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»**

**«Компьютерная игра эндшпиль “Король, 2 пешки -Король, пешка”»**

Р.02069337.22/257-12 ПЗ-01

Листов 51

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2025 г.

**Исполнитель**:

студент гр. ИСТбд-31

*Кузьмин Владимир Сергеевич*

« » 2025 г.

**2025**

Содержание

**Аннотация...............................................................................................................3**

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ...............................................................................4**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА........................................................................9**

**РУКОВОДСТВО ПРОГРАММИСТА..........................................................….16**  
**Текст программы.............................................................................................…..34**

# **Аннотация**

Данный документ представляет собой пояснительную записку на курсовую работу на тему «Компьютерная игра эндшпиль “Король, 2 пешки -Король, пешка”». Документ содержит следующие разделы: техническое задание, пояснительная записка и руководство программиста, код программы; в нем излагается постановка задачи и описание реализуемой программы, ее назначение. Документ может быть использован в качестве инструкции для применения рассматриваемого программного средства

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»**

**Тема:** **Компьютерная игра эндшпиль «Король, 2 пешки -Король, пешка»**

Р.02069337.22/257-12 ТЗ-01

Листов 5

**Исполнитель**:

студент гр. ИСТбд-31

*Кузьмин Владимир Сергеевич*

« » 2025 г.

**2025**

**Введение**

Разрабатываемое приложение под наименованием «Эндшпиль, король и пешки», представляет собой шахматный эндшпиль, моделирующий ситуацию «Король и две пешки против Короля и пешки». Игроки поочередно перемещаются своими фигурами на игровом поле. Один игрок играет за фигуры черного цвета, а второй игрок - за фигуры белого цвета. Цель игры заключается в том, чтобы забрать короля противника. Игра продолжается до момента пока один из игроков не “срубит” вражеского короля.

**1. Основания для разработки**

Основание для разработки является учебный план направления 09.03.02 "Информационные системы и технологии", а также распоряжение по факультету.

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

**2.1. Функциональное назначение**

**Функциональное назначение**

Требуется разработать однопользовательское десктопное приложение по теме: Компьютерная игра эндшпиль «Король, 2 пешки -Король, пешка» - с графическим интерфейсом в среде Windows.

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно быть разработано в виде одного модуля с дополнительными информационными файлами при необходимости.

Приложение должно соответствовать следующим правилам игры.

**Поле и игроки.** Игра ведется между двумя соперниками: пользователем и компьютером. Игровое поле представляет собой стандартную шахматную доску размером 8×8 клеток. Игрок управляет белыми фигурами, а компьютер — черными.

**Специальные термины.** В дальнейшем будут использоваться следующие термины:

Шах - ситуация, в которой король находится под ударом фигуры противника и должен защититься, иначе проиграет на следующем ходу врага.

Рубка - взятие фигуры противника, при котором одна фигура перемещается на клетку, занятую другой, и удаляет её с доски.

**Порядок ходов.** Первый ход делает игрок с белыми фигурами. После этого игроки поочередно передвигают свои фигуры по позициям на игровой доске.

**Цель игры.** Срубить вражеского короля раньше, чем это сделает противник. Игрок, у которого остается король на доске - побеждает, а второй игрок считается проигравшим

2.2.2 Требования к составу функций приложения

В приложении должны быть реализованы в графическом режиме следующие основные функции:

- регистрация/авторизация пользователя;

- отрисовка игрового поля;

- взаимодействие с пользователем;

- проверка правильности и отрисовка хода пользователя;

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода компьютера;

- информирование пользователя об окончании игры и победителе.

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Изображения фигур могут храниться в отдельных графических файлах. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле. Ключ от шифрования так же хранится в виде отдельного файла. Информационные сообщения для пользователя должны выводиться во всплывающих сообщениях.

**2.3 Требования к надёжности**

Приложение должно быть стабильным и работоспособным, не вызывать сбоев или ошибок. В случае сбоя или ошибки, приложение должно быть способно восстановиться без потери данных.

**2.4 Требования к информационной и программной совместимости**

Операционная система: Windows 10.

Версия языка программирования: Python 3.13.2

Среда разработки: Visual Studio Code 1.98.0

При создании программы используются встроенные библиотеки “os”, “random”, “json”, “hashlib” и сторонние библиотеки “tkinter 8.6”, “tkinter.messagebox”, “pycryptodome” (модули Crypto.Cipher, Crypto.Util.Padding, Crypto.Random), а также “base64” для кодирования данных.

**2.5 Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

**2.6 Требования к транспортированию и хранению**

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются.

2.6 2 Условия хранения

Проект будет храниться в репозитории на сайте github.com по ссылке https://github.com/Ungergoo/KRaisd

2.6 3 Сроки хранения

Срок хранения – до июля 2026 года.

**3. Требования к программной документации**

Определяются заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приёмки**

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»

**Курсовая работа**

**по дисциплине «Алгоритмы и структуры данных»**

# **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**Тема:** **Компьютерная игра эндшпиль «Король, 2 пешки -Король, пешка»**

Р.02069337.22/257-12 ПЗ-01

Листов 7

**Исполнитель**:

студент гр. ИСТбд-31

*Кузьмин Владимир Сергеевич*

« » 2025 г.

**2025**

**Введение**

Приложение «Эндшпиль» реализует функционал шахматной игры в формате эндшпиля.

В качестве подхода для разработки была выбрана каскадная модель разработки. Каскадная модель была выбрана из-за своей простоты, она позволяет наглядно представить объём работы и сроки выполнения, а также эффективно разбить проект на несколько подзадач.

Приложение «Эндшпиль» представляет собой игру в шахматы для двоих игроков, с ограниченным количеством фигур на стандартном шахматном столе размером 8 на 8 клеток.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании.

**1.2 Математические методы**

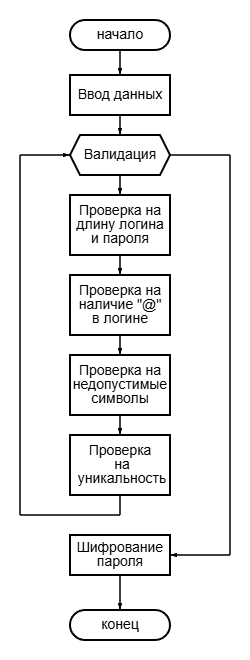
В качестве математической модели для представления и отрисовки шахматного поля был выбран двумерный массив размером 8×8. Этот массив позволяет удобно хранить текущее расположение фигур, выполнять проверки на возможные ходы и определять их корректность. В массиве пустые клетки представлены значением None, чёрные фигуры обозначаются строчными буквами (‘p’ - пешка, ‘q’ - ферзь, ’k’ - король), а белые — заглавными (‘P’ - пешка, ‘Q’ - ферзь, ‘K’ - король).

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

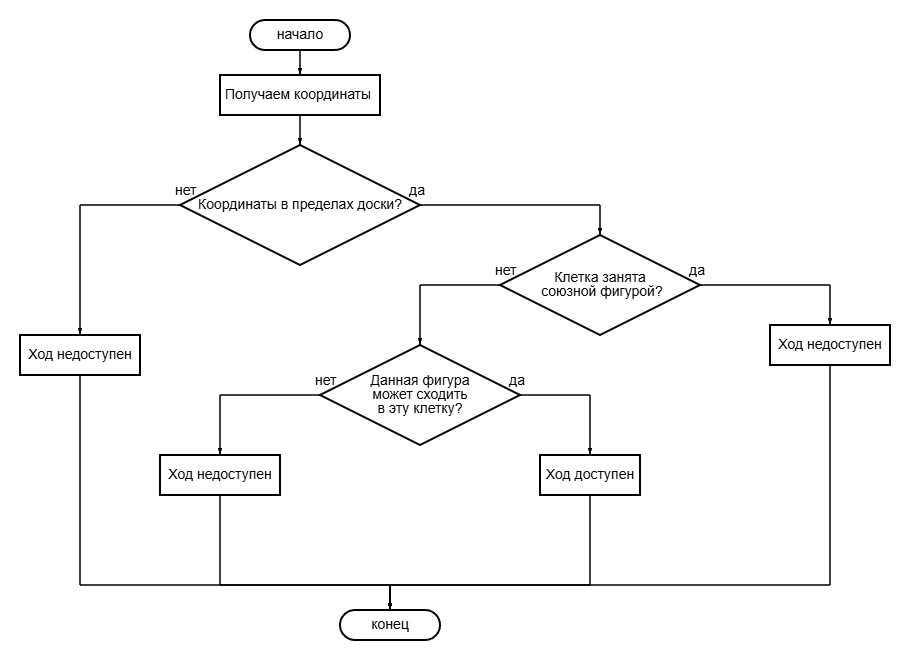


1.3.2.1. Алгоритм авторизации

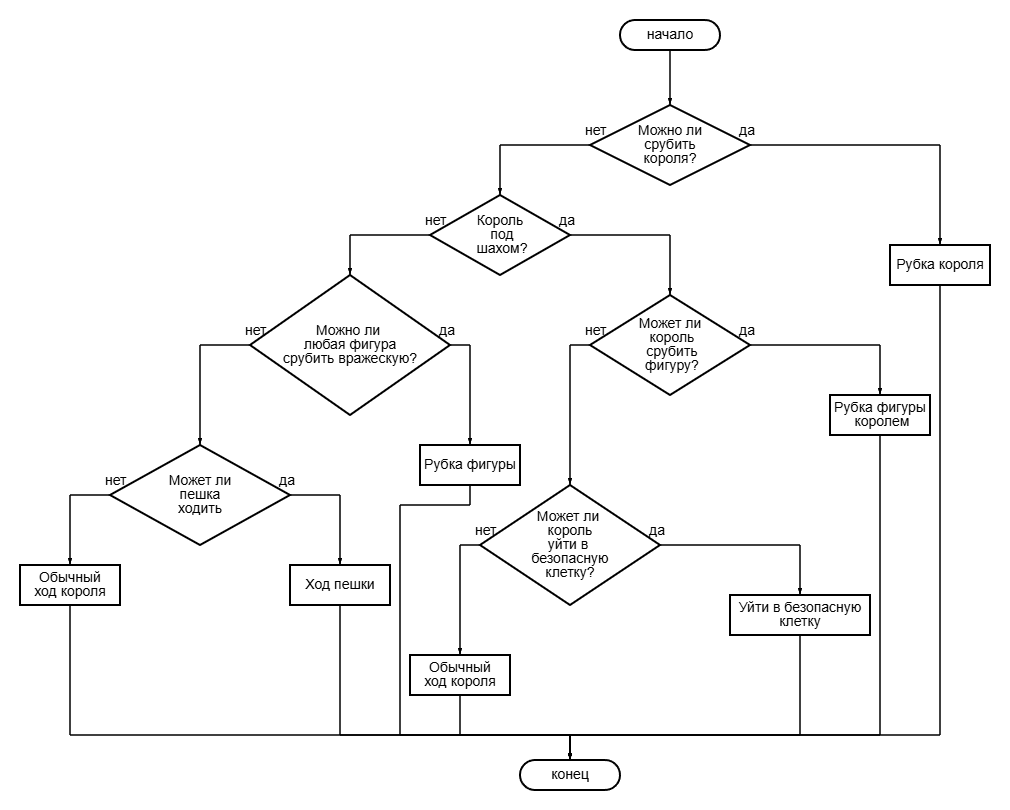


Данный алгоритм регистрации пользователя необходим для безопасности данных пользователей. Он принимает на вход пароль и логин пользователя, осуществляет проверку соответствия установленным требованиям, таким как минимальная длинна и наличие “@” в логине, после чего шифрует пароль, а затем сохраняет полученные учетные данные для последующего использования при входе в систему.

1.3.2.3. Алгоритм проверки доступности хода

 Данный алгоритм регистрации хода необходим для контроля за соблюдением правил игры «Эндшпиль: Король и пешки». Алгоритм принимает на вход координаты выбранной клетки на шахматной доске, где первая координата указывает на столбец, а вторая на строку. После получения этих координат алгоритм проверяет возможность совершения хода, основываясь на правилах шахмат. Если ход возможен, фигура перемещается в указанную клетку. В случае недопустимого хода игрок должен выбрать другое место.

1.3.2.4. Алгоритм выбора хода компьютером

 Данный алгоритм необходим для осмысленных ходов компьютера. В алгоритме предусмотрено решение на атаки противника, а также выставлен приоритет действий для компьютера. Самый большой приоритет имеет взятие вражеского короля, а самый маленький приоритет - обычный ход короля.

**1.4 Тестирование**

Весь процесс тестирования проходил вручную, без привлечения специального ПО. На протяжении хода разработки, использовался метод белого ящика, так как в любом время имелся доступ ко всем компонентам программы. Всё тестирование выполнялось без подготовки специальных тестов.

В процессе разработки каждая новая функция проходила системное тестирование для выявления и устранения возможных ошибок. После успешного прохождения тестирования функция считалась полностью интегрированной в программу.

**2. Источники, использованные при разработке**

1. Шишкин, В.В. Разработка логических компьютерных игр с графическим интерфейсом в среде Python [Электронный ресурс] / В.В. Шишкин, Д.С. Афонин. – Ульяновск: УлГТУ, 2023. – 89 с. – Режим доступа: для всех пользователей. – URL: <http://lib.ulstu.ru/venec/disk/2023/112.pdf> (дата обращения: 05.03.25).

2. Graphical User Interfaces with Tk [Электронный ресурс]. – Режим доступа: для всех пользователей. – URL: <https://docs.python.org/3/library/tk.html> (дата обращения: 07.03.25).

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»**

# **РУКОВОДСТВО ПРОГРАММИСТА**

**Тема:** **Компьютерная игра эндшпиль «Король, 2 пешки -Король, пешка»**

Р.02069337.22/257-12 РП-01

Листов 18

**Исполнитель**:

студент гр. ИСТбд-31

*Кузьмин Владимир Сергеевич*

« » 2025 г.

**2025**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение «Эндшпиль, король и пешки» реализовано для развлечения пользователей. Приложение реализовано по следующим правилам:

**Поле и игроки.** Игра ведется между двумя соперниками: пользователем и компьютером. Игровое поле представляет собой стандартную шахматную доску размером 8×8 клеток. Игрок управляет белыми фигурами, а компьютер — черными. Фигуры перемещаются по стандартным шахматным правилам.

**Порядок ходов.** Игроки ходят по очереди. Первый ход делает игрок за белые фигуры.

**Цель игры.** Срубить вражеского короля раньше противника. Игрок. Последний оставшийся с королем на доске - побеждает.

В приложении предоставляется возможность авторизоваться и входить в свой аккаунт. В приложении реализованы правила игры в шахматы - ходы фигур, превращение пешек в ферзи, рубка фигур.

**1.2 Условия, необходимые для использования приложения**

Рекомендуется к использованию на Windows 10.

При создании программы используются встроенные библиотеки “os”, “random”, “json”, “hashlib” и сторонние библиотеки “tkinter 8.6”, “tkinter.messagebox”, “pycryptodome” (модули Crypto.Cipher, Crypto.Util.Padding, Crypto.Random), а также “base64” для кодирования данных.

Разработка велась в Visual Studio Code 1.98.0. на версии языка программирования Python 3.13.2

**2. Характеристики программы**

**2.1 Характеристики приложения**

Значимых строк кода 515

Структура данных одна – массив.

**Использованные библиотеки:**

**“os”** – стандартная библиотека для работы с операционной системой. Позволяет управлять файлами и директориями, получать доступ к переменным окружения и выполнять системные команды.

**“tkinter”** – встроенная библиотека для создания графического интерфейса. Позволяет создавать окна, кнопки, поля ввода и другие элементы GUI.

**“messagebox (tkinter)”** – модуль для вывода всплывающих окон с сообщениями, предупреждениями и запросами подтверждения.

**“Json"** – модуль для работы с JSON-форматом, позволяющий сохранять и загружать данные в файлы или передавать их между программами.

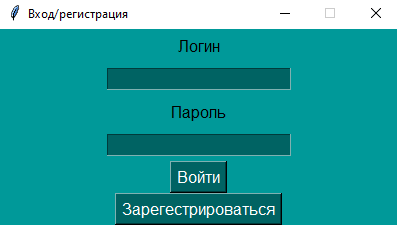
**“random”** – библиотека для генерации случайных чисел и значений. Используется, например, при выборе случайных позиций на доске.

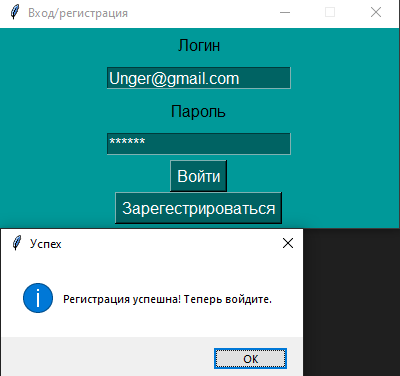
**“Crypto.Cipher (AES)”** – модуль из библиотеки PyCryptodome, предоставляющий реализацию алгоритма шифрования AES. Используется для безопасного хранения паролей.

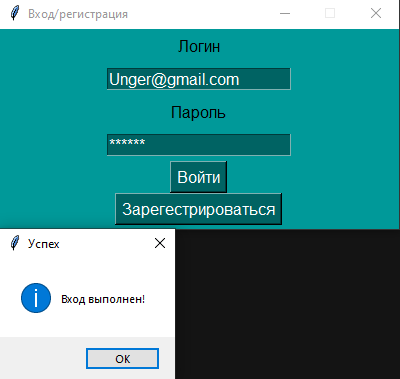
**“Crypto.Util.Padding”** – вспомогательный модуль для дополнения (padding) данных перед шифрованием.

**“Crypto.Random”** – модуль для генерации криптографически стойких случайных данных, например, ключей шифрования.

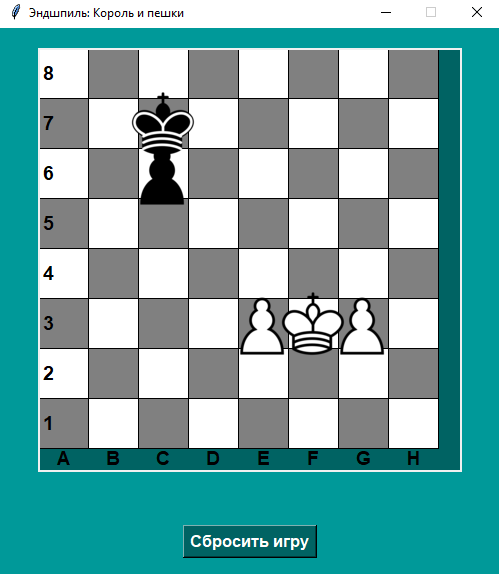
**“Base64"** – библиотека для кодирования и декодирования данных в формате Base64, используется при шифровании паролей.

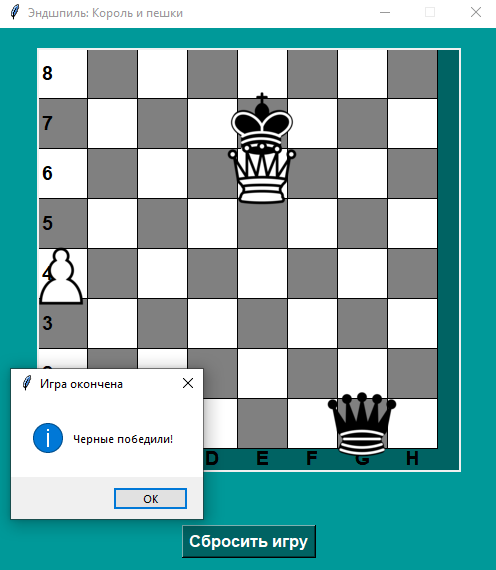
Работа приложения начинается с окна авторизации и регистрации: 

Успешная регистрация:  


Успешная авторизация:  


Отрисовка игрового поля:



Конец игры:  


**2.2 Особенности реализации приложения**

В программе в качестве структуры данных используется массив, который применяется для отрисовки игрового поля и проверки ходов. Данный подход позволяет легко отслеживать, какие клетки заняты, а какие свободны. Кроме того, двумерный массив упрощает проверку условий победы, так как позволяет эффективно анализировать расположение фигур в различных направлениях.

**3. Обращение к программе**

Функции:  
1) load\_or\_create\_key()

Проверяет существование файла с ключом шифрования. Если файл есть, загружает ключ, если нет — создаёт новый 16-байтовый ключ AES-128, сохраняет его в файл и возвращает.

2) encrypt\_password(password)

Шифрует пароль с использованием алгоритма AES. Она генерирует случайный ключ, шифрует пароль и возвращает результат в формате IV:зашифрованный\_пароль.

3) decrypt\_password(encrypted\_password)

Расшифровывает пароль с помощью AES

4) load\_users()

Загружает список пользователей из файла, если файл существует. Если файл не найден, она возвращает пустой список.

5) save\_users(users)

Сохраняет пользователя в файл

6) register()

Обрабатывает процесс регистрации пользователя. Она проверяет корректность введённых данных, такие как логин и пароль, и убеждается, что они соответствуют определённым условиям. Если данные корректны и логин уникален, функция сохраняет информацию о пользователе и сообщает о успешной регистрации.

7) login()

Осуществляет процесс авторизации пользователя. Она проверяет, чтобы все поля были заполнены, затем ищет пользователя в базе данных. Если логин найден, она проверяет правильность пароля, используя функцию для расшифровки. При успешной авторизации запускается игра, в противном случае выводится сообщение об ошибке.

Класс ChessGame (Служит для работы с игровой логикой и обработке различных событий игрового характера)

1) \_\_init\_\_(self, master)

Конструктор метода.

2) init\_board(self)

Создает шахматную доску, устанавливает ее размер, рисует клетки с чередующимися цветами и добавляет координаты по краям доски. Клетки рисуются на канвасе, и для каждой строки и столбца добавляются соответствующие текстовые метки (цифры для строк и буквы для столбцов).

3) get\_random\_empty\_position(for\_pawn=False, is\_white=False, is\_king=False)

Выбирает случайную пустую клетку на доске, с учетом некоторых условий. Она проверяет, чтобы выбранная позиция была свободной, и в зависимости от переданных флагов, учитывает дополнительные ограничения, такие как запрет на размещение пешек в первых или последних рядах, или ограничение для королей на определенные строки доски.

4) draw\_pieces(self)

Отвечает за отрисовку фигур на доске. Она удаляет все текущие изображения фигур на холсте, затем загружает изображения для разных типов фигур (король, ферзь, пешка) для черных и белых. Для каждой клетки доски, в зависимости от содержимого, она отображает соответствующую фигуру на холсте с помощью изображений.

5) is\_king\_captured(self, player)

Проверяет, захвачен ли король указанного игрока

6) on\_click(self, event)

Обрабатываем клик по доске

7) move\_piece(self, from\_pos, to\_pos)

Перемещает фигуру и сразу проверяет победу

8) is\_valid\_move(self, piece, from\_row, from\_col, to\_row, to\_col)

Метод проверяет, является ли ход допустимым для выбранной фигуры. Он учитывает тип фигуры, ее правила движения и наличие преград на пути (например, других фигур).

9) reset\_game(self)

Сбрасывает игру, создавая новую доску

10) show\_victory\_message(self, message)

Показывает всплывающее окно с сообщением о победе и перезапускает игру

11) check\_victory(self)

Метод проверяет, есть ли на доске короли обоих игроков. Если одного из королей нет, игра заканчивается, и объявляется победитель.

12) get\_all\_valid\_moves(self, piece, row, col)

Возвращает все допустимые ходы для данной фигуры

13) is\_check(self, color)

Проверяет, находится ли король на текущем цвете под шахом.

14) find\_king(self, color)

Находит позицию короля на поле для данного цвета.

15) is\_safe\_move(self, row, col)

Проверяет, безопасен ли ход на данную клетку для короля.

16) is\_under\_attack(self, row, col)

Проверяет, атакует ли клетку вражеская фигура.

17) king\_escape(self)

Метод проверяет, может ли король избежать шаха, передвигаясь на безопасные клетки. Если существует безопасный ход, король перемещается, и проверяется, остаётся ли он под шахом. Если удаётся избежать шаха, функция возвращает True, иначе — False.

18) simulate\_move(self, from\_row, from\_col, to\_row, to\_col)

Временный ход для проверки безопасности клетки

19) undo\_move(self, from\_row, from\_col, to\_row, to\_col)

Откат хода после проверки безопасности клетки

20) ai\_move(self)

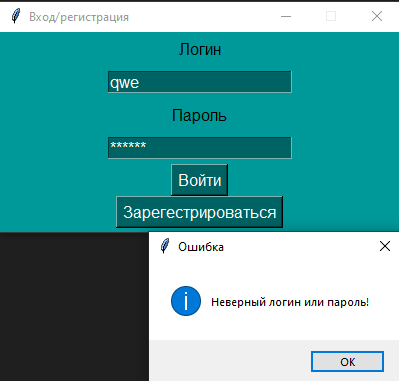
Метод управляет ходами ИИ, при этом она делает выбор по определенным приоритетам. Сначала он проверяет, можно ли сразу взять короля противника. Если король под шахом, ИИ пытается найти безопасный ход или ход с захватом. Далее ИИ проверяет, может ли он атаковать фигуры противника, делает ход ферзя или продвигает пешку, если это возможно. После этого ИИ выбирает обычный ход для других фигур. Ход выполняется в зависимости от приоритета, начиная с самых критичных ситуаций, таких как захват короля.

21) start\_game()

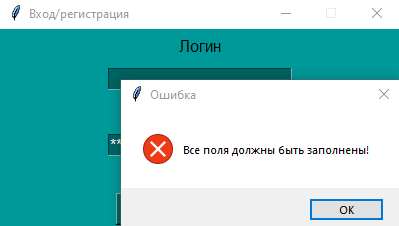
Запуск игры

**4. Сообщения**

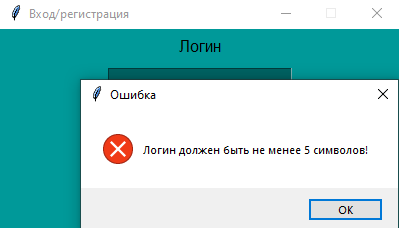
Пользователь не найден или введен неправильный пароль:



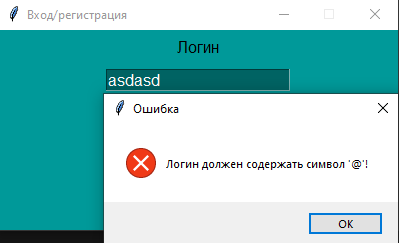
Какое-либо из полей было не заполнено:



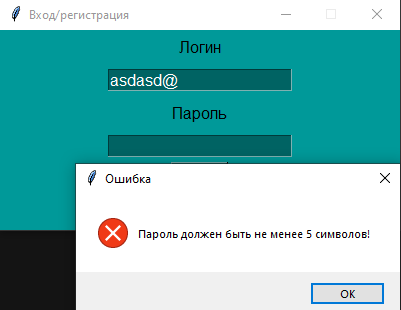
Ограничение на минимальную длину логина:



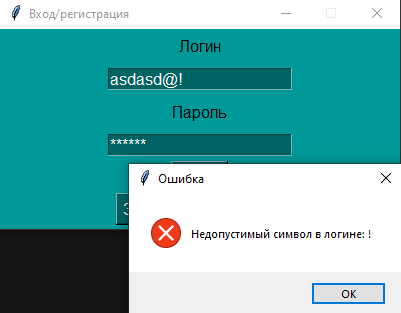
Ограничение требующее наличие знака “@” в логине:

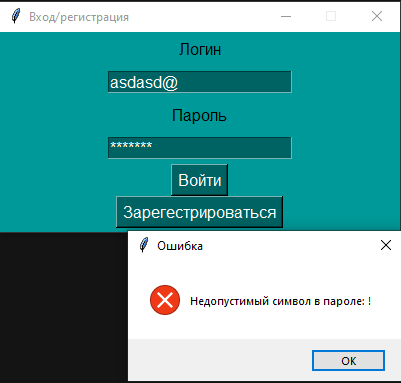


Ограничение на минимальную длину пароля:

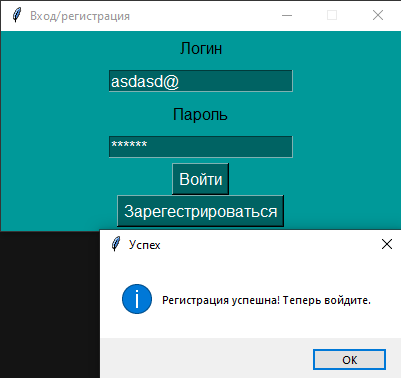


Ограничение на недопустимые символы в логине:

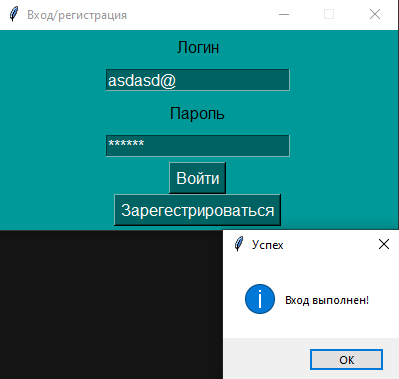


Ограничение на недопустимые символы в пароле:  


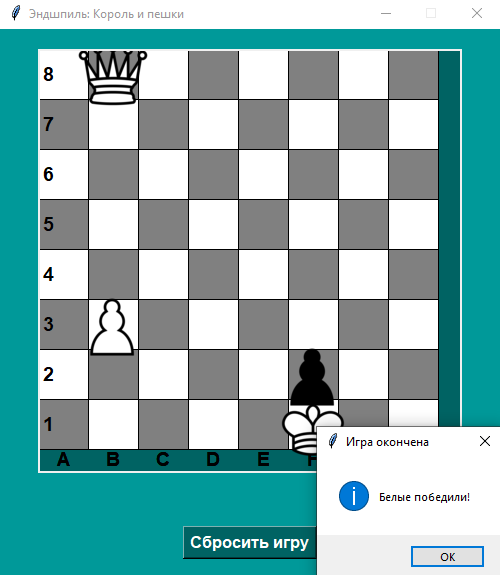
Сообщение об успешной регистрации:



Сообщение об успешной авторизации:



Сообщение об победе определенной стороны:



# **Текст программы:**

**import** tkinter **as** tk

**from** tkinter **import** messagebox

**import** json

**import** os

**import** random

**from** Crypto.Cipher **import** AES

**from** Crypto.Util.Padding **import** pad, unpad

**from** Crypto.Random **import** get\_random\_bytes

**import** base64

​

​

​

# ===================== Функции работы с пользователями =====================

KEY\_FILE = "encryption\_key.key" # Файл для хранения ключа шифрования

DB\_FILE = 'users.json' # Файл для хранения данных пользователей

incorrect = ('!@#$%^&\*+\_-=|/?><~`[]±§')

# Проверяем, существует ли ключ и загружаем его, если нет — создаём новый

**def** load\_or\_create\_key():

**if** os.path.exists(KEY\_FILE):

**with** open(KEY\_FILE, "rb") **as** f:

**return** f.read()

**else**:

key = get\_random\_bytes(16) # 16 байт для AES-128

**with** open(KEY\_FILE, "wb") **as** f:

f.write(key)

**return** key

​

KEY = load\_or\_create\_key()

​

# Шифруем пароль с помощью AES

**def** encrypt\_password(password):

cipher = AES.new(KEY, AES.MODE\_CBC)

ct\_bytes = cipher.encrypt(pad(password.encode(), AES.block\_size)) # шифруем пароль

iv = base64.b64encode(cipher.iv).decode('utf-8')

ct = base64.b64encode(ct\_bytes).decode('utf-8')

**return** iv + ":" + ct

​

# Расшифровываем пароль с помощью AES

**def** decrypt\_password(encrypted\_password):

**try**:

iv, ct = encrypted\_password.split(":")

**except** ValueError:

**raise** ValueError("Неверный формат зашифрованного пароля.")

iv = base64.b64decode(iv)

ct = base64.b64decode(ct)

cipher = AES.new(KEY, AES.MODE\_CBC, iv)

pt = unpad(cipher.decrypt(ct), AES.block\_size)

**return** pt.decode('utf-8')

​

# Загружаем пользователей из файла

**def** load\_users():

"""Загружаем пользователей как список словарей"""

**if** os.path.exists(DB\_FILE):

**with** open(DB\_FILE, "r") **as** f:

**return** json.load(f)

**return** []

​

# Сохраняем пользователей в файл

**def** save\_users(users):

**with** open(DB\_FILE, "w") **as** f:

json.dump(users, f)

​

# ===================== Функции регистрации и авторизации =====================

​

**def** register():

"""Регистрация пользователя с проверками"""

username = entry\_username.get()

password = entry\_password.get()

**if** len(username) < 5:

messagebox.showerror("Ошибка", "Логин должен быть не менее 5 символов!")

**return**

**if** '@' **not** **in** username:

messagebox.showerror("Ошибка", "Логин должен содержать символ '@'!")

**return**

**if** len(password) < 5:

messagebox.showerror("Ошибка", "Пароль должен быть не менее 5 символов!")

**return**

**for** char **in** username:

**if** char **in** incorrect **and** char != '@':

messagebox.showerror("Ошибка", f"Недопустимый символ в логине: {char}")

**return**

**for** char **in** password:

**if** char **in** incorrect:

messagebox.showerror("Ошибка", f"Недопустимый символ в пароле: {char}")

**return**

**if** '@' **in** password:

messagebox.showerror("Ошибка", "Пароль не должен содержать символ '@'!")

**return**

users = load\_users()

**if** any(user['login'] == username **for** user **in** users):

messagebox.showerror("Ошибка", "Пользователь с таким логином уже существует!")

**return**

encrypted\_password = encrypt\_password(password)

users.append({'login': username, 'password': encrypted\_password})

save\_users(users)

messagebox.showinfo("Успех", "Регистрация успешна! Теперь войдите.")

​

**def** login():

"""Авторизация пользователя с проверками"""

username = entry\_username.get()

password = entry\_password.get()

**if** **not** username **or** **not** password:

messagebox.showerror("Ошибка", "Все поля должны быть заполнены!")

**return**

users = load\_users()

​

**for** user **in** users:

**if** user['login'] == username:

**try**:

**if** decrypt\_password(user['password']) == password:

messagebox.showinfo("Успех", "Вход выполнен!")

root.destroy()

start\_game()

**break**

**except** ValueError:

messagebox.showinfo("Ошибка", "Неверный формат зашифрованного пароля.")

**break**

**else**:

messagebox.showinfo("Ошибка", "Неверный логин или пароль!")

​

# ===================== Игровая логика =====================

​

**class** ChessGame:

**def** \_\_init\_\_(self, master):

self.game\_over = False

self.master = master

self.master.title("Эндшпиль: Король и пешки")

self.master.geometry("500x550")

self.master.resizable(width=False, height=False)

self.master.configure(bg='#009999')

​

self.canvas = tk.Canvas(master, width=400, height=500, bg='#006363')

self.canvas.pack(side=tk.TOP, padx=20, pady=20)

​

self.reset\_button = tk.Button(master, text="Сбросить игру", command=self.reset\_game,

bg='#006363', fg='white', font=("Arial", 12, "bold"))

self.reset\_button.pack(side=tk.BOTTOM, padx=20, pady=20)

​

self.board = [[None **for** \_ **in** range(8)] **for** \_ **in** range(8)]

self.selected\_piece = None

self.init\_board()

​

self.canvas.bind("<Button-1>", self.on\_click)

​

​

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

"""Создаем шахматную доску и добавляем координаты"""

cell\_size = 50 # Размер клетки

board\_size = 8 \* cell\_size # Размер доски

offset = 20 # Отступ для координат

​

self.canvas.config(width=board\_size + offset, height=board\_size + offset)

​

colors = ["white", "gray"]

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

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

color = colors[(row + col) % 2]

self.canvas.create\_rectangle(

col \* cell\_size, row \* cell\_size,

(col + 1) \* cell\_size, (row + 1) \* cell\_size,

fill=color

)

​

#координаты (цифры 1-8)

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

self.canvas.create\_text(

offset // 2, row \* cell\_size + cell\_size // 2,

text=str(8 - row), font=("Arial", 14, "bold"),

anchor="center"

)

​

#координаты (буквы A-H)

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

self.canvas.create\_text(

col \* cell\_size + cell\_size // 2, board\_size + offset // 2,

text=chr(65 + col), font=("Arial", 14, "bold"),

anchor="center"

)

​

positions = set()

​

**def** get\_random\_empty\_position(for\_pawn=False, is\_white=False, is\_king=False):

"""Выбирает случайную пустую клетку на доске"""

**while** True:

row = random.randint(0, 7)

col = random.randint(0, 7)

​

**if** for\_pawn:

**if** row == 0 **or** row == 7 **or** row == 1 **or** row == 6:

**continue**

​

**if** is\_king:

**if** is\_white:

**if** row < 4:

**continue**

**else**:

**if** row > 3:

**continue**

​

**if** (row, col) **not** **in** positions:

positions.add((row, col))

**return** row, col

​

self.wk\_pos = get\_random\_empty\_position(is\_king=True, is\_white=True) # Белый король

​

self.bk\_pos = get\_random\_empty\_position(is\_king=True, is\_white=False) # Черный король

​

king\_row, king\_col = self.bk\_pos

attack\_positions = [

(king\_row - 1, king\_col - 1), (king\_row - 1, king\_col), (king\_row - 1, king\_col + 1),

(king\_row, king\_col - 1), (king\_row, king\_col + 1),

(king\_row + 1, king\_col - 1), (king\_row + 1, king\_col), (king\_row + 1, king\_col + 1)

]

​

attack\_positions = [

(r, c) **for** r, c **in** attack\_positions **if** 0 <= r < 8 **and** 0 <= c < 8

]

​

self.wp1\_pos = get\_random\_empty\_position(for\_pawn=True, is\_white=True)

**while** self.wp1\_pos **in** attack\_positions:

self.wp1\_pos = get\_random\_empty\_position(for\_pawn=True, is\_white=True)

​

self.wp2\_pos = get\_random\_empty\_position(for\_pawn=True, is\_white=True)

**while** self.wp2\_pos **in** attack\_positions:

self.wp2\_pos = get\_random\_empty\_position(for\_pawn=True, is\_white=True)

​

self.wk\_pos = get\_random\_empty\_position(is\_king=True, is\_white=True)

**while** self.wk\_pos **in** attack\_positions:

self.wk\_pos = get\_random\_empty\_position(is\_king=True, is\_white=True)

​

self.bp\_pos = get\_random\_empty\_position(for\_pawn=True, is\_white=False)

​

self.board[self.wk\_pos[0]][self.wk\_pos[1]] = "K"

self.board[self.wp1\_pos[0]][self.wp1\_pos[1]] = "P"

self.board[self.wp2\_pos[0]][self.wp2\_pos[1]] = "P"

self.board[self.bk\_pos[0]][self.bk\_pos[1]] = "k"

self.board[self.bp\_pos[0]][self.bp\_pos[1]] = "p"

​

self.draw\_pieces()

​

​

**def** draw\_pieces(self):

"""Рисуем фигуры на доске с использованием изображений"""

self.canvas.delete("piece")

​

self.images = {

"wK": tk.PhotoImage(file="images/w\_king.png"),

"bK": tk.PhotoImage(file="images/b\_king.png"),

"wQ": tk.PhotoImage(file="images/w\_queen.png"),

"bQ": tk.PhotoImage(file="images/b\_queen.png"),

"wP": tk.PhotoImage(file="images/w\_pawn.png"),

"bP": tk.PhotoImage(file="images/b\_pawn.png")

}

​

cell\_size = 50

​

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

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

piece = self.board[row][col]

**if** piece:

img = None

**if** piece == "K":

img = self.images["wK"]

**elif** piece == "k":

img = self.images["bK"]

**elif** piece == "Q":

img = self.images["wQ"]

**elif** piece == "q":

img = self.images["bQ"]

**elif** piece == "P":

img = self.images["wP"]

**elif** piece == "p":

img = self.images["bP"]

​

**if** img:

x = col \* cell\_size + cell\_size // 2

y = row \* cell\_size + cell\_size // 2

self.canvas.create\_image(x, y, image=img, tags="piece")

​

​

**def** is\_king\_captured(self, player):

"""Проверяет, захвачен ли король указанного игрока"""

**if** player == 1:

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

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

**if** self.board[row][col] == 'K':

**return** False

**elif** player == 2:

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

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

**if** self.board[row][col] == 'k':

**print**("Король белых на доске")

**return** False

**print**("Король захвачен!")

**return** True

​

​

**def** on\_click(self, event):

"""Обрабатываем клик по доске"""

**if** self.game\_over:

**return**

​

col = event.x // 50

row = event.y // 50

clicked\_piece = self.board[row][col]

​

**if** self.selected\_piece:

from\_row, from\_col = self.selected\_piece

​

**if** clicked\_piece **and** clicked\_piece.isupper():

self.selected\_piece = (row, col)

**return**

​

**if** **not** self.is\_valid\_move(self.board[from\_row][from\_col], from\_row, from\_col, row, col):

self.selected\_piece = None

**return**

​

self.move\_piece(self.selected\_piece, (row, col))

self.selected\_piece = None

​

# Проверяем, не потерял ли игрок своего короля

**if** self.is\_king\_captured(1):

self.game\_over = True

self.winner\_label.config(text="Черные победили!")

self.restart\_button.pack()

self.resize\_and\_center(640, 690)

**return**

​

self.ai\_move()

​

**elif** clicked\_piece **and** clicked\_piece.isupper():

self.selected\_piece = (row, col)

​

​

**def** move\_piece(self, from\_pos, to\_pos):

"""Перемещает фигуру и сразу проверяет победу"""

from\_row, from\_col = from\_pos

to\_row, to\_col = to\_pos

piece = self.board[from\_row][from\_col]

​

**if** self.is\_valid\_move(piece, from\_row, from\_col, to\_row, to\_col):

self.board[to\_row][to\_col] = piece

self.board[from\_row][from\_col] = None

**if** piece == "P" **and** to\_row == 0:

self.board[to\_row][to\_col] = "Q"

**elif** piece == "p" **and** to\_row == 7:

self.board[to\_row][to\_col] = "q"

self.draw\_pieces()

​

**if** self.check\_victory():

**return**

​

​

**def** is\_valid\_move(self, piece, from\_row, from\_col, to\_row, to\_col):

"""Проверяет, допустим ли ход"""

target\_piece = self.board[to\_row][to\_col]

​

# Нельзя ходить на свою фигуру

**if** target\_piece **and** (piece.isupper() == target\_piece.isupper()):

**return** False

​

# Король – ходит на 1 клетку во всех направлениях

**if** piece **in** ("K", "k"):

**return** abs(from\_row - to\_row) <= 1 **and** abs(from\_col - to\_col) <= 1

​

# Белая пешка – ходит вперед, бьет по диагонали

**if** piece == "P":

**if** from\_col == to\_col **and** from\_row - to\_row == 1 **and** target\_piece **is** None:

**return** True

​

**if** abs(from\_col - to\_col) == 1 **and** from\_row - to\_row == 1 **and** target\_piece **and** target\_piece.islower():

**return** True

​

# Черная пешка

**if** piece == "p":

**if** from\_col == to\_col **and** to\_row - from\_row == 1 **and** target\_piece **is** None:

**return** True

​

**if** abs(from\_col - to\_col) == 1 **and** to\_row - from\_row == 1 **and** target\_piece **and** target\_piece.isupper():

**return** True

​

# Ферзь – ходит как король, но по вертикали, горизонтали и диагоналям на любое количество клеток

**if** piece **in** ("Q", "q"):

**if** from\_row == to\_row:

**for** col **in** range(min(from\_col, to\_col) + 1, max(from\_col, to\_col)):

**if** self.board[from\_row][col] **is** **not** None:

**return** False

**return** True

**elif** from\_col == to\_col:

**for** row **in** range(min(from\_row, to\_row) + 1, max(from\_row, to\_row)):

**if** self.board[row][from\_col] **is** **not** None:

**return** False

**return** True

**elif** abs(from\_row - to\_row) == abs(from\_col - to\_col):

row\_step = 1 **if** to\_row > from\_row **else** -1

col\_step = 1 **if** to\_col > from\_col **else** -1

row, col = from\_row + row\_step, from\_col + col\_step

**while** row != to\_row **and** col != to\_col:

**if** self.board[row][col] **is** **not** None:

**return** False

row += row\_step

col += col\_step

**return** True

​

**return** False

​

​

**def** reset\_game(self):

"""Сбрасывает игру, создавая новую доску"""

self.board = [[None **for** \_ **in** range(8)] **for** \_ **in** range(8)]

self.selected\_piece = None

self.game\_over = False

self.init\_board()

​

**def** show\_victory\_message(self, message):

"""Показывает всплывающее окно с сообщением о победе и перезапускает игру"""

self.game\_over = True

messagebox.showinfo("Победа!", message)

self.reset\_game()

**def** check\_victory(self):

"""Проверяет, остался ли на доске хотя бы один король"""

white\_king\_alive = False

black\_king\_alive = False

​

**for** row **in** self.board:

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

**if** piece == "K":

white\_king\_alive = True

**elif** piece == "k":

black\_king\_alive = True

​

**if** **not** white\_king\_alive:

messagebox.showinfo("Игра окончена", "Черные победили!")

self.game\_over = True

self.reset\_game()

**return** True

**elif** **not** black\_king\_alive:

messagebox.showinfo("Игра окончена", "Белые победили!")

self.game\_over = True

self.reset\_game()

**return** True

​

**return** False

**def** get\_all\_valid\_moves(self, piece, row, col):

"""Возвращает все допустимые ходы для данной фигуры"""

valid\_moves = []

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

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

**if** self.is\_valid\_move(piece, row, col, r, c):

valid\_moves.append((r, c))

**return** valid\_moves

**def** is\_check(self, color):

"""Проверяет, находится ли король на текущем цвете под шахом."""

king\_pos = self.find\_king(color)

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

**return** False

​

king\_row, king\_col = king\_pos

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

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

piece = self.board[row][col]

**if** piece **and** piece.isupper() **if** color == 'black' **else** piece.islower():

valid\_moves = self.get\_all\_valid\_moves(piece, row, col)

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

to\_row, to\_col = move

**if** to\_row == king\_row **and** to\_col == king\_col:

**return** True

**return** False

**def** find\_king(self, color):

"""Находит позицию короля на поле для данного цвета."""

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

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

piece = self.board[row][col]

**if** (piece == 'k' **and** color == 'black') **or** (piece == 'K' **and** color == 'white'):

**return** row, col

**return** None

​

​

**def** is\_safe\_move(self, row, col):

"""Проверяет, безопасен ли ход на данную клетку для короля, включая атаки со стороны вражеских фигур."""

**if** **not** (0 <= row < 8 **and** 0 <= col < 8):

**return** False

​

target\_piece = self.board[row][col]

**if** target\_piece **and** target\_piece.islower():

**return** False

​

**if** self.is\_under\_attack(row, col):

**return** False

​

**return** True

​

​

**def** is\_under\_attack(self, row, col):

"""Проверяет, атакует ли клетку вражеская фигура."""

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

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

piece = self.board[r][c]

**if** piece **and** piece.isupper():

valid\_moves = self.get\_all\_valid\_moves(piece, r, c)

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

to\_row, to\_col = move

**if** to\_row == row **and** to\_col == col:

**return** True

**return** False

​

​

**def** king\_escape(self):

"""Логика ухода короля от шаха, проверяет, не окажется ли король под шахом после хода и не подставится ли он под рубку от вражеских фигур."""

king\_pos = self.find\_king('black')

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

**return** False

​

king\_row, king\_col = king\_pos

possible\_moves = [

(king\_row - 1, king\_col - 1), (king\_row - 1, king\_col), (king\_row - 1, king\_col + 1),

(king\_row, king\_col - 1), (king\_row, king\_col + 1),

(king\_row + 1, king\_col - 1), (king\_row + 1, king\_col), (king\_row + 1, king\_col + 1)

]

​

**for** row, col **in** possible\_moves:

**if** self.is\_safe\_move(row, col) **and** **not** self.is\_under\_attack(row, col):

self.move\_piece((king\_row, king\_col), (row, col))

**if** **not** self.is\_check('black'):

**return** True

​

self.move\_piece((row, col), (king\_row, king\_col))

​

**return** False

​

**def** simulate\_move(self, from\_row, from\_col, to\_row, to\_col):

"""Временный ход для проверки безопасности"""

self.temp\_piece = self.board[to\_row][to\_col]

self.board[to\_row][to\_col] = self.board[from\_row][from\_col]

self.board[from\_row][from\_col] = None

​

**def** undo\_move(self, from\_row, from\_col, to\_row, to\_col):

"""Откат хода после проверки"""

self.board[from\_row][from\_col] = self.board[to\_row][to\_col]

self.board[to\_row][to\_col] = self.temp\_piece

self.temp\_piece = None

​

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

"""ИИ делает ход по приоритетам"""

ai\_moves = [] # Обычные ходы

pawn\_moves = [] # Ходы пешек

capture\_moves = [] # Обычные взятия

queen\_moves = [] # Ходы ферзя

king\_moves = [] # Ходы короля (избегает шаха)

safe\_king\_moves = [] # Безопасные ходы короля

king\_capture = [] # Самый приоритетный ход – взятие короля

​

# Проверяем, можно ли сразу взять короля

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

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

piece = self.board[row][col]

**if** piece **and** piece.islower():

valid\_moves = self.get\_all\_valid\_moves(piece, row, col)

​

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

to\_row, to\_col = move

target\_piece = self.board[to\_row][to\_col]

​

**if** target\_piece == "K":

king\_capture.append((row, col, to\_row, to\_col, 999))

​

# Если можно сразу взять короля, делаем это

**if** king\_capture:

from\_row, from\_col, to\_row, to\_col, \_ = king\_capture[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**return**

​

# Если король под шахом, ищем безопасный ход или рубку

**if** self.is\_check('black'):

**print**("Король под шахом! Ищем безопасный ход...")

# Проверим, может ли король взять фигуру

king\_row, king\_col = self.find\_king('black')

valid\_king\_moves = self.get\_all\_valid\_moves('k', king\_row, king\_col)

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

to\_row, to\_col = move

target\_piece = self.board[to\_row][to\_col]

**if** target\_piece **and** target\_piece.isupper():

capture\_moves.append((king\_row, king\_col, to\_row, to\_col, 3))

​

**if** capture\_moves:

capture\_moves.sort(key=**lambda** x: x[4], reverse=True)

from\_row, from\_col, to\_row, to\_col, \_ = capture\_moves[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**return**

​

# Если король не может рубить, пытаемся найти безопасные ходы.

**if** self.king\_escape():

**return**

​

king\_under\_check = self.is\_check('black')

​

# Перебираем все фигуры ИИ

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

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

piece = self.board[row][col]

**if** piece **and** piece.islower(): # Черные фигуры

valid\_moves = self.get\_all\_valid\_moves(piece, row, col)

​

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

to\_row, to\_col = move

target\_piece = self.board[to\_row][to\_col]

​

# Взятие короля

**if** target\_piece == "K":

king\_capture.append((row, col, to\_row, to\_col, 999))

​

**if** king\_under\_check **and** piece == "k":

**if** **not** self.is\_under\_attack(to\_row, to\_col):

safe\_king\_moves.append((row, col, to\_row, to\_col, 10))

**continue**

​

# Взятие любой белой фигуры

**if** target\_piece **and** target\_piece.isupper() **and** piece != "k":

capture\_moves.append((row, col, to\_row, to\_col, 3))

​

# Ход ферзя

**elif** piece == "q":

queen\_moves.append((row, col, to\_row, to\_col, 2))

​

# Продвижение пешки

**elif** piece == "p" **and** to\_row == 7:

pawn\_moves.append((row, col, to\_row, to\_col, 2))

**elif** piece == "p" **and** **not** target\_piece:

pawn\_moves.append((row, col, to\_row, to\_col, 1))

​

# Обычный ход короля

**elif** piece == "k" **and** **not** self.is\_under\_attack(to\_row, to\_col):

**if** to\_row **in** (0, 7) **or** to\_col **in** (0, 7):

king\_moves.append((row, col, to\_row, to\_col, 0))

**else**:

king\_moves.append((row, col, to\_row, to\_col, 1))

​

**elif** **not** target\_piece:

ai\_moves.append((row, col, to\_row, to\_col, 1))

​

​

# Если можно рубить фигуру, делаем это

**if** capture\_moves:

capture\_moves.sort(key=**lambda** x: x[4], reverse=True)

from\_row, from\_col, to\_row, to\_col, \_ = capture\_moves[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**return**

​

# Ход ферзя

**if** queen\_moves:

queen\_moves.sort(key=**lambda** x: x[4], reverse=True)

from\_row, from\_col, to\_row, to\_col, \_ = queen\_moves[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**return**

​

# Продвижение пешки

**if** pawn\_moves:

pawn\_moves.sort(key=**lambda** x: x[4], reverse=True)

from\_row, from\_col, to\_row, to\_col, \_ = pawn\_moves[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**return**

​

# Обычный ход короля

**if** king\_moves:

king\_moves.sort(key=**lambda** x: x[4], reverse=True)

from\_row, from\_col, to\_row, to\_col, \_ = king\_moves[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**return**

​

# Обычный ход

**if** ai\_moves:

ai\_moves.sort(key=**lambda** x: x[4], reverse=True)

from\_row, from\_col, to\_row, to\_col, \_ = ai\_moves[0]

self.move\_piece((from\_row, from\_col), (to\_row, to\_col))

**def** start\_game():

"""Запуск игры"""

game\_window = tk.Tk()

ChessGame(game\_window)

game\_window.mainloop()

​

# ===================== Интерфейс авторизации =====================

​

root = tk.Tk()

root.title('Вход/регистрация')

root.geometry("400x200")

root.resizable(width=False, height=False)

root.configure(bg='#009999')

​

# Логин

log\_label = tk.Label(root, bg='#009999', text='Логин', font=('Arial', 12))

log\_label.pack(pady=5)

entry\_username = tk.Entry(root, bg='#006363',fg='white', font=('Arial', 12))

entry\_username.pack(pady=5)

​

# Пароль

password\_label = tk.Label(root, bg='#009999', text='Пароль', font=('Arial', 12))

password\_label.pack(pady=5)

entry\_password = tk.Entry(root, bg='#006363',fg='white', font=('Arial', 12), show="\*")

entry\_password.pack(pady=5)

​

# Кнопка "Вход"

btn\_log = tk.Button(root, text='Войти', bg='#006363', fg='white', font=('Arial', 12), command=login)

btn\_log.pack()

​

# Кнопка "Зарегистрироваться"

btn\_reg = tk.Button(root, text='Зарегестрироваться', bg='#006363', fg='white', font=('Arial', 12), command=register)

btn\_reg.pack()

​

root.mainloop()