестве фигур.

5. Основной интерфейс (main.py)

Цель: Проверить корректность работы основного интерфейса.

Ожидаемый результат:

* Программа должна запускаться корректно.
* При выборе режима "Человек против ИИ" игра должна начинаться.
* При нажатии кнопки "Пауза" игра должна быть сохранена, и программа должна завершиться.

6. Настройки (settings.py)

Цель: Проверить корректность работы настроек программы.

Ожидаемый результат:

* Цвета и размеры доски должны соответствовать заданным настройкам.
* Программа должна работать в полноэкранном режиме.

1.4.5 Выводы

Тестирование показало, что приложение работает корректно и соответствует требованиям технического задания.

Рекомендации по улучшению включают:

* Добавление дополнительных тестов для проверки граничных условий.
* Улучшение алгоритма оценки состояния игры для повышения качества игры.

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, пешка – Король, пешка»**

Р.02069337. №23/748-Вариант 17

Листов 20

Исполнитель:

студент гр. ИСТбд-23

Никольский Максим Сергеевич

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

2024

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение "Компьютерный эндшпиль: Король, пешка - Король, пешка" предназначено для реализации шахматного эндшпиля с использованием фигур "король" и "пешка". 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.
* Сохранение и загрузка игр: Пользователь может сохранять текущую игру и загружать ранее сохранённые игры.
* 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, macOS или Linux.
2. Интерпретатор Python: Версия 3.7 или выше.
3. Библиотеки Python:

* Pygame (для графического интерфейса).
* SQLite3 (для хранения данных).
* Bcrypt (для хеширования паролей).

1. Требования к оборудованию:

* Процессор: x86 или x64.
* ОЗУ: 2 ГБ или больше.
* Графика: Поддержка OpenGL или DirectX.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество строк кода: примерно 1500 строк.

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

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

Используемые библиотеки:

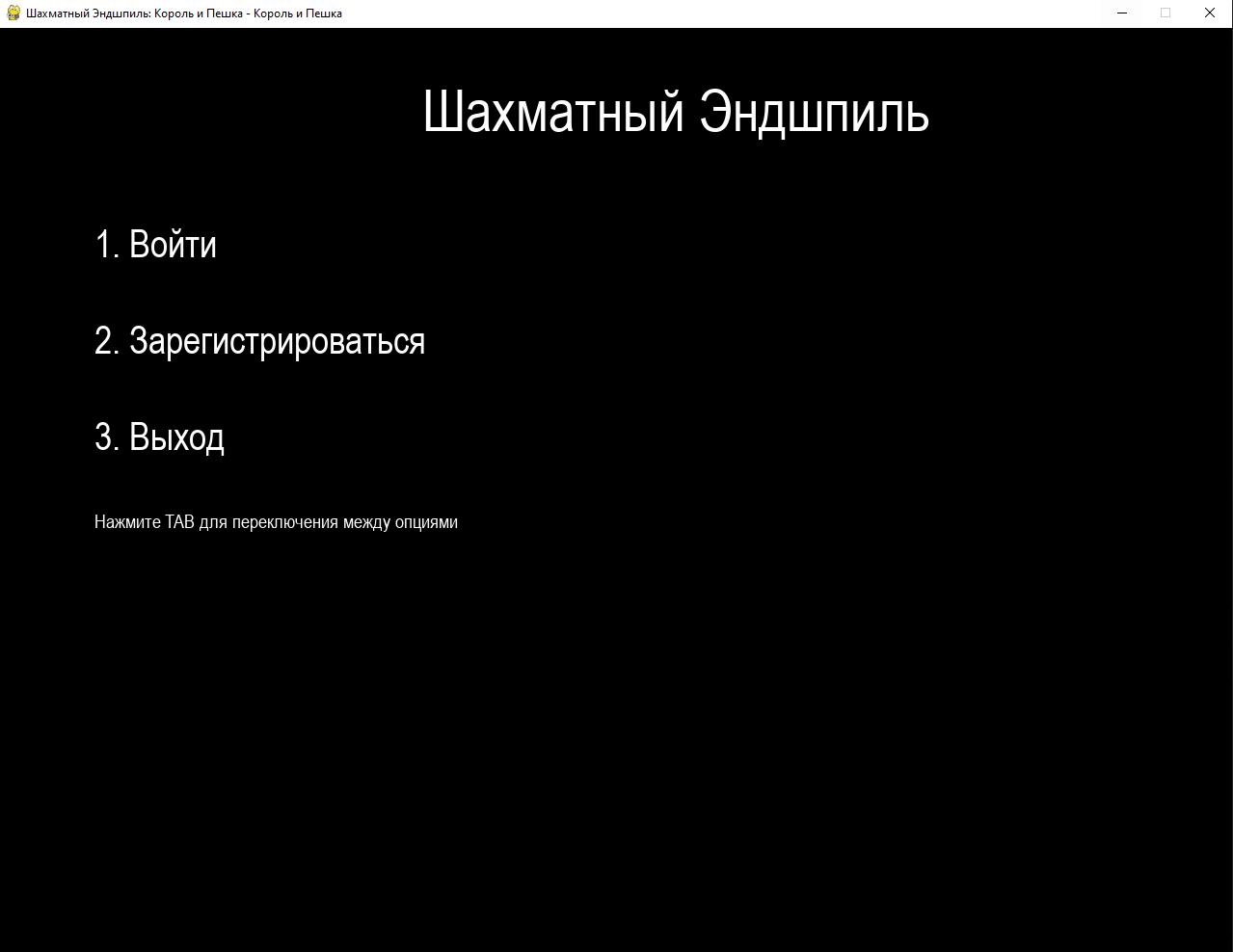
* Pygame: для создания графического интерфейса.
* SQLite3: для хранения данных пользователей и игр.
* Bcrypt: для хеширования паролей.
* Math: для математических операций.

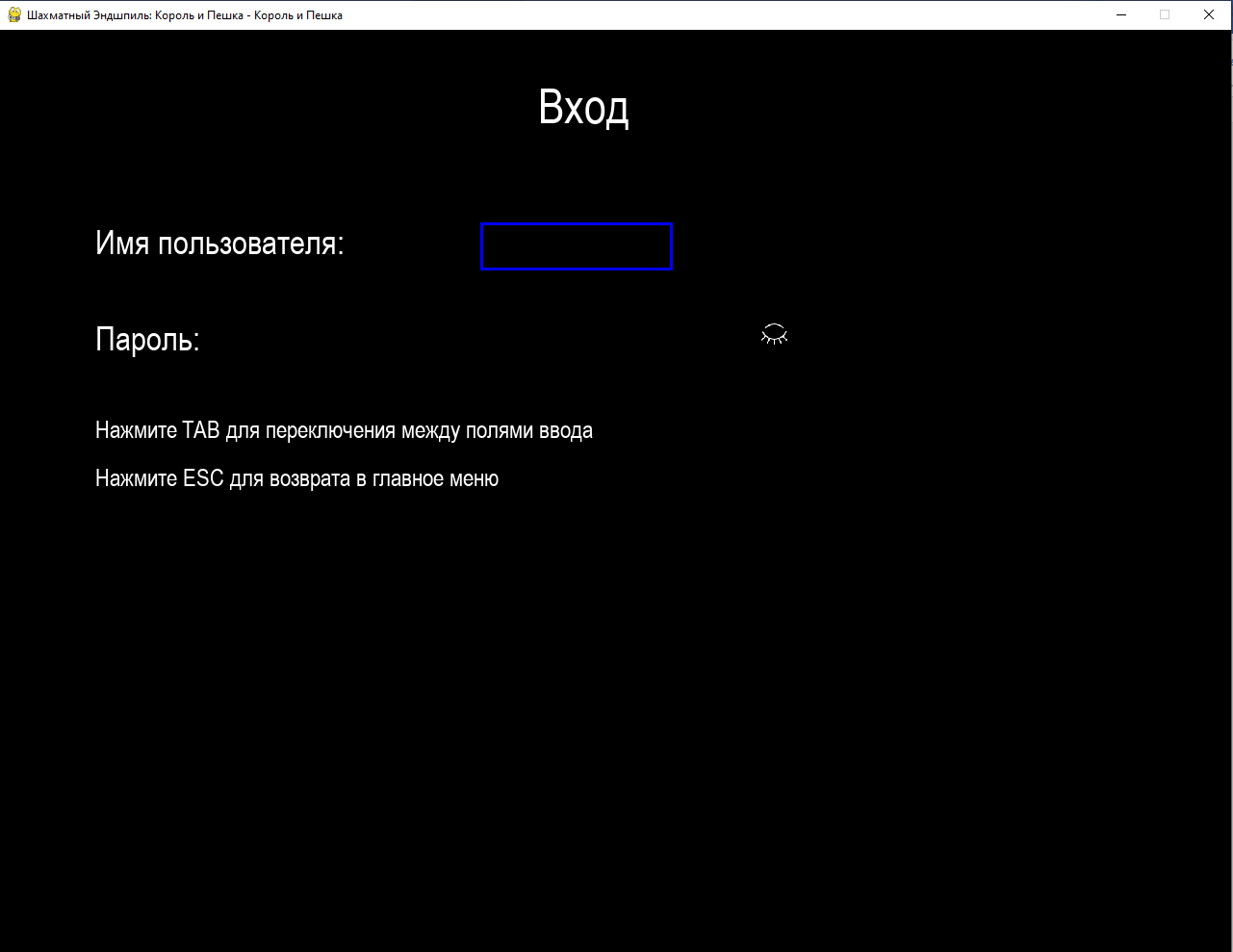
Работа приложения:

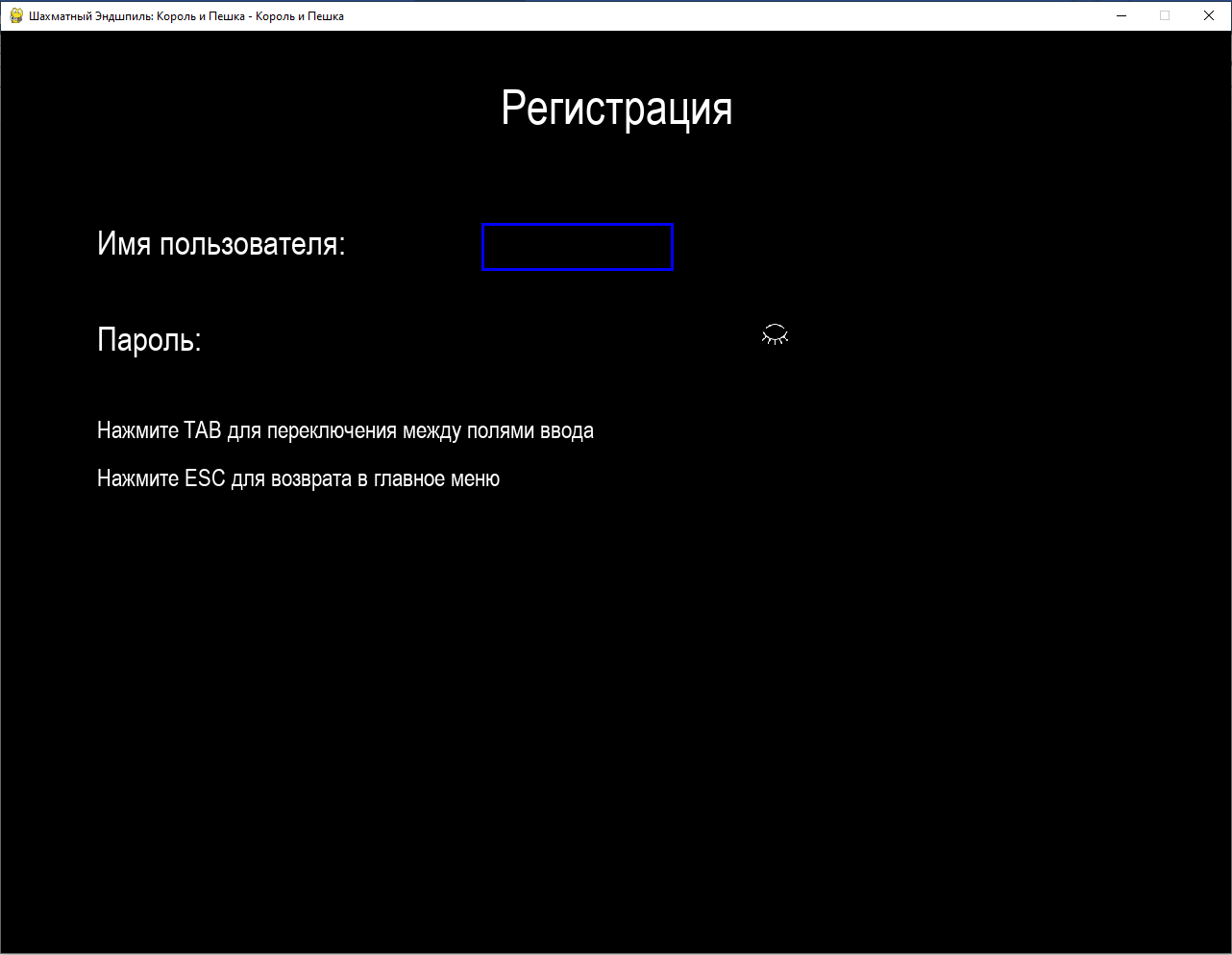
* Приложение запускается с помощью файла main.py.
* Пользователь может зарегистрироваться или войти в систему.
* После входа пользователь может начать новую игру или загрузить сохранённую игру.
* Во время игры пользователь взаимодействует с графическим интерфейсом, выбирая фигуры и делая ходы.
* ИИ анализирует ходы с помощью алгоритма минимакс и делает свои ходы.
* Игра может быть сохранена в любой момент.

Внешний вид приложения:

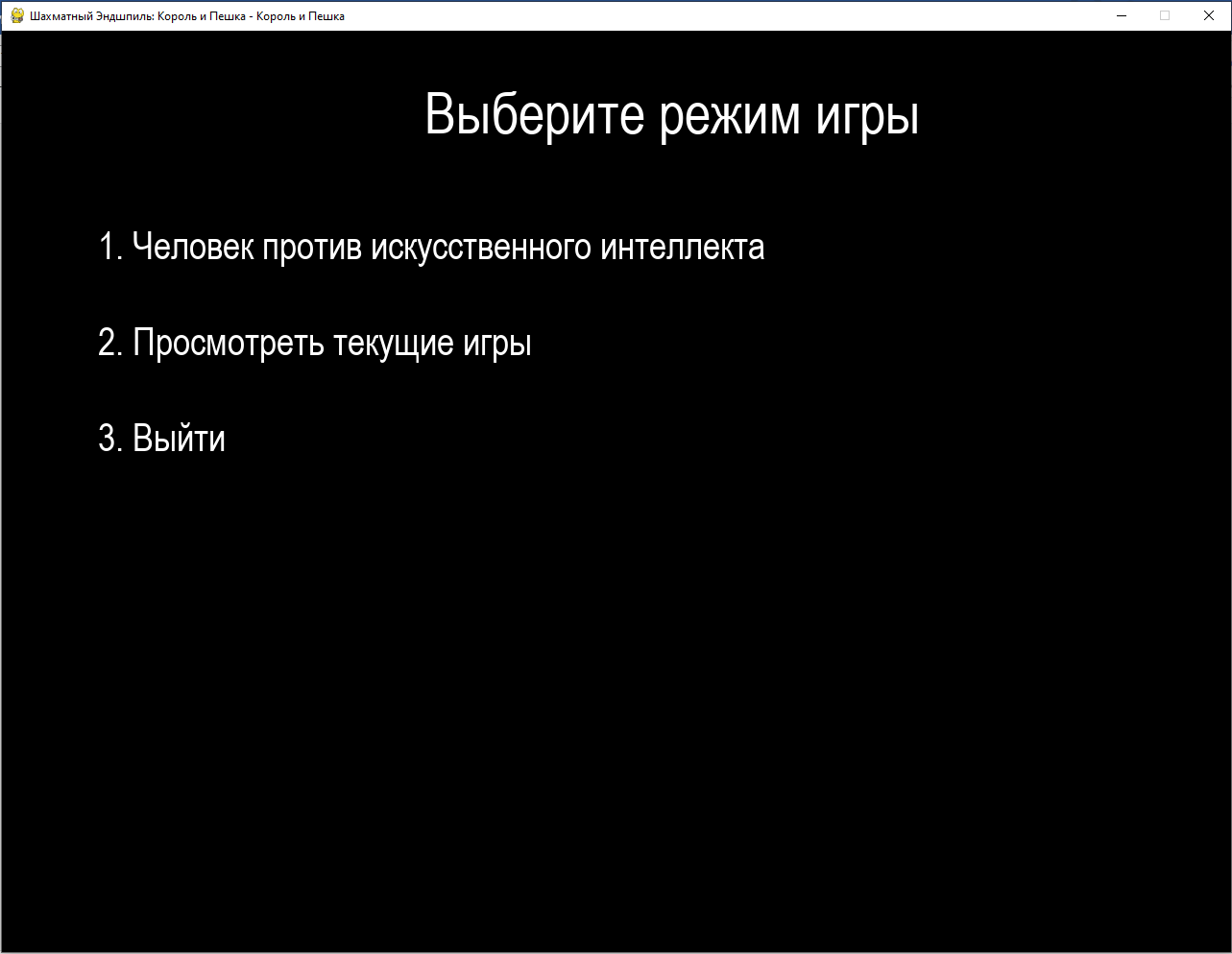
1. Главный экран: Экран входа/регистрации.

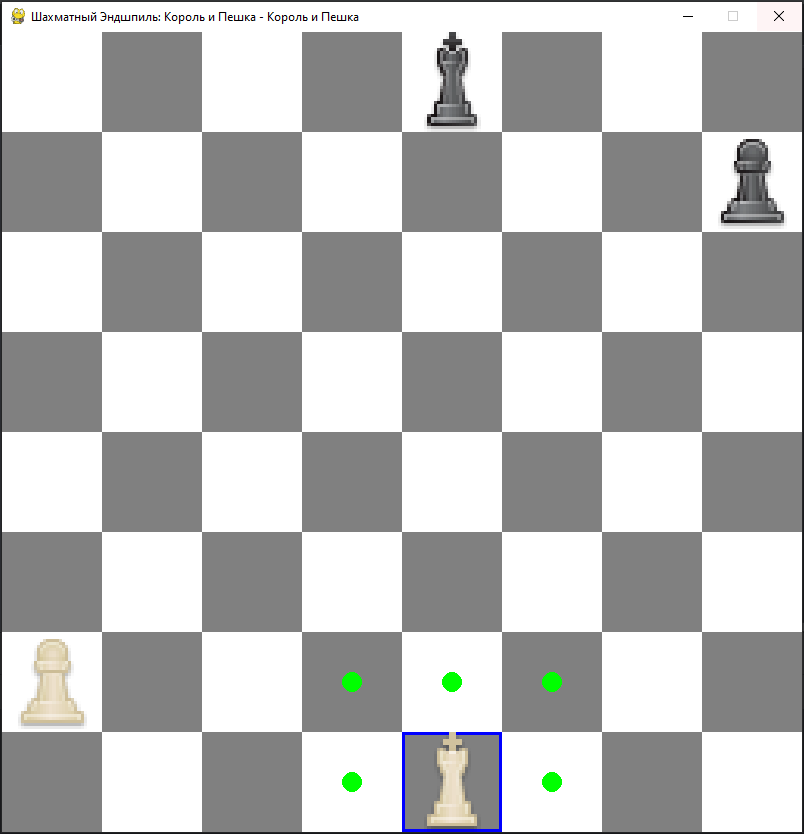




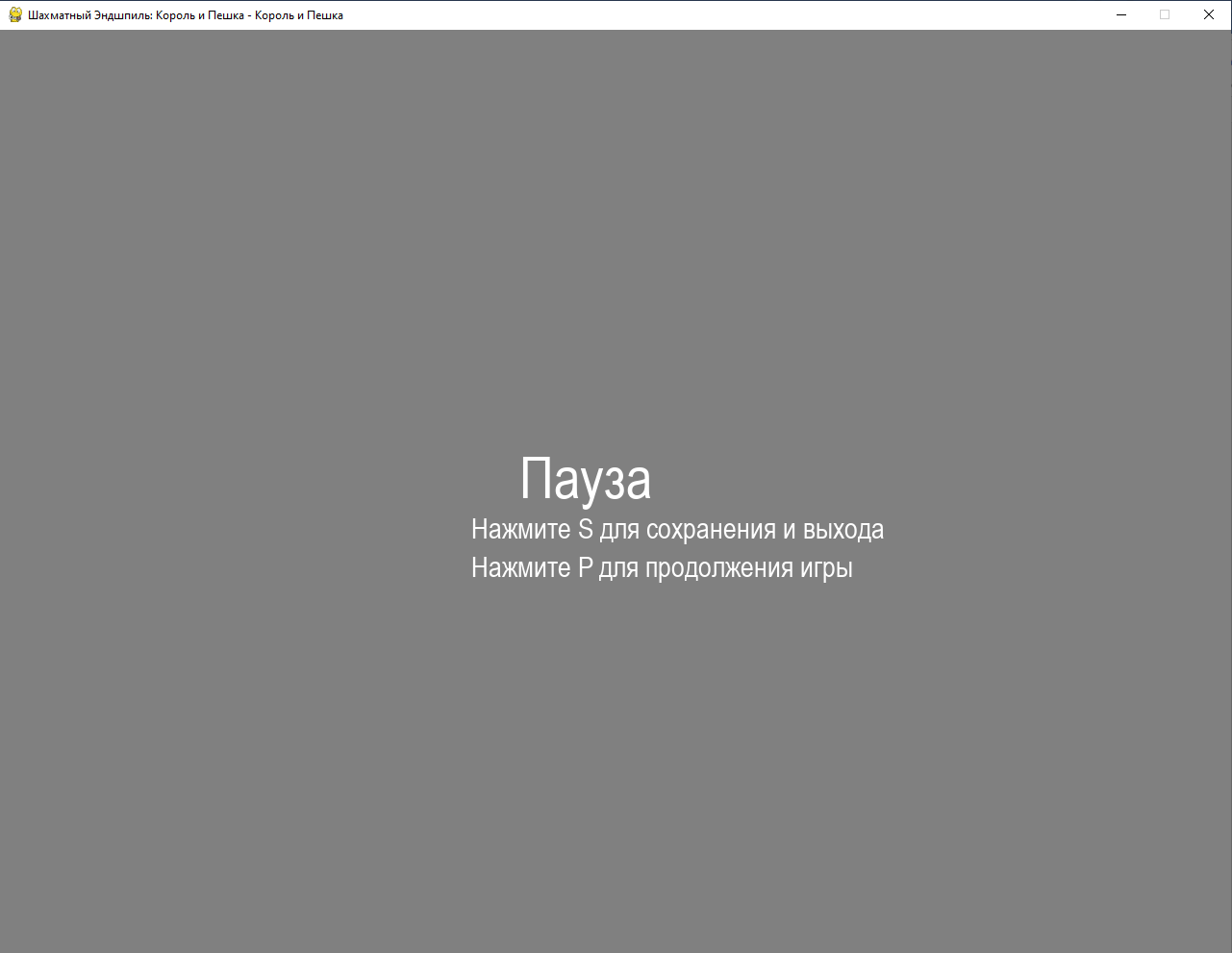


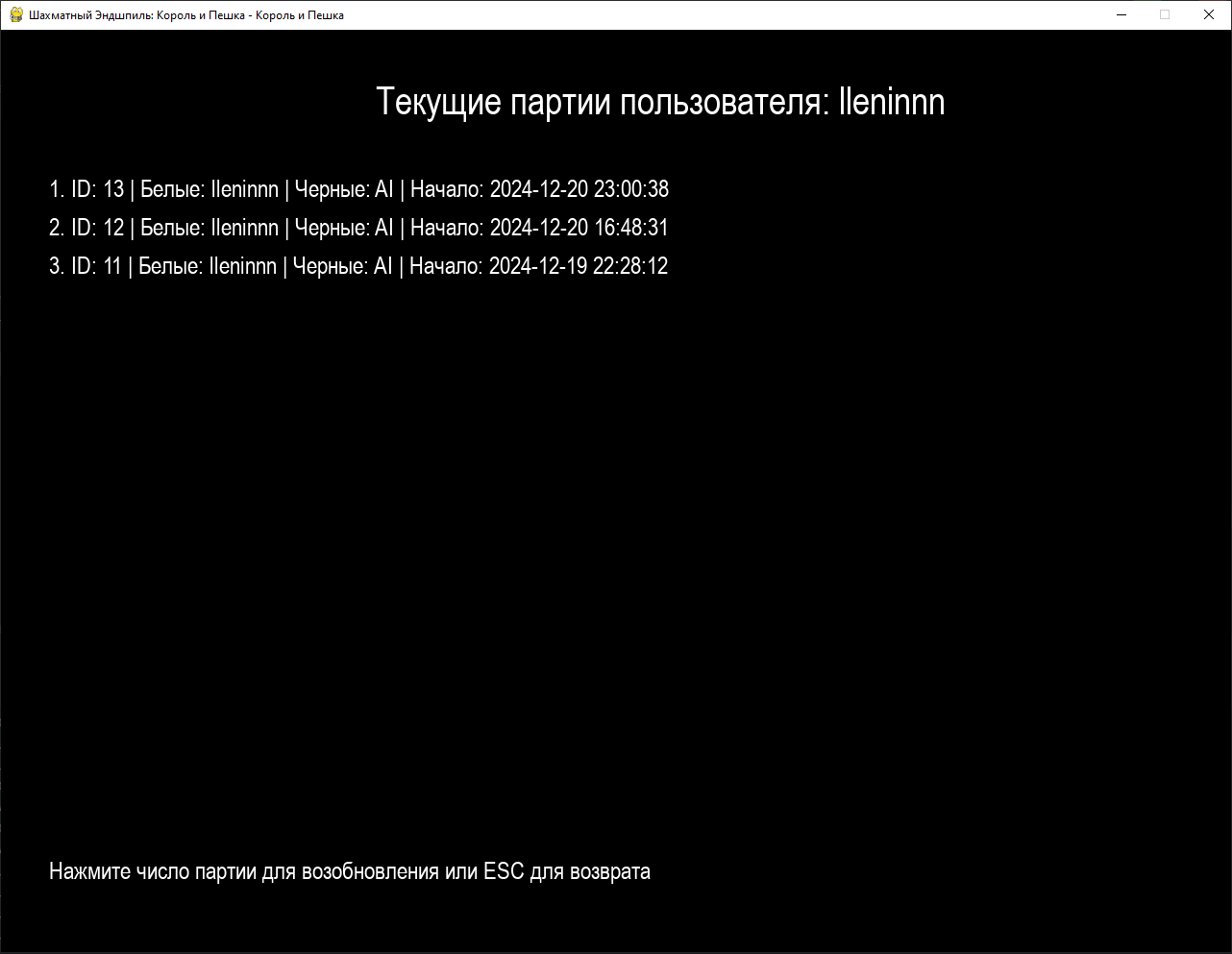
1. Экран игры: Графическое представление доски с фигурами





1. Экран сохранения игры: Выбор имени для сохранения игры





Средства контроля корректности ввода/вывода:

* Проверка корректности ввода пароля и имени пользователя при регистрации и входе.
* Проверка корректности ходов на доске (например, запрет ходов, которые ставят короля под шах).

**2.2 Особенности реализации приложения**

Структуры данных:

1. Класс Game:

* Представляет состояние игры, включая доску, ходы и логику.
* Используется для управления игрой и взаимодействия с базой данных.

1. Класс Move:

* Представляет ходы игроков, включая начальную и конечную позиции, фигуру и возможные превращения пешки.

1. User:

* Хранит информацию о пользователе (имя, пароль).

Альтернативные варианты:

* Вместо класса Game можно было использовать простую структуру данных, но класс позволяет инкапсулировать логику игры и упрощает управление состоянием.
* Вместо класса Move можно было использовать кортежи, но класс обеспечивает большую гибкость и читаемость кода.

**3. Обращение к программе**

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

1.1 register (username, password)

* Описание: Регистрация нового пользователя.
* Параметры:
* username (str): Имя пользователя.
* password (str): Пароль пользователя.
* Возвращаемое значение: Кортеж (success, message), где success — флаг успешности регистрации, а message — сообщение об ошибке или успешной регистрации.
* Алгоритм:
* Проверяется длина имени пользователя и пароля.
* Проверяется, существует ли уже пользователь с таким именем.
* Пароль хешируется с использованием библиотеки bcrypt.
* Пользователь сохраняется в базе данных.

1.2 login (username, password)

* Описание: Вход пользователя в систему.
* Параметры:
* username (str): Имя пользователя.
* password (str): Пароль пользователя.
* Возвращаемое значение: Кортеж (success, message, username), где success — флаг успешности входа, message — сообщение об ошибке или успешном входе, username — имя пользователя.
* Алгоритм:
* Проверяется, существует ли пользователь с указанным именем.
* Сравнивается хешированный пароль с введённым паролем.
* Возвращается результат входа.

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

2.1 initialize\_db ()

* Описание: Инициализация базы данных.
* Алгоритм:
* Создаются таблицы users и games, если они не существуют.
* Таблица users содержит столбцы username и password\_hash.
* Таблица games содержит столбцы game\_id, white\_player, black\_player, moves, result, start\_time, end\_time, status.

2.2 save\_user (username, password\_hash)

* Описание: Сохранение пользователя в базе данных.
* Параметры:
* username (str): Имя пользователя.
* password\_hash (str): Хешированный пароль.
* Возвращаемое значение: Кортеж (success, message), где success — флаг успешности сохранения, а message — сообщение об ошибке или успешном сохранении.
* Алгоритм:
* Вставляет данные пользователя в таблицу users.
  1. get\_user (username)
* Описание: Получение данных пользователя.
* Параметры:
* username (str): Имя пользователя.
* Возвращаемое значение: Кортеж с данными пользователя или None, если пользователь не найден.
* Алгоритм:
* Возвращает данные пользователя по его имени.
  1. create\_new\_game (white\_player, black\_player)
* Описание: Создание новой игры.
* Параметры:
* white\_player (str): Имя игрока за белых.
* black\_player (str): Имя игрока за чёрных.
* Возвращаемое значение: Идентификатор новой игры.
* Алгоритм:
* Вставляет новую игру в таблицу games.
  1. update\_game (game\_id, moves, result, end\_time, status)
* Описание: Обновление состояния игры.
* Параметры:
* game\_id (int): Идентификатор игры.
* moves (list): Список ходов.
* result (str): Результат игры.
* end\_time (str): Время окончания игры.
* status (str): Статус игры.
* Алгоритм:
* Обновляет данные игры в таблице games.
  1. get\_games\_by\_user (username, status)
* Описание: Получение списка игр пользователя.
* Параметры:
* username (str): Имя пользователя.
* status (str): Статус игры (например, in\_progress или completed).
* Возвращаемое значение: Список игр пользователя.
* Алгоритм:
* Возвращает список игр пользователя по его имени и статусу.
  1. get\_game\_by\_id(game\_id)
* Описание: Получение игры по её ID.
* Параметры:
* game\_id (int): Идентификатор игры.
* Возвращаемое значение: Данные игры или None, если игра не найдена.
* Алгоритм:
* Возвращает данные игры по её идентификатору.

**3. Методы и алгоритмы модуля game.py**

3.1 Game.\_\_init\_\_(self, white\_player, black\_player, game\_id)

* Описание: Инициализация игры.
* Параметры:
* white\_player (str): Имя игрока за белых.
* black\_player (str): Имя игрока за чёрных.
* game\_id (int): Идентификатор игры.
* Алгоритм:
* Инициализирует начальное состояние игры.
* Если указан game\_id, загружает игру из базы данных.
  1. Game.create\_initial\_board (self)
* Описание: Создание начальной доски.
* Возвращаемое значение: Начальное состояние доски.
* Алгоритм:
* Создаёт доску с начальными позициями фигур.
  1. Game.load\_game (self, game\_id)
* Описание: Загрузка игры из базы данных.
* Параметры:
* game\_id (int): Идентификатор игры.
* Алгоритм:
* Загружает данные игры из базы данных.
* Восстанавливает состояние доски и историю ходов.
  1. Game.reconstruct\_board (self)
* Описание: Восстановление доски из истории ходов.
* Алгоритм:
* Восстанавливает состояние доски на основе истории ходов.
  1. Game.make\_move (self, move, update\_state)
* Описание: Выполнение хода.
* Параметры:
* move (Move): Ход.
* update\_state (bool): Флаг, указывающий, нужно ли обновлять состояние игры.
* Алгоритм:
* Выполняет ход на доске.
* Добавляет ход в историю.
* Обновляет состояние игры, если указано.
  1. Game.undo\_move (self)
* Описание: Отмена последнего хода.
* Алгоритм:
* Отменяет последний ход на доске.
* Удаляет ход из истории.
* Обновляет состояние игры.
  1. Game.get\_valid\_moves (self)
* Описание: Получение всех допустимых ходов.
* Возвращаемое значение: Список допустимых ходов.
* Алгоритм:
* Генерирует все возможные ходы.
* Фильтрует ходы, чтобы исключить ходы, приводящие к шаху.
  1. Game.get\_all\_possible\_moves (self)
* Описание: Получение всех возможных ходов.
* Возвращаемое значение: Список возможных ходов.
* Алгоритм:
* Генерирует все возможные ходы для текущего игрока.
  1. Game.in\_check (self, white)
* Описание: Проверка, находится ли король под шахом.
* Параметры:
* white (bool): Флаг, указывающий, за какого игрока проверяется шах.
* Возвращаемое значение: True, если король под шахом, иначе False.
* Алгоритм:
* Находит позицию короля.
* Проверяет, атакуется ли король фигурами противника.
  1. Game.check\_game\_state (self)
* Описание: Проверка состояния игры (шах, мат, пат).
* Алгоритм:
* Проверяет, находится ли король под шахом.
* Если ходов нет, проверяет, является ли ситуация матом или патом.
  1. Game.save\_current\_game (self)
* Описание: Сохранение текущего состояния игры.
* Алгоритм:
* Сохраняет текущее состояние игры в базе данных.
  1. Game.save\_game\_completion (self)
* Описание: Сохранение завершённой игры.
* Алгоритм:
* Сохраняет завершённую игру в базе данных.
  1. Game.export\_pgn (self)
* Описание: Экспорт игры в формате PGN.
* Алгоритм:
* Создаёт файл PGN с историей ходов и результатом игры.
  1. Game.draw (self, win, images, selected\_square, valid\_moves)
* Описание: Отрисовка игры.
* Параметры:
* win (pygame.Surface): Поверхность для отрисовки.
* images (dict): Словарь с изображениями фигур.
* selected\_square (tuple): Выбранная клетка.
* valid\_moves (list): Список допустимых ходов.
* Алгоритм:
* Отрисовывает доску, фигуры и состояние игры.
  1. Game.draw\_board (self, win, selected\_square, valid\_moves)
* Описание: Отрисовка доски.
* Параметры:
* win (pygame.Surface): Поверхность для отрисовки.
* selected\_square (tuple): Выбранная клетка.
* valid\_moves (list): Список допустимых ходов.
* Алгоритм:
* Отрисовывает клетки доски.
* Подсвечивает выбранную клетку и допустимые ходы.
  1. Game.draw\_pieces (self, win, images)
* Описание: Отрисовка фигур.
* Параметры:
* win (pygame.Surface): Поверхность для отрисовки.
* images (dict): Словарь с изображениями фигур.
* Алгоритм:
* Отрисовывает фигуры на доске.
  1. Game.draw\_game\_state (self, win)
* Описание: Отрисовка состояния игры.
* Параметры:
* win (pygame.Surface): Поверхность для отрисовки.
* Алгоритм:
* Отрисовывает сообщения о шахе, мате или пате.
  1. Game.is\_move\_valid (self, move)
* Описание: Проверка, является ли ход допустимым.
* Параметры:
* move (Move): Ход.
* Возвращаемое значение: True, если ход допустим, иначе False.
* Алгоритм:
* Проверяет, содержится ли ход в списке допустимых ходов.
  1. Game.get\_piece\_moves (self, r, c)
* Описание: Получение всех ходов для фигуры на заданной позиции.
* Параметры:
* r (int): Строка доски.
* c (int): Столбец доски.
* Возвращаемое значение: Список ходов для фигуры.
* Алгоритм:
* Генерирует все возможные ходы для фигуры на заданной позиции.
* Фильтрует ходы, чтобы исключить ходы, приводящие к шаху.

**4. Методы и алгоритмы модуля ai.py**

4.1 find\_best\_move (game, depth)

* Описание: Поиск наилучшего хода для искусственного интеллекта.
* Параметры:
* game (Game): Текущее состояние игры.
* depth (int): Глубина поиска.
* Возвращаемое значение: Наилучший ход.
* Алгоритм:
* Использует алгоритм Minimax с альфа-бета отсечением для поиска наилучшего хода.
  1. minimax (game, depth, alpha, beta, is\_maximizing)
* Описание: Рекурсивный алгоритм Minimax с альфа-бета отсечением.
* Параметры:
* game (Game): Текущее состояние игры.
* depth (int): Глубина поиска.
* alpha (float): Значение альфа.
* beta (float): Значение бета.
* is\_maximizing (bool): Флаг, указывающий, максимизируется или минимизируется оценка.
* Возвращаемое значение: Оценка состояния игры.
* Алгоритм:
* Если достигнута максимальная глубина или игра завершена, возвращает оценку игры.
* Генерирует все возможные ходы.
* Для каждого хода вызывает рекурсивно minimax.
* Применяет альфа-бета отсечение.
  1. evaluate\_game (game)
* Описание: Оценка текущего состояния игры.
* Параметры:
* game (Game): Текущее состояние игры.
* Возвращаемое значение: Оценка игры.
* Алгоритм:
* Вычисляет разницу в количестве фигур на доске.

**5. Методы и алгоритмы модуля main.py**

5.1 auth\_screen ()

* Описание: Экран аутентификации.
* Возвращаемое значение: Имя пользователя.
* Алгоритм:
* Отображает экран с выбором регистрации или входа.
* Вызывает login\_prompt или register\_prompt.

5.2 login\_prompt ()

* Описание: Ввод данных для входа.
* Возвращаемое значение: Кортеж (success, message, username).
* Алгоритм:
* Отображает экран входа.
* Вызывает функцию login для проверки данных.

5.3 register\_prompt ()

* Описание: Ввод данных для регистрации.
* Возвращаемое значение: Кортеж (success, message).
* Алгоритм:
* Отображает экран регистрации.
* Вызывает функцию register для сохранения данных.

5.4 select\_mode (username)

* Описание: Выбор режима игры.
* Параметры:
* username (str): Имя пользователя.
* Возвращаемое значение: Режим игры (ai или view\_games).
* Алгоритм:
* Отображает экран с выбором режима игры.

5.5 view\_games (username)

* Описание: Просмотр списка игр.
* Параметры:
* username (str): Имя пользователя.
* Алгоритм:
* Отображает список игр пользователя.
* Позволяет возобновить игру.

5.6 resume\_game (game)

* Описание: Возобновление игры.
* Параметры:
* game (tuple): Данные игры.
* Алгоритм:
* Загружает игру из базы данных.
* Отображает экран игры.

5.7 game\_screen (mode, white\_player, black\_player)

* Описание: Экран игры.
* Параметры:
* mode (str): Режим игры (ai или view\_games).
* white\_player (str): Имя игрока за белых.
* black\_player (str): Имя игрока за чёрных.
* Алгоритм:
* Создаёт новую игру.
* Отображает экран игры.

5.8 main ()

* Описание: Главная функция, запускающая приложение.
* Алгоритм:
* Инициализирует базу данных.
* Отображает экран аутентификации.
* Вызывает функцию select\_mode для выбора режима игры.

**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.draw.circle (): Отрисовка кругов.
* 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 sqlite3

* Описание: Библиотека для работы с базой данных SQLite.
* Функции:
* sqlite3.connect(): Подключение к базе данных.
* cursor.execute(): Выполнение SQL-запросов.
* conn.commit(): Сохранение изменений в базе данных.
* cursor.fetchone(): Получение одной строки из результата запроса.
* cursor.fetchall(): Получение всех строк из результата запроса.
* Применение: Используется для хранения пользователей и игр.

6.3 bcrypt

* Описание: Библиотека для хеширования паролей.
* Функции:
* bcrypt.hashpw(): Хеширование пароля.
* bcrypt.checkpw(): Проверка пароля.
* Применение: Используется для безопасного хранения паролей пользователей.

6.4 math

* Описание: Стандартная библиотека Python для математических операций.
* Функции:
* math.inf: Бесконечность.
* Применение: Используется в алгоритме Minimax для установки начальных значений.

6.5 json

* Описание: Стандартная библиотека Python для работы с JSON.
* Функции:
* json.dumps(): Сериализация данных в JSON.
* json.loads(): Десериализация данных из JSON.
* Применение: Используется для хранения и загрузки ходов в базе данных.

6.6 os

* Описание: Стандартная библиотека Python для работы с операционной системой.
* Функции:
* os.path.join(): Объединение путей.
* Применение: Используется для работы с путями к файлам.

**4. Сообщения**

1. Сообщение об ошибке входа:

* Текст: "Неверный пароль."
* Условие: Неправильный пароль при входе.

2. Сообщение об ошибке регистрации:

* Текст: "Пользователь уже существует."
* Условие: Попытка регистрации с уже существующим именем пользователя.

3. Сообщение об успешной регистрации:

* Текст: "Регистрация успешна. Можете войти."
* Условие: Успешная регистрация нового пользователя.

4. Сообщение о шахе:

* Текст: "Шах!"
* Условие: Король игрока под шахом.

5. Сообщение о мате:

* Текст: "Шах и мат!"
* Условие: Король игрока под матом.

6. Сообщение о пате:

* Текст: "Пат!"
* Условие: Игрок не может сделать ход, но король не под шахом.

# **Листинг программы**

# ai.py

**import** math

**from** game **import** Move

# Функция для поиска лучшего хода для текущей игры

**def** find\_best\_move**(**game**,** depth**):**

"""

Функция ищет лучший ход для текущей игры с использованием алгоритма минимакс с альфа-бета отсечениями.

:param game: Экземпляр класса Game, представляющий текущую игру.

:param depth: Глубина поиска в дереве ходов.

:return: Лучший ход для текущей игры.

"""

best\_move **=** **None**

**if** game**.***white\_to\_move***:**

# Если ход белых, ищем максимальное значение

best\_value **=** **-**math**.***inf*

**for** move **in** game**.***get\_valid\_moves***():**

game**.***make\_move***(**move**,** update\_state**=False)**

move\_value **=** minimax**(**game**,** depth **-** 1**,** **-**math**.***inf***,** math**.***inf***,** **False)**

game**.***undo\_move***()**

**if** move\_value **>** best\_value**:**

best\_value **=** move\_value

best\_move **=** move

**else:**

# Если ход чёрных, ищем минимальное значение

best\_value **=** math**.***inf*

**for** move **in** game**.***get\_valid\_moves***():**

game**.***make\_move***(**move**,** update\_state**=False)**

move\_value **=** minimax**(**game**,** depth **-** 1**,** **-**math**.***inf***,** math**.***inf***,** **True)**

game**.***undo\_move***()**

**if** move\_value **<** best\_value**:**

best\_value **=** move\_value

best\_move **=** move

**return** best\_move

# Алгоритм минимакс с альфа-бета отсечениями

**def** minimax**(**game**,** depth**,** alpha**,** beta**,** is\_maximizing**):**

"""

Алгоритм минимакс с альфа-бета отсечениями для оценки лучшего хода.

:param game: Экземпляр класса Game, представляющий текущую игру.

:param depth: Глубина поиска в дереве ходов.

:param alpha: Лучшая (максимальная) оценка для максимизирующего игрока.

:param beta: Лучшая (минимальная) оценка для минимизирующего игрока.

:param is\_maximizing: True, если текущий игрок максимизирует оценку, иначе False.

:return: Оценка текущего состояния игры.

"""

# Если достигнута максимальная глубина или игра завершена, возвращаем оценку

**if** depth **==** 0 **or** game**.***checkmate* **or** game**.***stalemate***:**

**return** evaluate\_game**(**game**)**

**if** is\_maximizing**:**

max\_eval **=** **-**math**.***inf*

**for** move **in** game**.***get\_valid\_moves***():**

game**.***make\_move***(**move**,** update\_state**=False)**

**eval** **=** minimax**(**game**,** depth **-** 1**,** alpha**,** beta**,** **False)**

game**.***undo\_move***()**

max\_eval **=** **max(**max\_eval**,** **eval)**

alpha **=** **max(**alpha**,** **eval)**

# Если beta меньше или равно alpha, отсекаем ветвь

**if** beta **<=** alpha**:**

**break**

**return** max\_eval

**else:**

min\_eval **=** math**.***inf*

**for** move **in** game**.***get\_valid\_moves***():**

game**.***make\_move***(**move**,** update\_state**=False)**

**eval** **=** minimax**(**game**,** depth **-** 1**,** alpha**,** beta**,** **True)**

game**.***undo\_move***()**

min\_eval **=** **min(**min\_eval**,** **eval)**

beta **=** **min(**beta**,** **eval)**

# Если beta меньше или равно alpha, отсекаем ветвь

**if** beta **<=** alpha**:**

**break**

**return** min\_eval

# Функция для оценки текущего состояния игры

**def** evaluate\_game**(**game**):**

"""

Оценивает текущее состояние игры на основе разницы в количестве фигур.

:param game: Экземпляр класса Game, представляющий текущую игру.

:return: Оценка текущего состояния игры.

"""

# Простая оценка: разница в количестве фигур

piece\_values **=** **{**'K'**:** 1000**,** 'Q'**:** 9**,** 'R'**:** 5**,** 'B'**:** 3**,** 'N'**:** 3**,** 'P'**:** 1**}**

white\_score **=** 0

black\_score **=** 0

**for** row **in** game**.***board***:**

**for** piece **in** row**:**

**if** piece **!=** '--'**:**

value **=** piece\_values**.***get***(**piece**[**1**],** 0**)**

**if** piece**[**0**]** **==** 'w'**:**

white\_score **+=** value

**else:**

black\_score **+=** value

**return** white\_score **-** black\_score

# auth.py

**import** bcrypt

**from** database **import** save\_user**,** get\_user

# Функция для регистрации нового пользователя

**def** register**(**username**,** password**):**

"""

Регистрирует нового пользователя в системе.

:param username: Имя пользователя.

:param password: Пароль пользователя.

:return: Кортеж (успех, сообщение), где успех - True, если регистрация прошла успешно, иначе False.

"""

# Проверка длины имени пользователя и пароля

**if** **len(**username**)** **<** 3 **or** **len(**password**)** **<** 6**:**

**return** **False,** 'Имя пользователя должно быть не менее 3 символов, а пароль - не менее 6.'

# Проверка, существует ли уже пользователь

user **=** get\_user**(**username**)**

**if** user**:**

**return** **False,** 'Пользователь уже существует.'

# Хеширование пароля

hashed **=** bcrypt**.***hashpw***(**password**.***encode***(**'utf-8'**),** bcrypt**.***gensalt***()).***decode***(**'utf-8'**)**

# Сохранение пользователя в базе данных

success**,** msg **=** save\_user**(**username**,** hashed**)**

**return** success**,** msg

# Функция для входа пользователя в систему

**def** login**(**username**,** password**):**

"""

Вход пользователя в систему.

:param username: Имя пользователя.

:param password: Пароль пользователя.

:return: Кортеж (успех, сообщение, имя пользователя), где успех - True, если вход прошёл успешно, иначе False.

"""

user **=** get\_user**(**username**)**

**if** **not** user**:**

**return** **False,** 'Пользователь не найден.'**,** **None**

stored\_hash **=** user**[**1**].***encode***(**'utf-8'**)** # Предполагается, что второй столбец - password\_hash

**if** bcrypt**.***checkpw***(**password**.***encode***(**'utf-8'**),** stored\_hash**):**

**return** **True,** 'Вход успешен.'**,** username

**else:**

**return** **False,** 'Неверный пароль.'**,** **None**

# database.py

**import** sqlite3

**from** datetime **import** datetime

**import** json

DATABASE\_FILE **=** 'chess.db'

# Функция для получения соединения с базой данных

**def** get\_connection**():**

"""

Возвращает соединение с базой данных.

:return: Объект соединения с базой данных.

"""

conn **=** sqlite3**.***connect***(**DATABASE\_FILE**)**

**return** conn

# Функция для инициализации базы данных

**def** initialize\_db**():**

"""

Инициализирует базу данных, создавая таблицы пользователей и игр, если они ещё не существуют.

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

# Создание таблицы пользователей (если еще не существует)

cursor**.***execute***(**'''

CREATE TABLE IF NOT EXISTS users (

username TEXT PRIMARY KEY,

password\_hash TEXT NOT NULL

)

'''**)**

# Создание таблицы партий

cursor**.***execute***(**'''

CREATE TABLE IF NOT EXISTS games (

game\_id INTEGER PRIMARY KEY AUTOINCREMENT,

white\_player TEXT NOT NULL,

black\_player TEXT NOT NULL,

moves TEXT NOT NULL, -- Сохранение ходов в формате JSON

result TEXT,

start\_time TEXT,

end\_time TEXT,

status TEXT NOT NULL, -- 'in\_progress' или 'completed'

FOREIGN KEY (white\_player) REFERENCES users(username),

FOREIGN KEY (black\_player) REFERENCES users(username)

)

'''**)**

conn**.***commit***()**

conn**.***close***()**

# Функция для сохранения пользователя в базе данных

**def** save\_user**(**username**,** password\_hash**):**

"""

Сохраняет нового пользователя в базе данных.

:param username: Имя пользователя.

:param password\_hash: Хеш пароля пользователя.

:return: Кортеж (успех, сообщение), где успех - True, если сохранение прошло успешно, иначе False.

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

**try:**

cursor**.***execute***(**'INSERT INTO users (username, password\_hash) VALUES (?, ?)'**,** **(**username**,** password\_hash**))**

conn**.***commit***()**

**return** **True,** 'Пользователь успешно зарегистрирован.'

**except** sqlite3**.***IntegrityError***:**

**return** **False,** 'Пользователь уже существует.'

**finally:**

conn**.***close***()**

# Функция для получения пользователя из базы данных

**def** get\_user**(**username**):**

"""

Получает пользователя из базы данных по имени пользователя.

:param username: Имя пользователя.

:return: Кортеж с данными пользователя или None, если пользователь не найден.

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

cursor**.***execute***(**'SELECT \* FROM users WHERE username = ?'**,** **(**username**,))**

user **=** cursor**.***fetchone***()**

conn**.***close***()**

**return** user

# Функция для создания новой игры

**def** create\_new\_game**(**white\_player**,** black\_player**):**

"""

Создаёт новую игру в базе данных.

:param white\_player: Имя пользователя, играющего за белых.

:param black\_player: Имя пользователя, играющего за чёрных.

:return: ID новой игры.

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

start\_time **=** datetime**.***now***().***strftime***(**'%Y-%m-%d %H:%M:%S'**)**

moves\_json **=** json**.***dumps***([])**

cursor**.***execute***(**'''

INSERT INTO games (white\_player, black\_player, moves, result, start\_time, end\_time, status)

VALUES (?, ?, ?, ?, ?, ?, ?)

'''**,** **(**white\_player**,** black\_player**,** moves\_json**,** **None,** start\_time**,** **None,** 'in\_progress'**))**

game\_id **=** cursor**.***lastrowid*

conn**.***commit***()**

conn**.***close***()**

**return** game\_id

# Функция для обновления игры в базе данных

**def** update\_game**(**game\_id**,** moves**,** result**=None,** end\_time**=None,** status**=**'in\_progress'**):**

"""

Обновляет информацию о игре в базе данных.

:param game\_id: ID игры.

:param moves: Список ходов в формате JSON.

:param result: Результат игры (например, 'White wins', 'Black wins', 'Draw').

:param end\_time: Время окончания игры.

:param status: Статус игры ('in\_progress' или 'completed').

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

moves\_json **=** json**.***dumps***(**moves**)**

cursor**.***execute***(**'''

UPDATE games

SET moves = ?, result = ?, end\_time = ?, status = ?

WHERE game\_id = ?

'''**,** **(**moves\_json**,** result**,** end\_time**,** status**,** game\_id**))**

conn**.***commit***()**

conn**.***close***()**

# Функция для получения списка игр пользователя

**def** get\_games\_by\_user**(**username**,** status**=None):**

"""

Получает список игр пользователя по его имени.

:param username: Имя пользователя.

:param status: Статус игры ('in\_progress' или 'completed'). Если None, возвращает все игры.

:return: Список игр пользователя.

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

**if** status**:**

cursor**.***execute***(**'''

SELECT \* FROM games

WHERE (white\_player = ? OR black\_player = ?) AND status = ?

ORDER BY start\_time DESC

'''**,** **(**username**,** username**,** status**))**

**else:**

cursor**.***execute***(**'''

SELECT \* FROM games

WHERE white\_player = ? OR black\_player = ?

ORDER BY start\_time DESC

'''**,** **(**username**,** username**))**

games **=** cursor**.***fetchall***()**

conn**.***close***()**

**return** games

# Функция для получения игры по её ID

**def** get\_game\_by\_id**(**game\_id**):**

"""

Получает игру по её ID.

:param game\_id: ID игры.

:return: Кортеж с данными игры или None, если игра не найдена.

"""

conn **=** get\_connection**()**

cursor **=** conn**.***cursor***()**

cursor**.***execute***(**'SELECT \* FROM games WHERE game\_id = ?'**,** **(**game\_id**,))**

game **=** cursor**.***fetchone***()**

conn**.***close***()**

**return** game

# game.py

**import** pygame

**import** settings # Исправлено: импортируем как модуль

**from** settings **import** **\***

**from** copy **import** deepcopy

**from** datetime **import** datetime

**from** database **import** update\_game**,** get\_game\_by\_id

**import** json # Добавлен импорт json

# Класс, представляющий ход в игре

**class** **Move:**

**def** \_\_init\_\_**(**self**,** start\_pos**,** end\_pos**,** piece\_moved**,** piece\_captured**,** is\_pawn\_promotion**=False,** promotion\_choice**=**'Q'**):**

"""

Инициализирует объект хода.

:param start\_pos: Начальная позиция хода (строка, столбец).

:param end\_pos: Конечная позиция хода (строка, столбец).

:param piece\_moved: Фигура, которая была передвинута.

:param piece\_captured: Фигура, которая была взята (если есть).

:param is\_pawn\_promotion: True, если это ход пешки на последнюю линию и она превращается.

:param promotion\_choice: Выбранная фигура для превращения пешки (по умолчанию 'Q' - ферзь).

"""

self**.***start\_row***,** self**.***start\_col* **=** start\_pos

self**.***end\_row***,** self**.***end\_col* **=** end\_pos

self**.***piece\_moved* **=** piece\_moved

self**.***piece\_captured* **=** piece\_captured

self**.***is\_pawn\_promotion* **=** is\_pawn\_promotion

self**.***promotion\_choice* **=** promotion\_choice

**def** \_\_eq\_\_**(**self**,** other**):**

"""

Проверяет, равны ли два объекта Move.

:param other: Другой объект Move.

:return: True, если объекты равны, иначе False.

"""

**if** **isinstance(**other**,** Move**):**

**return** **(**self**.***start\_row* **==** other**.***start\_row* **and**

self**.***start\_col* **==** other**.***start\_col* **and**

self**.***end\_row* **==** other**.***end\_row* **and**

self**.***end\_col* **==** other**.***end\_col* **and**

self**.***piece\_moved* **==** other**.***piece\_moved* **and**

self**.***piece\_captured* **==** other**.***piece\_captured* **and**

self**.***is\_pawn\_promotion* **==** other**.***is\_pawn\_promotion* **and**

self**.***promotion\_choice* **==** other**.***promotion\_choice***)**

**return** **False**

**def** get\_chess\_notation**(**self**):**

"""

Возвращает шахматную нотацию хода.

:return: Строка с шахматной нотацией хода.

"""

cols\_to\_files **=** **{**0**:** 'a'**,** 1**:** 'b'**,** 2**:** 'c'**,** 3**:** 'd'**,**

4**:** 'e'**,** 5**:** 'f'**,** 6**:** 'g'**,** 7**:** 'h'**}**

**return** cols\_to\_files**[**self**.***start\_col***]** **+** **str(**8 **-** self**.***start\_row***)** **+** \

cols\_to\_files**[**self**.***end\_col***]** **+** **str(**8 **-** self**.***end\_row***)**

**def** to\_dict**(**self**):**

"""

Преобразует объект Move в словарь.

:return: Словарь с данными хода.

"""

**return** **{**

'start\_pos'**:** **[**self**.***start\_row***,** self**.***start\_col***],**

'end\_pos'**:** **[**self**.***end\_row***,** self**.***end\_col***],**

'piece\_moved'**:** self**.***piece\_moved***,**

'piece\_captured'**:** self**.***piece\_captured***,**

'is\_pawn\_promotion'**:** self**.***is\_pawn\_promotion***,**

'promotion\_choice'**:** self**.***promotion\_choice*

**}**

*@classmethod*

**def** from\_dict**(**cls**,** move\_dict**):**

"""

Создаёт объект Move из словаря.

:param move\_dict: Словарь с данными хода.

:return: Объект Move.

"""

**return** cls**(**

start\_pos**=tuple(**move\_dict**[**'start\_pos'**]),**

end\_pos**=tuple(**move\_dict**[**'end\_pos'**]),**

piece\_moved**=**move\_dict**[**'piece\_moved'**],**

piece\_captured**=**move\_dict**[**'piece\_captured'**],**

is\_pawn\_promotion**=**move\_dict**[**'is\_pawn\_promotion'**],**

promotion\_choice**=**move\_dict**.***get***(**'promotion\_choice'**,** 'Q'**)**

**)**

# Класс, представляющий игру

**class** **Game:**

**def** \_\_init\_\_**(**self**,** white\_player**=**'White'**,** black\_player**=**'AI'**,** game\_id**=None):**

"""

Инициализирует объект игры.

:param white\_player: Имя пользователя, играющего за белых.

:param black\_player: Имя пользователя, играющего за чёрных.

:param game\_id: ID игры (если игра загружается из базы данных).

"""

self**.***white\_player* **=** white\_player

self**.***black\_player* **=** black\_player

self**.***game\_id* **=** game\_id

**if** game\_id**:**

self**.***load\_game***(**game\_id**)**

**else:**

self**.***board* **=** self**.***create\_initial\_board***()**

self**.***white\_to\_move* **=** **True**

self**.***move\_log* **=** **[]**

self**.***selected\_square* **=** **None**

self**.***valid\_moves* **=** **[]**

self**.***checkmate* **=** **False**

self**.***stalemate* **=** **False**

self**.***en\_passant\_possible* **=** **()**

self**.***promotion\_choice* **=** 'Q'

self**.***start\_time* **=** datetime**.***now***().***strftime***(**'%Y-%m-%d %H:%M:%S'**)**

self**.***end\_time* **=** **None**

self**.***result* **=** **None**

**def** create\_initial\_board**(**self**):**

"""

Создаёт начальную доску для игры.

:return: Двумерный список, представляющий доску.

"""

board **=** **[[**'--' **for** \_ **in** **range(**8**)]** **for** \_ **in** **range(**8**)]**

board**[**7**][**4**]** **=** 'wK' # Белый король на e1

board**[**6**][**0**]** **=** 'wP' # Белая пешка на a2

board**[**0**][**4**]** **=** 'bK' # Чёрный король на e8

board**[**1**][**7**]** **=** 'bP' # Чёрная пешка на h7

**return** board

**def** load\_game**(**self**,** game\_id**):**

"""

Загружает игру из базы данных по её ID.

:param game\_id: ID игры.

"""

game **=** get\_game\_by\_id**(**game\_id**)**

**if** game**:**

\_**,** white\_player**,** black\_player**,** moves**,** result**,** start\_time**,** end\_time**,** status **=** game

self**.***white\_player* **=** white\_player

self**.***black\_player* **=** black\_player

self**.***move\_log* **=** **[**Move**.***from\_dict***(**move\_dict**)** **for** move\_dict **in** json**.***loads***(**moves**)]**

self**.***reconstruct\_board***()**

self**.***result* **=** result

self**.***start\_time* **=** start\_time

self**.***end\_time* **=** end\_time

self**.***checkmate* **=** status **==** 'completed' **and** **(**'checkmate' **in** **(**result**.***lower***())** **if** result **else** **False)**

self**.***stalemate* **=** status **==** 'completed' **and** **(**'stalemate' **in** **(**result**.***lower***())** **if** result **else** **False)**

self**.***white\_to\_move* **=** **len(**self**.***move\_log***)** **%** 2 **==** 0

**else:**

**print(**f"Игра с ID {game\_id} не найдена."**)**

self**.***board* **=** self**.***create\_initial\_board***()**

self**.***white\_to\_move* **=** **True**

self**.***move\_log* **=** **[]**

self**.***selected\_square* **=** **None**

self**.***valid\_moves* **=** **[]**

self**.***checkmate* **=** **False**

self**.***stalemate* **=** **False**

self**.***en\_passant\_possible* **=** **()**

self**.***promotion\_choice* **=** 'Q'

self**.***start\_time* **=** datetime**.***now***().***strftime***(**'%Y-%m-%d %H:%M:%S'**)**

self**.***end\_time* **=** **None**

self**.***result* **=** **None**

**def** reconstruct\_board**(**self**):**

"""

Восстанавливает доску на основе списка ходов.

"""

self**.***board* **=** self**.***create\_initial\_board***()**

**for** move **in** self**.***move\_log***:**

self**.***board***[**move**.***start\_row***][**move**.***start\_col***]** **=** '--'

self**.***board***[**move**.***end\_row***][**move**.***end\_col***]** **=** move**.***piece\_moved*

**if** move**.***is\_pawn\_promotion***:**

self**.***board***[**move**.***end\_row***][**move**.***end\_col***]** **=** move**.***piece\_moved***[**0**]** **+** move**.***promotion\_choice*

**def** make\_move**(**self**,** move**,** update\_state**=True):**

"""

Выполняет ход на доске.

:param move: Объект Move, представляющий ход.

:param update\_state: True, если нужно обновлять состояние игры (например, проверять шах и мат).

"""

self**.***board***[**move**.***start\_row***][**move**.***start\_col***]** **=** '--'

self**.***board***[**move**.***end\_row***][**move**.***end\_col***]** **=** move**.***piece\_moved*

self**.***move\_log***.***append***(**move**)**

self**.***white\_to\_move* **=** **not** self**.***white\_to\_move*

**if** move**.***is\_pawn\_promotion***:**

self**.***board***[**move**.***end\_row***][**move**.***end\_col***]** **=** move**.***piece\_moved***[**0**]** **+** move**.***promotion\_choice*

**if** update\_state**:**

self**.***check\_game\_state***()**

self**.***save\_current\_game***()**

**def** undo\_move**(**self**):**

"""

Отменяет последний ход.

"""

**if** self**.***move\_log***:**

move **=** self**.***move\_log***.***pop***()**

self**.***board***[**move**.***start\_row***][**move**.***start\_col***]** **=** move**.***piece\_moved*

self**.***board***[**move**.***end\_row***][**move**.***end\_col***]** **=** move**.***piece\_captured*

self**.***white\_to\_move* **=** **not** self**.***white\_to\_move*

self**.***check\_game\_state***()**

self**.***save\_current\_game***()**

**def** get\_valid\_moves**(**self**):**

"""

Возвращает список допустимых ходов для текущего игрока.

:return: Список объектов Move.

"""

moves **=** self**.***get\_all\_possible\_moves***()**

valid\_moves **=** **[]**

**for** move **in** moves**:**

game\_copy **=** deepcopy**(**self**)**

game\_copy**.***make\_move***(**move**,** update\_state**=False)**

**if** **not** game\_copy**.***in\_check***(not** self**.***white\_to\_move***):**

valid\_moves**.***append***(**move**)**

**return** valid\_moves

**def** get\_all\_possible\_moves**(**self**):**

"""

Возвращает список всех возможных ходов для текущего игрока.

:return: Список объектов Move.

"""

moves **=** **[]**

**for** r **in** **range(**8**):**

**for** c **in** **range(**8**):**

piece **=** self**.***board***[**r**][**c**]**

**if** piece **==** '--'**:**

**continue**

**if** self**.***white\_to\_move* **and** piece**[**0**]** **!=** 'w'**:**

**continue**

**if** **not** self**.***white\_to\_move* **and** piece**[**0**]** **!=** 'b'**:**

**continue**

piece\_type **=** piece**[**1**]**

**if** piece\_type **==** 'K'**:**

self**.***get\_king\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'P'**:**

self**.***get\_pawn\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'Q'**:**

self**.***get\_queen\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'R'**:**

self**.***get\_rook\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'B'**:**

self**.***get\_bishop\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'N'**:**

self**.***get\_knight\_moves***(**r**,** c**,** moves**)**

**return** moves

**def** get\_king\_moves**(**self**,** r**,** c**,** moves**):**

"""

Добавляет возможные ходы короля в список ходов.

:param r: Строка, где находится король.

:param c: Столбец, где находится король.

:param moves: Список ходов.

"""

directions **=** **[(-**1**,** **-**1**),** **(-**1**,** 0**),** **(-**1**,** 1**),**

**(**0**,** **-**1**),** **(**0**,** 1**),**

**(**1**,** **-**1**),** **(**1**,** 0**),** **(**1**,** 1**)]**

ally\_color **=** 'w' **if** self**.***white\_to\_move* **else** 'b'

**for** dr**,** dc **in** directions**:**

end\_row**,** end\_col **=** r **+** dr**,** c **+** dc

**if** 0 **<=** end\_row **<** 8 **and** 0 **<=** end\_col **<** 8**:**

target **=** self**.***board***[**end\_row**][**end\_col**]**

**if** target **==** '--' **or** target**[**0**]** **!=** ally\_color**:**

moves**.***append***(**Move**((**r**,** c**),** **(**end\_row**,** end\_col**),** self**.***board***[**r**][**c**],** target**))**

**def** get\_pawn\_moves**(**self**,** r**,** c**,** moves**):**

"""

Добавляет возможные ходы пешки в список ходов.

:param r: Строка, где находится пешка.

:param c: Столбец, где находится пешка.

:param moves: Список ходов.

"""

direction **=** **-**1 **if** self**.***white\_to\_move* **else** 1

start\_row **=** 6 **if** self**.***white\_to\_move* **else** 1

enemy\_color **=** 'b' **if** self**.***white\_to\_move* **else** 'w'

# Пешка движется вперёд на 1 клетку

**if** self**.***board***[**r **+** direction**][**c**]** **==** '--'**:**

moves**.***append***(**Move**((**r**,** c**),** **(**r **+** direction**,** c**),** self**.***board***[**r**][**c**],** '--'**))**

# Пешка может пойти на 2 клетки из начальной позиции

**if** r **==** start\_row **and** self**.***board***[**r **+** 2 **\*** direction**][**c**]** **==** '--'**:**

moves**.***append***(**Move**((**r**,** c**),** **(**r **+** 2 **\*** direction**,** c**),** self**.***board***[**r**][**c**],** '--'**))**

# Пешка бьёт по диагонали, но \*\*НЕ\*\* атакует короля

**for** dc **in** **[-**1**,** 1**]:**

**if** 0 **<=** c **+** dc **<** 8 **and** 0 **<=** r **+** direction **<** 8**:**

piece\_at\_target **=** self**.***board***[**r **+** direction**][**c **+** dc**]**

**if** piece\_at\_target **!=** '--' **and** piece\_at\_target**[**0**]** **==** enemy\_color **and** piece\_at\_target**[**1**]** **!=** 'K'**:** # НЕ бьёт короля

moves**.***append***(**Move**((**r**,** c**),** **(**r **+** direction**,** c **+** dc**),** self**.***board***[**r**][**c**],** piece\_at\_target**))**

**def** get\_queen\_moves**(**self**,** r**,** c**,** moves**):**

"""

Добавляет возможные ходы ферзя в список ходов.

:param r: Строка, где находится ферзь.

:param c: Столбец, где находится ферзь.

:param moves: Список ходов.

"""

self**.***get\_rook\_moves***(**r**,** c**,** moves**)**

self**.***get\_bishop\_moves***(**r**,** c**,** moves**)**

**def** get\_rook\_moves**(**self**,** r**,** c**,** moves**):**

"""

Добавляет возможные ходы ладьи в список ходов.

:param r: Строка, где находится ладья.

:param c: Столбец, где находится ладья.

:param moves: Список ходов.

"""

directions **=** **[(-**1**,** 0**),** **(**1**,** 0**),** **(**0**,** **-**1**),** **(**0**,** 1**)]**

ally\_color **=** 'w' **if** self**.***white\_to\_move* **else** 'b'

**for** dr**,** dc **in** directions**:**

end\_row**,** end\_col **=** r **+** dr**,** c **+** dc

**while** 0 **<=** end\_row **<** 8 **and** 0 **<=** end\_col **<** 8**:**

target **=** self**.***board***[**end\_row**][**end\_col**]**

**if** target **==** '--'**:**

moves**.***append***(**Move**((**r**,** c**),** **(**end\_row**,** end\_col**),** self**.***board***[**r**][**c**],** target**))**

**else:**

**if** target**[**0**]** **!=** ally\_color**:**

moves**.***append***(**Move**((**r**,** c**),** **(**end\_row**,** end\_col**),** self**.***board***[**r**][**c**],** target**))**

**break**

end\_row **+=** dr

end\_col **+=** dc

**def** get\_bishop\_moves**(**self**,** r**,** c**,** moves**):**

"""

Добавляет возможные ходы слона в список ходов.

:param r: Строка, где находится слон.

:param c: Столбец, где находится слон.

:param moves: Список ходов.

"""

directions **=** **[(-**1**,** **-**1**),** **(-**1**,** 1**),** **(**1**,** **-**1**),** **(**1**,** 1**)]**

ally\_color **=** 'w' **if** self**.***white\_to\_move* **else** 'b'

**for** dr**,** dc **in** directions**:**

end\_row**,** end\_col **=** r **+** dr**,** c **+** dc

**while** 0 **<=** end\_row **<** 8 **and** 0 **<=** end\_col **<** 8**:**

target **=** self**.***board***[**end\_row**][**end\_col**]**

**if** target **==** '--'**:**

moves**.***append***(**Move**((**r**,** c**),** **(**end\_row**,** end\_col**),** self**.***board***[**r**][**c**],** target**))**

**else:**

**if** target**[**0**]** **!=** ally\_color**:**

moves**.***append***(**Move**((**r**,** c**),** **(**end\_row**,** end\_col**),** self**.***board***[**r**][**c**],** target**))**

**break**

end\_row **+=** dr

end\_col **+=** dc

**def** get\_knight\_moves**(**self**,** r**,** c**,** moves**):**

"""

Добавляет возможные ходы коня в список ходов.

:param r: Строка, где находится конь.

:param c: Столбец, где находится конь.

:param moves: Список ходов.

"""

knight\_moves **=** **[(-**2**,** **-**1**),** **(-**1**,** **-**2**),** **(-**2**,** 1**),** **(-**1**,** 2**),**

**(**1**,** **-**2**),** **(**2**,** **-**1**),** **(**1**,** 2**),** **(**2**,** 1**)]**

ally\_color **=** 'w' **if** self**.***white\_to\_move* **else** 'b'

**for** dr**,** dc **in** knight\_moves**:**

end\_row**,** end\_col **=** r **+** dr**,** c **+** dc

**if** 0 **<=** end\_row **<** 8 **and** 0 **<=** end\_col **<** 8**:**

target **=** self**.***board***[**end\_row**][**end\_col**]**

**if** target **==** '--' **or** target**[**0**]** **!=** ally\_color**:**

moves**.***append***(**Move**((**r**,** c**),** **(**end\_row**,** end\_col**),** self**.***board***[**r**][**c**],** target**))**

**def** in\_check**(**self**,** white**):**

"""

Проверяет, находится ли король под шахом.

:param white: True, если проверяем для белых, иначе False.

:return: True, если король под шахом, иначе False.

"""

king\_pos **=** **None**

ally\_color **=** 'w' **if** white **else** 'b'

enemy\_color **=** 'b' **if** white **else** 'w'

**for** r **in** **range(**8**):**

**for** c **in** **range(**8**):**

piece **=** self**.***board***[**r**][**c**]**

**if** piece **==** '--'**:**

**continue**

**if** piece**[**0**]** **==** ally\_color **and** piece**[**1**]** **==** 'K'**:**

king\_pos **=** **(**r**,** c**)**

**break**

**if** king\_pos**:**

**break**

**if** **not** king\_pos**:**

**return** **False**

**for** r **in** **range(**8**):**

**for** c **in** **range(**8**):**

piece **=** self**.***board***[**r**][**c**]**

**if** piece **==** '--' **or** piece**[**0**]** **!=** enemy\_color**:**

**continue**

piece\_type **=** piece**[**1**]**

**if** piece\_type **==** 'K'**:**

**if** **max(abs(**king\_pos**[**0**]** **-** r**),** **abs(**king\_pos**[**1**]** **-** c**))** **==** 1**:**

**return** **True**

**elif** piece\_type **==** 'P'**:**

**if** self**.***is\_square\_attacked\_by\_pawn***(**king\_pos**,** **(**r**,** c**)):**

**return** **True**

**elif** piece\_type **==** 'N'**:**

**if** self**.***is\_square\_attacked\_by\_knight***(**king\_pos**,** **(**r**,** c**)):**

**return** **True**

**elif** piece\_type **==** 'B'**:**

**if** self**.***is\_square\_attacked\_by\_bishop***(**king\_pos**,** **(**r**,** c**)):**

**return** **True**

**elif** piece\_type **==** 'R'**:**

**if** self**.***is\_square\_attacked\_by\_rook***(**king\_pos**,** **(**r**,** c**)):**

**return** **True**

**elif** piece\_type **==** 'Q'**:**

**if** self**.***is\_square\_attacked\_by\_queen***(**king\_pos**,** **(**r**,** c**)):**

**return** **True**

**return** **False**

**def** is\_square\_attacked\_by\_pawn**(**self**,** king\_pos**,** attacker\_pos**):**

"""

Проверяет, атакует ли пешка данную клетку.

:param king\_pos: Позиция клетки, которую проверяем.

:param attacker\_pos: Позиция пешки.

:return: True, если пешка атакует клетку, иначе False.

"""

r**,** c **=** attacker\_pos

king\_r**,** king\_c **=** king\_pos

piece **=** self**.***board***[**r**][**c**]**

direction **=** 1 **if** piece**[**0**]** **==** 'w' **else** **-**1

**return** **(**king\_r **==** r **+** direction **and** **abs(**king\_c **-** c**)** **==** 1**)**

**def** is\_square\_attacked\_by\_knight**(**self**,** king\_pos**,** attacker\_pos**):**

"""

Проверяет, атакует ли конь данную клетку.

:param king\_pos: Позиция клетки, которую проверяем.

:param attacker\_pos: Позиция коня.

:return: True, если конь атакует клетку, иначе False.

"""

knight\_moves **=** **[(-**2**,** **-**1**),** **(-**1**,** **-**2**),** **(-**2**,** 1**),** **(-**1**,** 2**),**

**(**1**,** **-**2**),** **(**2**,** **-**1**),** **(**1**,** 2**),** **(**2**,** 1**)]**

r**,** c **=** attacker\_pos

king\_r**,** king\_c **=** king\_pos

**for** dr**,** dc **in** knight\_moves**:**

**if** **(**r **+** dr**,** c **+** dc**)** **==** king\_pos**:**

**return** **True**

**return** **False**

**def** is\_square\_attacked\_by\_bishop**(**self**,** king\_pos**,** attacker\_pos**):**

"""

Проверяет, атакует ли слон данную клетку.

:param king\_pos: Позиция клетки, которую проверяем.

:param attacker\_pos: Позиция слона.

:return: True, если слон атакует клетку, иначе False.

"""

directions **=** **[(-**1**,** **-**1**),** **(-**1**,** 1**),** **(**1**,** **-**1**),** **(**1**,** 1**)]**

**return** self**.***is\_square\_attacked\_along\_directions***(**king\_pos**,** attacker\_pos**,** directions**)**

**def** is\_square\_attacked\_by\_rook**(**self**,** king\_pos**,** attacker\_pos**):**

"""

Проверяет, атакует ли ладья данную клетку.

:param king\_pos: Позиция клетки, которую проверяем.

:param attacker\_pos: Позиция ладьи.

:return: True, если ладья атакует клетку, иначе False.

"""

directions **=** **[(-**1**,** 0**),** **(**1**,** 0**),** **(**0**,** **-**1**),** **(**0**,** 1**)]**

**return** self**.***is\_square\_attacked\_along\_directions***(**king\_pos**,** attacker\_pos**,** directions**)**

**def** is\_square\_attacked\_by\_queen**(**self**,** king\_pos**,** attacker\_pos**):**

"""

Проверяет, атакует ли ферзь данную клетку.

:param king\_pos: Позиция клетки, которую проверяем.

:param attacker\_pos: Позиция ферзя.

:return: True, если ферзь атакует клетку, иначе False.

"""

directions **=** **[(-**1**,** **-**1**),** **(-**1**,** 0**),** **(-**1**,** 1**),**

**(**0**,** **-**1**),** **(**0**,** 1**),**

**(**1**,** **-**1**),** **(**1**,** 0**),** **(**1**,** 1**)]**

**return** self**.***is\_square\_attacked\_along\_directions***(**king\_pos**,** attacker\_pos**,** directions**)**

**def** is\_square\_attacked\_along\_directions**(**self**,** king\_pos**,** attacker\_pos**,** directions**):**

"""

Проверяет, атакует ли фигура данную клетку вдоль заданных направлений.

:param king\_pos: Позиция клетки, которую проверяем.

:param attacker\_pos: Позиция фигуры.

:param directions: Список направлений.

:return: True, если фигура атакует клетку, иначе False.

"""

r**,** c **=** attacker\_pos

king\_r**,** king\_c **=** king\_pos

**for** dr**,** dc **in** directions**:**

end\_row**,** end\_col **=** r **+** dr**,** c **+** dc

**while** 0 **<=** end\_row **<** 8 **and** 0 **<=** end\_col **<** 8**:**

target **=** self**.***board***[**end\_row**][**end\_col**]**

**if** **(**end\_row**,** end\_col**)** **==** king\_pos**:**

**return** **True**

**if** target **!=** '--'**:**

**break**

end\_row **+=** dr

end\_col **+=** dc

**return** **False**

**def** check\_game\_state**(**self**):**

"""

Проверяет текущее состояние игры (шах, мат, пат).

"""

**if** self**.***in\_check***(**self**.***white\_to\_move***):**

**if** **not** self**.***get\_valid\_moves***():**

self**.***checkmate* **=** **True**

self**.***stalemate* **=** **False**

self**.***result* **=** 'Black wins by checkmate' **if** self**.***white\_to\_move* **else** 'White wins by checkmate'

self**.***end\_time* **=** datetime**.***now***().***strftime***(**'%Y-%m-%d %H:%M:%S'**)**

self**.***update\_game\_status***(**'completed'**)**

**else:**

self**.***checkmate* **=** **False**

self**.***stalemate* **=** **False**

**else:**

**if** **not** self**.***get\_valid\_moves***():**

self**.***stalemate* **=** **True**

self**.***checkmate* **=** **False**

self**.***result* **=** 'Draw by stalemate'

self**.***end\_time* **=** datetime**.***now***().***strftime***(**'%Y-%m-%d %H:%M:%S'**)**

self**.***update\_game\_status***(**'completed'**)**

**else:**

self**.***stalemate* **=** **False**

self**.***checkmate* **=** **False**

**def** update\_game\_status**(**self**,** status**):**

"""

Обновляет статус игры в базе данных.

:param status: Новый статус игры ('in\_progress' или 'completed').

"""

**if** self**.***game\_id***:**

update\_game**(**

game\_id**=**self**.***game\_id***,**

moves**=[**move**.***to\_dict***()** **for** move **in** self**.***move\_log***],**

result**=**self**.***result***,**

end\_time**=**self**.***end\_time***,**

status**=**status

**)**

**def** save\_current\_game**(**self**):**

"""

Сохраняет текущее состояние игры в базе данных.

"""

**if** self**.***game\_id***:**

update\_game**(**

game\_id**=**self**.***game\_id***,**

moves**=[**move**.***to\_dict***()** **for** move **in** self**.***move\_log***],**

result**=**self**.***result***,**

end\_time**=**self**.***end\_time***,**

status**=**'completed' **if** self**.***result* **else** 'in\_progress'

**)**

**def** save\_game\_completion**(**self**):**

"""

Сохраняет завершённую игру в базе данных.

"""

**if** self**.***game\_id* **and** self**.***result***:**

update\_game**(**

game\_id**=**self**.***game\_id***,**

moves**=[**move**.***to\_dict***()** **for** move **in** self**.***move\_log***],**

result**=**self**.***result***,**

end\_time**=**self**.***end\_time***,**

status**=**'completed'

**)**

# Экспорт в PGN

self**.***export\_pgn***()**

**def** export\_pgn**(**self**):**

"""

Экспортирует игру в формат PGN.

"""

**if** **not** self**.***game\_id***:**

**return**

pgn\_content **=** f"[Event \"Chess Endgame\"]\n"

pgn\_content **+=** f"[Site \"Local\"]\n"

pgn\_content **+=** f"[Date \"{self**.***start\_time***.***split***(**' '**)[**0**]**}\"]\n"

pgn\_content **+=** f"[Round \"-\"]\n"

pgn\_content **+=** f"[White \"{self**.***white\_player*}\"]\n"

pgn\_content **+=** f"[Black \"{self**.***black\_player*}\"]\n"

pgn\_content **+=** f"[Result \"{self**.***result*}\"]\n\n"

move\_text **=** ''

**for** i **in** **range(**0**,** **len(**self**.***move\_log***),** 2**):**

move\_number **=** i**//**2 **+** 1

white\_move **=** self**.***move\_log***[**i**].***get\_chess\_notation***()**

black\_move **=** self**.***move\_log***[**i**+**1**].***get\_chess\_notation***()** **if** i**+**1 **<** **len(**self**.***move\_log***)** **else** ''

move\_text **+=** f"{move\_number}. {white\_move} {black\_move} "

move\_text **+=** self**.***result*

pgn\_content **+=** move\_text

# Сохранение в файл PGN

pgn\_filename **=** f"game\_{self**.***start\_time***.***replace***(**':'**,** '-'**).***replace***(**' '**,** '\_'**)**}\_id\_{self**.***game\_id*}.pgn"

**with** **open(**pgn\_filename**,** 'w'**)** **as** f**:**

f**.***write***(**pgn\_content**)**

**def** draw**(**self**,** win**,** images**,** selected\_square**=None,** valid\_moves**=None):**

"""

Отрисовывает текущее состояние игры.

:param win: Объект окна Pygame.

:param images: Словарь с изображениями фигур.

:param selected\_square: Выбранная клетка (если есть).

:param valid\_moves: Список допустимых ходов для выбранной фигуры.

"""

self**.***draw\_board***(**win**,** selected\_square**,** valid\_moves**)**

self**.***draw\_pieces***(**win**,** images**)**

self**.***draw\_game\_state***(**win**)**

**def** draw\_board**(**self**,** win**,** selected\_square**,** valid\_moves**):**

"""

Отрисовывает доску.

:param win: Объект окна Pygame.

:param selected\_square: Выбранная клетка (если есть).

:param valid\_moves: Список допустимых ходов для выбранной фигуры.

"""

colors **=** **[**WHITE**,** GRAY**]**

**for** r **in** **range(**8**):**

**for** c **in** **range(**8**):**

color **=** colors**[(**r **+** c**)** **%** 2**]**

pygame**.***draw***.***rect***(**win**,** color**,** pygame**.***Rect***(**c **\*** settings**.***CELL\_SIZE***,** r **\*** settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***))**

**if** selected\_square **and** **(**r**,** c**)** **==** selected\_square**:**

pygame**.***draw***.***rect***(**win**,** BLUE**,** pygame**.***Rect***(**c **\*** settings**.***CELL\_SIZE***,** r **\*** settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***),** 3**)**

**if** valid\_moves**:**

**for** move **in** valid\_moves**:**

**if** move**.***end\_row* **==** r **and** move**.***end\_col* **==** c**:**

center **=** **(**c **\*** settings**.***CELL\_SIZE* **+** settings**.***CELL\_SIZE***//**2**,** r **\*** settings**.***CELL\_SIZE* **+** settings**.***CELL\_SIZE***//**2**)**

pygame**.***draw***.***circle***(**win**,** GREEN**,** center**,** 10**)**

**def** draw\_pieces**(**self**,** win**,** images**):**

"""

Отрисовывает фигуры на доске.

:param win: Объект окна Pygame.

:param images: Словарь с изображениями фигур.

"""

**for** r **in** **range(**8**):**

**for** c **in** **range(**8**):**

piece **=** self**.***board***[**r**][**c**]**

**if** piece **!=** '--'**:**

img **=** images**.***get***(**piece**)**

**if** img**:**

win**.***blit***(**img**,** pygame**.***Rect***(**c **\*** settings**.***CELL\_SIZE***,** r **\*** settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***))**

**else:**

**print(**f"Изображение для {piece} не найдено."**)**

**def** draw\_game\_state**(**self**,** win**):**

"""

Отрисовывает состояние игры (шах, мат, пат).

:param win: Объект окна Pygame.

"""

**if** self**.***checkmate***:**

font **=** pygame**.***font***.***SysFont***(**'Arial'**,** 36**)**

text **=** font**.***render***(**'Шах и мат!'**,** **True,** RED**)**

win**.***blit***(**text**,** **(**settings**.***WINDOW\_WIDTH***//**2 **-** text**.***get\_width***()//**2**,** settings**.***WINDOW\_HEIGHT***//**2 **-** text**.***get\_height***()//**2**))**

**elif** self**.***stalemate***:**

font **=** pygame**.***font***.***SysFont***(**'Arial'**,** 36**)**

text **=** font**.***render***(**'Пат!'**,** **True,** RED**)**

win**.***blit***(**text**,** **(**settings**.***WINDOW\_WIDTH***//**2 **-** text**.***get\_width***()//**2**,** settings**.***WINDOW\_HEIGHT***//**2 **-** text**.***get\_height***()//**2**))**

**elif** self**.***in\_check***(**self**.***white\_to\_move***):**

font **=** pygame**.***font***.***SysFont***(**'Arial'**,** 24**)**

text **=** font**.***render***(**'Шах!'**,** **True,** RED**)**

win**.***blit***(**text**,** **(**10**,** 10**))**

**def** is\_move\_valid**(**self**,** move**):**

"""

Проверяет, является ли ход допустимым.

:param move: Объект Move, представляющий ход.

:return: True, если ход допустим, иначе False.

"""

**return** move **in** self**.***get\_valid\_moves***()**

**def** get\_piece\_moves**(**self**,** r**,** c**):**

"""

Возвращает список допустимых ходов для фигуры на заданной позиции.

:param r: Строка, где находится фигура.

:param c: Столбец, где находится фигура.

:return: Список объектов Move.

"""

piece **=** self**.***board***[**r**][**c**]**

**if** piece **==** '--'**:**

**return** **[]**

**if** self**.***white\_to\_move* **and** piece**[**0**]** **!=** 'w'**:**

**return** **[]**

**if** **not** self**.***white\_to\_move* **and** piece**[**0**]** **!=** 'b'**:**

**return** **[]**

moves **=** **[]**

piece\_type **=** piece**[**1**]**

**if** piece\_type **==** 'K'**:**

self**.***get\_king\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'P'**:**

self**.***get\_pawn\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'Q'**:**

self**.***get\_queen\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'R'**:**

self**.***get\_rook\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'B'**:**

self**.***get\_bishop\_moves***(**r**,** c**,** moves**)**

**elif** piece\_type **==** 'N'**:**

self**.***get\_knight\_moves***(**r**,** c**,** moves**)**

valid\_moves **=** **[]**

**for** move **in** moves**:**

game\_copy **=** deepcopy**(**self**)**

game\_copy**.***make\_move***(**move**,** update\_state**=False)**

**if** **not** game\_copy**.***in\_check***(not** self**.***white\_to\_move***):**

valid\_moves**.***append***(**move**)**

**return** valid\_moves

# main.py

**import** pygame

**import** sys

**import** os

**import** json # Добавлен импорт json

**import** settings # Исправлено: импортируем как модуль

**from** settings **import** **\***

**from** auth **import** login**,** register

**from** database **import** initialize\_db**,** get\_games\_by\_user**,** create\_new\_game**,** get\_game\_by\_id

**from** game **import** Game**,** Move

**from** ai **import** find\_best\_move

pygame**.***init***()**

pygame**.***display***.***set\_caption***(**'Шахматный Эндшпиль: Король и Пешка - Король и Пешка'**)**

# Настройка режима отображения

**if** settings**.***FULLSCREEN***:**

info **=** pygame**.***display***.***Info***()**

settings**.***WINDOW\_WIDTH***,** settings**.***WINDOW\_HEIGHT* **=** info**.***current\_w***,** info**.***current\_h*

screen **=** pygame**.***display***.***set\_mode***((**settings**.***WINDOW\_WIDTH***,** settings**.***WINDOW\_HEIGHT***),** pygame**.***FULLSCREEN***)**

# Масштабирование CELL\_SIZE под разрешение экрана

settings**.***CELL\_SIZE* **=** **min(**settings**.***WINDOW\_WIDTH***,** settings**.***WINDOW\_HEIGHT***)** **//** settings**.***BOARD\_SIZE*

**else:**

screen **=** pygame**.***display***.***set\_mode***((**settings**.***WINDOW\_WIDTH***,** settings**.***WINDOW\_HEIGHT***))**

settings**.***CELL\_SIZE* **=** settings**.***CELL\_SIZE* # Используем значение из settings.py

clock **=** pygame**.***time***.***Clock***()**

**def** load\_images**():**

"""

Загружает изображения фигур и иконок.

:return: Словарь с изображениями.

"""

images **=** **{}**

filename\_to\_key **=** **{**

'king\_white.png'**:** 'wK'**,**

'king\_black.png'**:** 'bK'**,**

'queen\_white.png'**:** 'wQ'**,**

'queen\_black.png'**:** 'bQ'**,**

'pawn\_white.png'**:** 'wP'**,**

'pawn\_black.png'**:** 'bP'**,**

'bishop\_white.png'**:** 'wB'**,**

'bishop\_black.png'**:** 'bB'**,**

'horse\_white.png'**:** 'wN'**,** # Конь

'horse\_black.png'**:** 'bN'**,** # Конь

'tower\_white.png'**:** 'wR'**,** # Ладья

'tower\_black.png'**:** 'bR'**,** # Ладья

'eye\_open.png'**:** 'eye\_open'**,**

'eye\_closed.png'**:** 'eye\_closed'

**}**

**for** filename **in** os**.***listdir***(**settings**.***ASSETS\_PATH***):**

**if** filename**.***endswith***(**'.png'**):**

path **=** os**.***path***.***join***(**settings**.***ASSETS\_PATH***,** filename**)**

key **=** filename\_to\_key**.***get***(**filename**)**

**if** key**:**

**try:**

img **=** pygame**.***image***.***load***(**path**).***convert\_alpha***()**

# Масштабирование изображения

**if** filename **in** **[**'eye\_open.png'**,** 'eye\_closed.png'**]:**

eye\_size **=** 30 # Размер иконки глаза

img **=** pygame**.***transform***.***scale***(**img**,** **(**eye\_size**,** eye\_size**))**

**else:**

img **=** pygame**.***transform***.***scale***(**img**,** **(**settings**.***CELL\_SIZE***,** settings**.***CELL\_SIZE***))**

images**[**key**]** **=** img

**print(**f"Загружено изображение: {filename}"**)**

**except** pygame**.***error* **as** e**:**

**print(**f"Ошибка загрузки изображения {path}: {e}"**)**

**else:**

**print(**f"Неизвестное изображение: {filename}"**)**

required\_keys **=** **[**'wK'**,** 'bK'**,** 'wP'**,** 'bP'**,** 'wQ'**,** 'bQ'**,** 'wR'**,** 'bR'**,** 'wB'**,** 'bB'**,** 'wN'**,** 'bN'**,** 'eye\_open'**,** 'eye\_closed'**]**

**for** key **in** required\_keys**:**

**if** key **not** **in** images**:**

**print(**f"Предупреждение: Изображение для {key} не найдено."**)**

**return** images

images **=** load\_images**()**

**def** draw\_text**(**win**,** text**,** size**,** color**,** x**,** y**):**

"""

Отрисовывает текст на экране.

:param win: Объект окна Pygame.

:param text: Текст для отрисовки.

:param size: Размер текста.

:param color: Цвет текста.

:param x: Координата X.

:param y: Координата Y.

"""

font **=** pygame**.***font***.***SysFont***(**'Arial'**,** size**)**

text\_surface **=** font**.***render***(**text**,** **True,** color**)**

win**.***blit***(**text\_surface**,** **(**x**,** y**))**

**def** auth\_screen**():**

"""

Отображает экран аутентификации.

:return: Имя пользователя, если вход успешен.

"""

message **=** ''

**while** **True:**

screen**.***fill***(**BLACK**)**

draw\_text**(**screen**,** 'Шахматный Эндшпиль'**,** 60**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 200**,** 50**)**

draw\_text**(**screen**,** '1. Войти'**,** 40**,** WHITE**,** 100**,** 200**)**

draw\_text**(**screen**,** '2. Зарегистрироваться'**,** 40**,** WHITE**,** 100**,** 300**)**

draw\_text**(**screen**,** '3. Выход'**,** 40**,** WHITE**,** 100**,** 400**)**

draw\_text**(**screen**,** 'Нажмите TAB для переключения между опциями'**,** 20**,** WHITE**,** 100**,** 500**)**

draw\_text**(**screen**,** message**,** 30**,** RED**,** 100**,** 600**)**

pygame**.***display***.***flip***()**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**if** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_1***:**

success**,** msg**,** user **=** login\_prompt**()**

**if** success**:**

**return** user

**else:**

message **=** msg

**if** event**.***key* **==** pygame**.***K\_2***:**

success**,** msg **=** register\_prompt**()**

message **=** msg

**if** event**.***key* **==** pygame**.***K\_3***:**

pygame**.quit()**

sys**.exit()**

clock**.***tick***(**FPS**)**

**def** login\_prompt**():**

"""

Отображает экран входа.

:return: Кортеж (успех, сообщение, имя пользователя), где успех - True, если вход успешен.

"""

username **=** ''

password **=** ''

input\_box **=** 'username'

message **=** ''

password\_visible **=** **False**

eye\_icon **=** images**.***get***(**'eye\_closed'**)**

eye\_rect **=** pygame**.***Rect***(**settings**.***WINDOW\_WIDTH***//**2 **+** 150**,** 300**,** 30**,** 30**)** # Позиция иконки глаза

**while** **True:**

screen**.***fill***(**BLACK**)**

draw\_text**(**screen**,** 'Вход'**,** 50**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 80**,** 50**)**

draw\_text**(**screen**,** 'Имя пользователя:'**,** 35**,** WHITE**,** 100**,** 200**)**

draw\_text**(**screen**,** username**,** 35**,** WHITE**,** 500**,** 200**)**

draw\_text**(**screen**,** 'Пароль:'**,** 35**,** WHITE**,** 100**,** 300**)**

**if** password\_visible**:**

display\_password **=** password

**else:**

display\_password **=** '\*' **\*** **len(**password**)**

draw\_text**(**screen**,** display\_password**,** 35**,** WHITE**,** 500**,** 300**)**

draw\_text**(**screen**,** 'Нажмите TAB для переключения между полями ввода'**,** 25**,** WHITE**,** 100**,** 400**)**

draw\_text**(**screen**,** 'Нажмите ESC для возврата в главное меню'**,** 25**,** WHITE**,** 100**,** 450**)**

draw\_text**(**screen**,** message**,** 30**,** RED**,** 100**,** 500**)**

# Подсветка активного поля

**if** input\_box **==** 'username'**:**

pygame**.***draw***.***rect***(**screen**,** BLUE**,** pygame**.***Rect***(**500**,** 200**,** 200**,** 50**),** 3**)**

**else:**

pygame**.***draw***.***rect***(**screen**,** BLUE**,** pygame**.***Rect***(**500**,** 300**,** 200**,** 50**),** 3**)**

# Рисуем иконку глаза

**if** password\_visible**:**

eye\_icon **=** images**.***get***(**'eye\_open'**)**

**else:**

eye\_icon **=** images**.***get***(**'eye\_closed'**)**

**if** eye\_icon**:**

screen**.***blit***(**eye\_icon**,** eye\_rect**)**

pygame**.***display***.***flip***()**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**if** event**.type** **==** pygame**.***MOUSEBUTTONDOWN***:**

**if** eye\_rect**.***collidepoint***(**event**.***pos***):**

password\_visible **=** **not** password\_visible

**if** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_TAB***:**

input\_box **=** 'password' **if** input\_box **==** 'username' **else** 'username'

**elif** event**.***key* **==** pygame**.***K\_ESCAPE***:**

**return** **False,** 'Возврат в главное меню.'**,** **None**

**elif** event**.***key* **==** pygame**.***K\_RETURN***:**

**if** **len(**username**)** **<** 3 **or** **len(**password**)** **<** 6**:**

message **=** 'Имя пользователя ≥ 3 символа, пароль ≥ 6 символов.'

**else:**

success**,** msg**,** user **=** login**(**username**,** password**)**

**if** success**:**

**return** **True,** msg**,** user

**else:**

message **=** msg

**elif** event**.***key* **==** pygame**.***K\_BACKSPACE***:**

**if** input\_box **==** 'username'**:**

username **=** username**[:-**1**]**

**else:**

password **=** password**[:-**1**]**

**else:**

**if** input\_box **==** 'username' **and** event**.***unicode***.***isprintable***():**

username **+=** event**.***unicode*

**elif** input\_box **==** 'password' **and** event**.***unicode***.***isprintable***():**

password **+=** event**.***unicode*

clock**.***tick***(**FPS**)**

**def** register\_prompt**():**

"""

Отображает экран регистрации.

:return: Кортеж (успех, сообщение), где успех - True, если регистрация успешна.

"""

username **=** ''

password **=** ''

input\_box **=** 'username'

message **=** ''

password\_visible **=** **False**

eye\_icon **=** images**.***get***(**'eye\_closed'**)**

eye\_rect **=** pygame**.***Rect***(**settings**.***WINDOW\_WIDTH***//**2 **+** 150**,** 300**,** 30**,** 30**)** # Позиция иконки глаза

**while** **True:**

screen**.***fill***(**BLACK**)**

draw\_text**(**screen**,** 'Регистрация'**,** 50**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 120**,** 50**)**

draw\_text**(**screen**,** 'Имя пользователя:'**,** 35**,** WHITE**,** 100**,** 200**)**

draw\_text**(**screen**,** username**,** 35**,** WHITE**,** 500**,** 200**)**

draw\_text**(**screen**,** 'Пароль:'**,** 35**,** WHITE**,** 100**,** 300**)**

**if** password\_visible**:**

display\_password **=** password

**else:**

display\_password **=** '\*' **\*** **len(**password**)**

draw\_text**(**screen**,** display\_password**,** 35**,** WHITE**,** 500**,** 300**)**

draw\_text**(**screen**,** 'Нажмите TAB для переключения между полями ввода'**,** 25**,** WHITE**,** 100**,** 400**)**

draw\_text**(**screen**,** 'Нажмите ESC для возврата в главное меню'**,** 25**,** WHITE**,** 100**,** 450**)**

draw\_text**(**screen**,** message**,** 30**,** RED**,** 100**,** 500**)**

# Подсветка активного поля

**if** input\_box **==** 'username'**:**

pygame**.***draw***.***rect***(**screen**,** BLUE**,** pygame**.***Rect***(**500**,** 200**,** 200**,** 50**),** 3**)**

**else:**

pygame**.***draw***.***rect***(**screen**,** BLUE**,** pygame**.***Rect***(**500**,** 300**,** 200**,** 50**),** 3**)**

# Рисуем иконку глаза

**if** password\_visible**:**

eye\_icon **=** images**.***get***(**'eye\_open'**)**

**else:**

eye\_icon **=** images**.***get***(**'eye\_closed'**)**

**if** eye\_icon**:**

screen**.***blit***(**eye\_icon**,** eye\_rect**)**

pygame**.***display***.***flip***()**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**if** event**.type** **==** pygame**.***MOUSEBUTTONDOWN***:**

**if** eye\_rect**.***collidepoint***(**event**.***pos***):**

password\_visible **=** **not** password\_visible

**if** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_TAB***:**

input\_box **=** 'password' **if** input\_box **==** 'username' **else** 'username'

**elif** event**.***key* **==** pygame**.***K\_ESCAPE***:**

**return** **False,** 'Возврат в главное меню.'

**elif** event**.***key* **==** pygame**.***K\_RETURN***:**

**if** **len(**username**)** **<** 3 **or** **len(**password**)** **<** 6**:**

message **=** 'Имя пользователя ≥ 3 символа, пароль ≥ 6 символов.'

**else:**

success**,** msg **=** register**(**username**,** password**)**

**if** success**:**

message **=** 'Регистрация успешна. Можете войти.'

**else:**

message **=** msg

**elif** event**.***key* **==** pygame**.***K\_BACKSPACE***:**

**if** input\_box **==** 'username'**:**

username **=** username**[:-**1**]**

**else:**

password **=** password**[:-**1**]**

**else:**

**if** input\_box **==** 'username' **and** event**.***unicode***.***isprintable***():**

username **+=** event**.***unicode*

**elif** input\_box **==** 'password' **and** event**.***unicode***.***isprintable***():**

password **+=** event**.***unicode*

clock**.***tick***(**FPS**)**

**def** select\_mode**(**username**):**

"""

Отображает экран выбора режима игры.

:param username: Имя пользователя.

:return: Режим игры ('ai' или None).

"""

message **=** ''

**while** **True:**

screen**.***fill***(**BLACK**)**

draw\_text**(**screen**,** 'Выберите режим игры'**,** 60**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 200**,** 50**)**

draw\_text**(**screen**,** '1. Человек против искусственного интеллекта'**,** 40**,** WHITE**,** 100**,** 200**)**

draw\_text**(**screen**,** '2. Просмотреть текущие игры'**,** 40**,** WHITE**,** 100**,** 300**)**

draw\_text**(**screen**,** '3. Выйти'**,** 40**,** WHITE**,** 100**,** 400**)**

draw\_text**(**screen**,** message**,** 30**,** RED**,** 100**,** 500**)**

pygame**.***display***.***flip***()**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**if** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_1***:**

**return** 'ai'

**if** event**.***key* **==** pygame**.***K\_2***:**

view\_games**(**username**)**

**if** event**.***key* **==** pygame**.***K\_3***:**

pygame**.quit()**

sys**.exit()**

clock**.***tick***(**FPS**)**

**def** view\_games**(**username**):**

"""

Отображает список текущих игр пользователя.

:param username: Имя пользователя.

"""

games **=** get\_games\_by\_user**(**username**,** status**=**'in\_progress'**)**

selected\_game **=** **None**

**while** **True:**

screen**.***fill***(**BLACK**)**

draw\_text**(**screen**,** f'Текущие партии пользователя: {username}'**,** 40**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 250**,** 50**)**

y\_offset **=** 150

**if** **not** games**:**

draw\_text**(**screen**,** 'Нет текущих партий. Начните новую игру.'**,** 30**,** WHITE**,** 50**,** y\_offset**)**

**else:**

**for** index**,** game **in** **enumerate(**games**[:**10**]):** # Отображаем последние 10 партий

game\_id**,** white\_player**,** black\_player**,** moves**,** result**,** start\_time**,** end\_time**,** status **=** game

game\_info **=** f"{index**+**1}. ID: {game\_id} | Белые: {white\_player} | Черные: {black\_player} | Начало: {start\_time}"

draw\_text**(**screen**,** game\_info**,** 25**,** WHITE**,** 50**,** y\_offset**)**

y\_offset **+=** 40

**if** y\_offset **>** settings**.***WINDOW\_HEIGHT* **-** 100**:**

**break**

draw\_text**(**screen**,** 'Нажмите число партии для возобновления или ESC для возврата'**,** 25**,** WHITE**,** 50**,** settings**.***WINDOW\_HEIGHT* **-** 100**)**

pygame**.***display***.***flip***()**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**if** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_ESCAPE***:**

**return**

**elif** pygame**.***K\_1* **<=** event**.***key* **<=** pygame**.***K\_9***:**

selected\_index **=** event**.***key* **-** pygame**.***K\_1*

**if** selected\_index **<** **len(**games**):**

selected\_game **=** games**[**selected\_index**]**

resume\_game**(**selected\_game**)**

clock**.***tick***(**FPS**)**

**def** resume\_game**(**game**):**

"""

Возобновляет игру по её ID.

:param game: Кортеж с данными игры.

"""

game\_id**,** white\_player**,** black\_player**,** moves**,** result**,** start\_time**,** end\_time**,** status **=** game

game\_instance **=** Game**(**white\_player**=**white\_player**,** black\_player**=**black\_player**,** game\_id**=**game\_id**)**

game\_screen\_instance**(**game\_instance**)**

**def** show\_game\_details**(**game**):**

"""

Отображает детали игры.

:param game: Кортеж с данными игры.

"""

game\_id**,** white\_player**,** black\_player**,** moves**,** result**,** start\_time**,** end\_time**,** status **=** game

moves\_list **=** json**.***loads***(**moves**)** # Преобразование JSON строки обратно в список

**while** **True:**

screen**.***fill***(**BLACK**)**

draw\_text**(**screen**,** f'Партия ID: {game\_id}'**,** 40**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 100**,** 50**)**

draw\_text**(**screen**,** f"Белые: {white\_player} | Черные: {black\_player}"**,** 30**,** WHITE**,** 50**,** 100**)**

draw\_text**(**screen**,** f"Результат: {result}"**,** 30**,** WHITE**,** 50**,** 150**)**

draw\_text**(**screen**,** f"Начало: {start\_time} | Конец: {end\_time}"**,** 30**,** WHITE**,** 50**,** 200**)**

draw\_text**(**screen**,** 'Ходы:'**,** 30**,** WHITE**,** 50**,** 250**)**

y\_offset **=** 300

**for** i **in** **range(**0**,** **len(**moves\_list**),** 2**):**

move\_number **=** i**//**2 **+** 1

white\_move **=** moves\_list**[**i**].***get\_chess\_notation***()**

black\_move **=** moves\_list**[**i**+**1**].***get\_chess\_notation***()** **if** i**+**1 **<** **len(**moves\_list**)** **else** ''

move\_text **=** f"{move\_number}. {white\_move} {black\_move}"

draw\_text**(**screen**,** move\_text**,** 25**,** WHITE**,** 50**,** y\_offset**)**

y\_offset **+=** 30

**if** y\_offset **>** settings**.***WINDOW\_HEIGHT* **-** 100**:**

**break**

draw\_text**(**screen**,** 'Нажмите ESC для возврата к списку партий'**,** 25**,** WHITE**,** 50**,** settings**.***WINDOW\_HEIGHT* **-** 50**)**

pygame**.***display***.***flip***()**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**if** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_ESCAPE***:**

**return**

clock**.***tick***(**FPS**)**

**def** game\_screen\_instance**(**game\_instance**):**

"""

Отображает экран игры.

:param game\_instance: Экземпляр класса Game.

"""

selected\_square **=** **None**

valid\_moves **=** **[]**

run **=** **True**

paused **=** **False**

**while** run**:**

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

run **=** **False**

pygame**.quit()**

sys**.exit()**

**elif** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_p***:**

paused **=** **not** paused

**elif** event**.type** **==** pygame**.***MOUSEBUTTONDOWN***:**

**if** **not** paused **and** **not** game\_instance**.***checkmate* **and** **not** game\_instance**.***stalemate***:**

pos **=** pygame**.***mouse***.***get\_pos***()**

row**,** col **=** pos**[**1**]** **//** settings**.***CELL\_SIZE***,** pos**[**0**]** **//** settings**.***CELL\_SIZE*

piece **=** game\_instance**.***board***[**row**][**col**]**

**if** selected\_square**:**

move **=** Move**(**selected\_square**,** **(**row**,** col**),** game\_instance**.***board***[**selected\_square**[**0**]][**selected\_square**[**1**]],** game\_instance**.***board***[**row**][**col**])**

**if** move**.***is\_pawn\_promotion***:**

move**.***promotion\_choice* **=** 'Q' # Можно добавить выбор фигуры через интерфейс

**if** game\_instance**.***is\_move\_valid***(**move**):**

game\_instance**.***make\_move***(**move**)**

selected\_square **=** **None**

valid\_moves **=** **[]**

# Если игра против ИИ и ход после этого принадлежит ИИ

**if** **isinstance(**game\_instance**.***black\_player***,** **str)** **and** game\_instance**.***black\_player***.***lower***()** **==** 'ai' **and** **not** game\_instance**.***white\_to\_move* **and** **not** game\_instance**.***checkmate* **and** **not** game\_instance**.***stalemate***:**

ai\_move **=** find\_best\_move**(**game\_instance**,** depth**=**settings**.***AI\_DEPTH***)**

**if** ai\_move**:**

game\_instance**.***make\_move***(**ai\_move**)**

**else:**

**if** piece **!=** '--' **and** **((**game\_instance**.***white\_to\_move* **and** piece**[**0**]** **==** 'w'**)** **or** **(not** game\_instance**.***white\_to\_move* **and** piece**[**0**]** **==** 'b'**)):**

selected\_square **=** **(**row**,** col**)**

valid\_moves **=** game\_instance**.***get\_piece\_moves***(**row**,** col**)**

**else:**

selected\_square **=** **None**

valid\_moves **=** **[]**

**else:**

**if** piece **!=** '--' **and** **((**game\_instance**.***white\_to\_move* **and** piece**[**0**]** **==** 'w'**)** **or** **(not** game\_instance**.***white\_to\_move* **and** piece**[**0**]** **==** 'b'**)):**

selected\_square **=** **(**row**,** col**)**

valid\_moves **=** game\_instance**.***get\_piece\_moves***(**row**,** col**)**

**if** **not** paused**:**

screen**.***fill***(**BLACK**)**

game\_instance**.***draw***(**screen**,** images**,** selected\_square**,** valid\_moves**)**

**else:**

# Показать паузу

screen**.***fill***(**GRAY**)**

draw\_text**(**screen**,** 'Пауза'**,** 60**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 100**,** settings**.***WINDOW\_HEIGHT***//**2 **-** 50**)**

draw\_text**(**screen**,** 'Нажмите S для сохранения и выхода'**,** 30**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 150**,** settings**.***WINDOW\_HEIGHT***//**2 **+** 20**)**

draw\_text**(**screen**,** 'Нажмите P для продолжения игры'**,** 30**,** WHITE**,** settings**.***WINDOW\_WIDTH***//**2 **-** 150**,** settings**.***WINDOW\_HEIGHT***//**2 **+** 60**)**

# Проверка событий в паузе

**for** event **in** pygame**.***event***.***get***():**

**if** event**.type** **==** pygame**.***QUIT***:**

pygame**.quit()**

sys**.exit()**

**elif** event**.type** **==** pygame**.***KEYDOWN***:**

**if** event**.***key* **==** pygame**.***K\_s***:**

# Сохранить и выйти в главное меню

run **=** **False**

**elif** event**.***key* **==** pygame**.***K\_p***:**

paused **=** **False**

pygame**.***display***.***flip***()**

clock**.***tick***(**FPS**)**

**if** **not** paused **and** **(**game\_instance**.***checkmate* **or** game\_instance**.***stalemate***):**

# Отображение результата уже происходит в draw\_game\_state

# Добавим задержку перед возвратом в главное меню

pygame**.***time***.***delay***(**5000**)**

run **=** **False**

# Если игра завершена, обновляем статус

**if** game\_instance**.***result***:**

game\_instance**.***save\_game\_completion***()**

**def** game\_screen**(**mode**,** white\_player**=**'White'**,** black\_player**=**'AI'**):**

"""

Отображает экран игры.

:param mode: Режим игры ('ai' или другой).

:param white\_player: Имя пользователя, играющего за белых.

:param black\_player: Имя пользователя, играющего за чёрных.

"""

# Создание новой игры или загрузка существующей

**if** mode **==** 'ai'**:**

**if** black\_player**.***lower***()** **==** 'ai'**:**

game\_id **=** create\_new\_game**(**white\_player**,** 'AI'**)**

game\_instance **=** Game**(**white\_player**=**white\_player**,** black\_player**=**'AI'**,** game\_id**=**game\_id**)**

**else:**

game\_id **=** create\_new\_game**(**white\_player**,** black\_player**)**

game\_instance **=** Game**(**white\_player**=**white\_player**,** black\_player**=**black\_player**,** game\_id**=**game\_id**)**

run\_game **=** **True**

**while** run\_game**:**

game\_screen\_instance**(**game\_instance**)**

run\_game **=** **False** # После завершения игры возвращаемся к выбору режима

**def** main**():**

"""

Основная функция, запускающая приложение.

"""

initialize\_db**()** # Инициализация базы данных при запуске приложения

current\_user **=** auth\_screen**()**

**if** current\_user**:**

mode **=** select\_mode**(**current\_user**)**

**if** mode **==** 'ai'**:**

game\_screen**(**mode**,** white\_player**=**current\_user**,** black\_player**=**'AI'**)**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

main**()**

# settings.py

**import** os

# Размер клетки в пикселях (будет масштабироваться при полноэкранном режиме)

CELL\_SIZE **=** 100

# Размер доски

BOARD\_SIZE **=** 8

# Цвета

WHITE **=** **(**255**,** 255**,** 255**)**

BLACK **=** **(**0**,** 0**,** 0**)**

GRAY **=** **(**128**,** 128**,** 128**)**

BLUE **=** **(**0**,** 0**,** 255**)**

GREEN **=** **(**0**,** 255**,** 0**)**

RED **=** **(**255**,** 0**,** 0**)**

# Путь к папке с изображениями

ASSETS\_PATH **=** os**.***path***.***join***(**os**.***path***.***dirname***(**\_\_file\_\_**),** 'assets'**)**

# Глубина поиска AI

AI\_DEPTH **=** 3

# Частота кадров

FPS **=** 60

# Флаг полноэкранного режима

FULLSCREEN **=** **False** # Установите в True для запуска игры в полноэкранном режиме

# Изначальные размеры окна (будут обновлены в main.py при запуске)

WINDOW\_WIDTH **=** 800

WINDOW\_HEIGHT **=** 800

# **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

# **Приложение 1**

