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»

Тема Компьютерная логическая игра «Халма»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 54

Руководитель разработки:

Шишкин Вадим

Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Апахова Ксения Денисовна

« » 2024 г.

2024

Содержание

Аннотация……...…………………………………………………………3

Техническое задание……………………………………………………4

Пояснительная записка...……………………………………………....9

Руководство программиста……………………………….…………..18

Тестовая документация………………………………………………..29

Листинг программы……………………………………………………..36

**АННОТАЦИЯ**

Тема курсового проекта: Компьютерная логическая игра «Халма».

Исполнитель: студент гр. ИСТбд-21 Апахова Ксения Денисовна.

Руководитель разработки: Шишкин Вадим Викторинович.

Работа состоит из технического задания, пояснительной записки, руководства программиста и текста программы.

В техническом задании описаны общие правила компьютерной логической игры «Халма», условия выигрыша, проигрыша, начальные позиции фигур. Приведены основания для разработки, функциональное назначение, основные требования к функциональным характеристикам, надежности, информационной и программной совместимости, хранению, транспортировке, программной документации.

В пояснительной записке указываются задачи, математические методы, архитектура и алгоритмы, тестирование и источники, использованные при разработке.

В руководстве программиста приводятся назначение и функции, выполняемые приложением, условия использования, характеристики приложения, особенности реализации приложения, обращение к программе, сообщения, выдаваемые по результатам контроля корректности ввода/вывода.

В тестовой документации описаны тест-кейсы с описанием ожидаемого результата и текущей реализации, описание тестов с использованием mind map.

Текст программы представляет собой полный код программы, реализующий компьютерную логическую игру «Халма».

Ключевые слова: компьютерная игра, шашки, халма, python.

Всего листов 54.

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 логическая игра «Халма»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 23/693 3 ТЗ-02

Листов 5

Исполнитель:

студент гр. ИСТбд-21

Апахова Ксения Денисовна

« » 2024 г.

2024

**Введение**

Разрабатываемое приложение наименования «Халма» реализует игру халма. Халма – вариант игры в "Шашки" на доске 8х8 с 9 шашками у каждого из игроков. Игра предназначена для двух игроков. Игроки поочередно перемещают шашки на игровом поле. Первый игрок использует шашки белого цвета, а второй игрок - черного. Цель – переместить все свои шашки на противоположную сторону доски в исходное расположение соперника.

**1. Основания для разработки**

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

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

**2.1. Функциональное назначение**

**Функциональное назначение**

Требуется разработать однопользовательское десктопное приложение по игре в халму 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 (пользователь-компьютер или пользователь 1 – пользователь 2) шашками разного цвета (белые и чёрные) на квадратном поле, размером 8х8 клеток. У каждого игрока в начальной позиции находится 9 шашек, расположенных квадратом 3х3 в левом нижнем углу (белые) и в правом верхнем (черные).

**Типы ходов.**

Существует 2 типа ходов:

1. Перемещение шашки на 1 свободную клетку вверх, вниз, вправо, влево.
2. «Прыжок» шашкой через занятую шашкой клетку, если за ней есть свободная клетка для посадки. Если после перемещения есть возможность продолжить «прыжки», игрок может совершить любое количество перемещений. «Прыжки» можно осуществлять как через фишки самого игрока, так и через фишки противника.

**Порядок ходов.** Ходы в игре происходят поочередно. Первым ходит игрок с белыми фишками (фишки пользователя), а затем ходит игрок с черными фишками (фишки компьютера или 2 игрока). Во время хода можно перемещать только 1 фишку.

**Цель игры.**

Цель игры заключается в перемещении всех своих фишек в противоположный угол доски, который изначально занимал противник.

**Завершение игры.**

Игра завершается, когда один из игроков перемещает все свои фишки в целевой угол.

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 ;

- интерактивные прием, 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 шашек могут храниться в отдельных графических файлах. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле в зашифрованном виде. Пояснительные информационные сообщения для пользователя должны выводиться внизу игрового поля по ходу игры, либо 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 Pro 20H2.

Версия языка программирования: Python 3.9.

Среда разработки: PyCharm Community Edition 2022.3.

При создании программы используются следующие библиотеки:

Pygame, Sys.

**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/ksushka2/halma\_game.

2.6. 3 Сроки хранения

Срок хранения – до июля 2024 года.

**3. Требования к программной документации**

Программная документация должна содержать:

* Техническое задание;
* Руководство программиста;
* Пояснительная записка;
* Тестовая документация.

**4. Стадии и этапы разработки**

1. Составление технического задания.
2. Разработка архитектуры проекта.
3. Написание кода программы:
   * + - Реализация движка игры.
       - Создание бота.
       - Разработка пользовательского интерфейса.
4. Тестирование и отладка.
5. Оформление документации.
6. Защита проекта.

**5. Порядок контроля и приёмки**

Контроль осуществляется путем тестирования всех частей приложения. Приемка работы включает повторное тестирование и защиту готового проекта.

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»

Тема Компьютерная логическая игра «Халма»

Пояснительная записка

Р.02069337. 23/693 3 ПЗ-02

Листов 9

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Исполнитель:

студент гр. ИСТбд-21

Апахова Ксения Денисовна

« » 2024 г.

2024

**Введение**

Программа представляет из себя однопользовательское десктопное приложение по игре в Халму с графическим интерфейсом в среде Windows.

В приложении реализованы в графическом режиме следующие функции:

* регистрация/авторизация пользователя;
* отрисовка игрового поля;
* взаимодействие с пользователем ;
* интерактивные прием, проверка правильности и отрисовка хода пользователя;
* проверка окончания игры;
* вычисление, проверка правильности и отрисовка хода компьютера;
* информирование пользователя об окончании игры и победителе.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

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

**1.2 Математические методы**

1.2.1 Модель игрового поля (матрица начального расположения шашек):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

1.2.2 Весовая модель поля ( матрица весов, для описания бота):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| 14 | 14 | 14 | 9 | 8 | 7 | 6 | 5 |
| 14 | 14 | 14 | 10 | 9 | 8 | 7 | 6 |
| 14 | 14 | 14 | 11 | 10 | 9 | 8 | 7 |

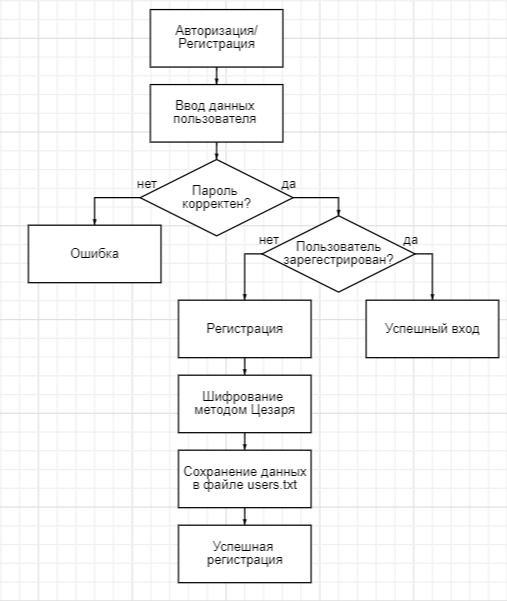
**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура.



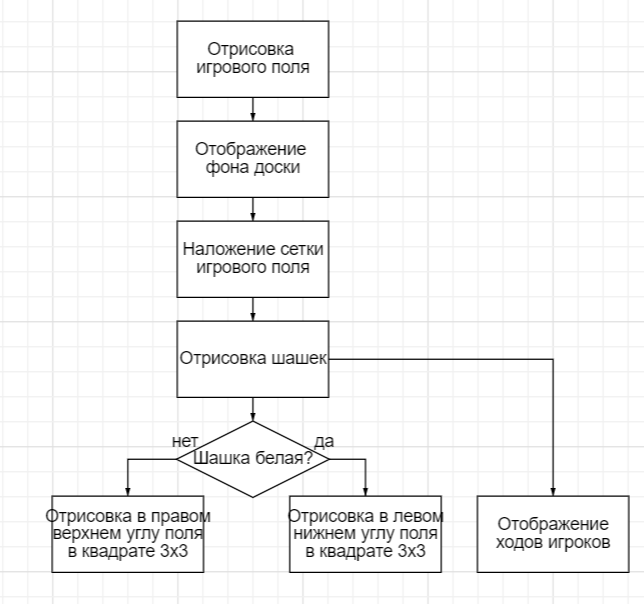
При запуске игры создается окно приложения, загружаются графические ресурсы и запускается система авторизации/регистрации пользователей. После успешной авторизации начинается игровой процесс, который включает в себя несколько ключевых элементов: отрисовку игровой доски, отображение истории ходов игроков и обработку пользовательского ввода. Игровая логика построена на обработке событий, где каждое действие игрока проходит проверку на возможность выполнения хода. Важной частью структуры является бот, реализованный через алгоритм минимакс с альфа-бета отсечением, который управляет ходами черных шашек. Система хранения данных обеспечивает сохранение информации о пользователях в зашифрованном виде.

1.3.2. Алгоритм авторизации/регистрации



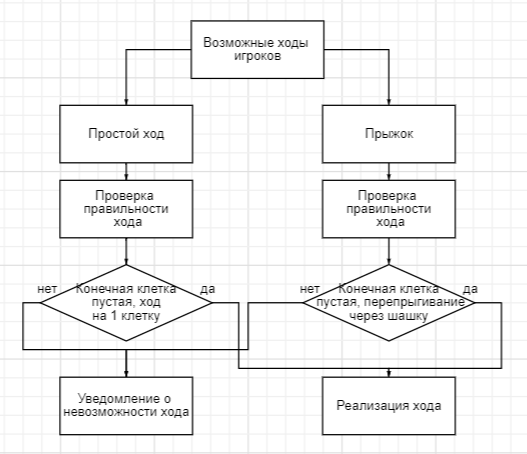
Алгоритм реализует систему авторизации и регистрации пользователей, обеспечивающий персонализацию игрового процесса. Пользователю необходимо ввести свои логин и пароль, после ввода происходит проверка корректности пароля (он должен содержать минимум 3 символа). При корректном пароле алгоритм проверяет, зарегистрирован ли пользователь, путем поиска его учетных данных в файле users.txt. Если пользователь уже зарегистрирован и ввел правильные данные, происходит успешный вход в игру. Если пользователь не найден, алгоритм предлагает зарегистрироваться. При регистрации введенные данные шифруются методом Цезаря, и сохраняются в таком виде в файле пользовательских данных.

1.3.3 Алгоритм визуализации игрового поля.



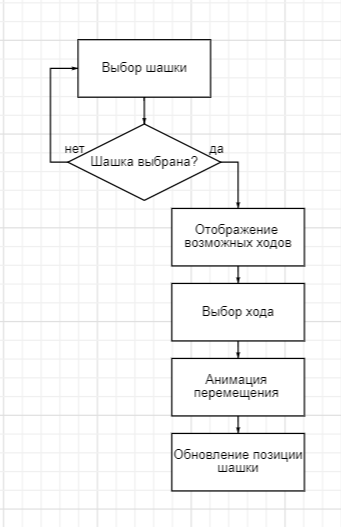
Данный алгоритм отвечает за отрисовку игрового поля. Сначала происходит заполнение экрана цветом, затем накладывается фоновое изображение доски с учетом отступов. После этого создается сетка игрового поля, отрисовка шашек, отображение истории хода каждого игрока.

1.3.4 Алгоритм определение возможности хода.



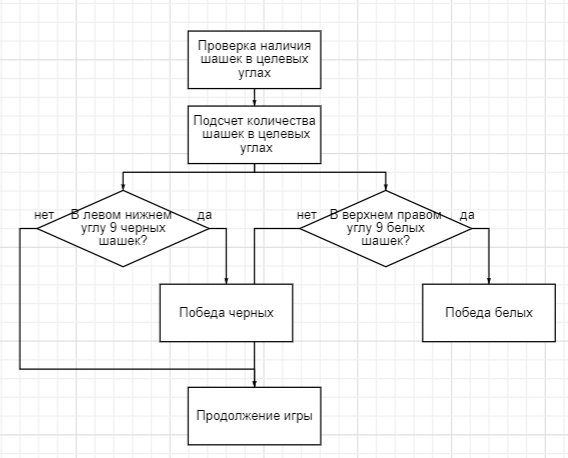
Алгоритм является основным компонентом, определяющим всю игровую механику, анализирует текущее положение шашки на доске и определяет все допустимые варианты перемещения согласно правилам игры, обеспечивает корректную работу как пользовательского интерфейса, так и бота.

1.3.5 Алгоритм отрисовки хода.



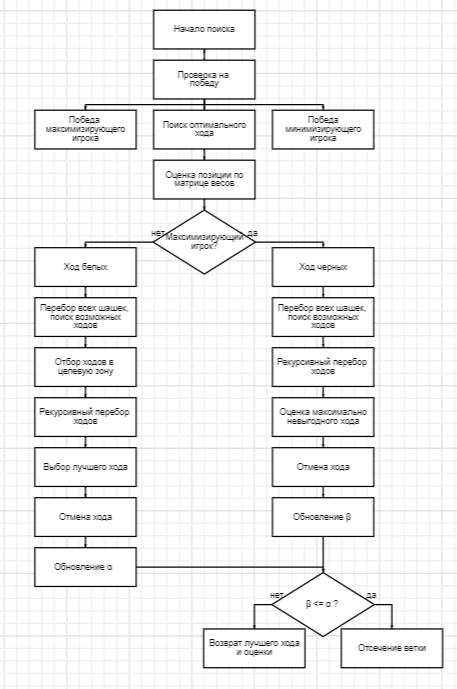
Алгоритм хода шашки осуществляется благодаря «подсветке» возможных ходов, при выборе шашки. После определения конечной позиции происходит анимация перемещения шашки, в конкретную точку.

1.3.6 Алгоритм определения победителя.



Суть алгоритма заключается в проверке достижения шашками противоположных углов доски. Алгоритм определения победителя является ключевым компонентом игры, обеспечивающим корректное завершение партии и правильную работу бота при оценке позиций и принятии решений.

1.3.7 Алгоритм Минимакс с альфа-бета отсечением.



Алгоритм минимакс с альфа-бета отсечением реализован в игре Халма для реализации игры бота. Основная функция алгоритма принимает несколько ключевых параметров: текущее состояние игровой доски, глубину поиска, значения альфа и бета, а также флаг максимизирующего игрока, который определяет, чей ход рассматривается - черных или белых шашек.

При запуске алгоритм первым делом проверяет терминальные состояния игры. Проверяется наличие победителя: если победили черные шашки, возвращается положительная бесконечность, если белые - отрицательная бесконечность. Также проверяется достижение максимальной глубины поиска. При достижении максимальной глубины происходит оценка текущей позиции с использованием специальной весовой матрицы.

Весовая матрица представляет собой двумерный массив значений, где большие веса присваиваются клеткам, находящимся в целевой зоне для черных шашек (нижний левый угол). Это помогает алгоритму оценивать позиции и принимать решения, способствующие достижению цели игры.

Для максимизирующего игрока алгоритм перебирает все возможные ходы, доступные каждой шашке. При этом применяется специальная оптимизация: если шашка находится близко к целевой зоне, то рассматриваются только ходы, которые приближают шашку к цели.

Для минимизирующего игрока алгоритм аналогично перебирает все возможные ходы, но стремится минимизировать оценку позиции, предполагая, что противник будет играть оптимально.

Механизм альфа-бета отсечения работает следующим образом: для максимизирующего игрока происходит обновление значения альфа, выбирая максимум между текущим значением альфа и оценкой позиции. Если значение бета становится меньше или равно альфа, происходит отсечение ветки - дальнейший поиск в этом направлении прекращается, так как он заведомо не приведет к лучшему результату. Аналогично для минимизирующего игрока обновляется значение бета, и происходит отсечение, если бета становится меньше или равно альфа.

**2. Источники, использованные при разработке**

1. Правила игры Халма. https://lotos-khv.ru/game/games/halma.pdf
2. Руководство по Pygame. https://pythonist.ru/pygame-tutorial/
3. Шишкин, В.В. Разработка логических компьютерных игр с графическим интерфейсом в среде питон / В.В. Шишкин, Д.С. Афонин. - Ульяновск: УлГТУ, 2023. – 89 с. – URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://lib.ulstu.ru/venec/disk/2023/112.pdf. – Текст: электронный.

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»

Тема Компьютерная логическая игра «Халма»

Руководство программиста

Р.02069337. 23/693 3 РП-02

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 11

Исполнитель:

студент гр. ИСТбд-21

Апахова Ксения Денисовна

« » 2024 г.

2024

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Однопользовательское десктопное приложение по игре с графическим интерфейсом в среде Windows.

**1.2 Условия, необходимые для использования приложения**

Версия операционной системы: Windows 7, 8, 8.1, 10, 11.

Язык программирования: Python.

Среда разработки: PyCharm Community Edition 2023.1.1.

Используемые библиотеки: графическая библиотека Pygame.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Программа состоит из файла с исходным кодом на языке Python. В качестве ресурсов используются графические файлы **white.png, black.png, desk.jpg, new\_icon.png, music\_fon.mp3**. В процессе работы создаётся текстовый файл для регистрации пользователей **users.txt**. В том числе при сохранении паролей применяется шифрование.

При реализации графического интерфейса программа создаёт основное окно для авторизации/регистрации пользователя. При успешной авторизации пользователь переходит в окно с игровой доской и расставленными шашками в соответствии с правилами шашек Халма. Согласно техническому заданию, игрок играет белыми фигурами и ходит первым. Игровое взаимодействие с пользователем происходит с помощью компьютерной мыши. В программе реализовано несколько окон: *Главное меню, Правила игры, Авторизация, Игра.*

Структура классов:

* + Game: основной класс, управляющий игровым процессом.
  + Board: отвечает за логику игрового поля.
  + Checker: реализует функционал шашек.
  + LoginScreen: управляет системой авторизации.

Взаимодействие с пользователем:

* + Обработка событий мыши реализована в методе **handle\_event,** который отслеживает клики и перемещения мыши.
  + Выбор шашки и определение целевой позиции происходит через методы **select\_checker** и **move\_checker.**

Визуализация игры:

* + Отрисовка игрового поля и всех элементов происходит в методе **draw** класса **Board.**
  + Анимация перемещения шашек реализована в методе **update\_animation** класса **Checker.**
  + Отображение истории ходов и игрового интерфейса выполняется методом **draw\_interface.**
  + Загрузка графических ресурсов (изображения шашек, доски) происходит при инициализации игры.

Игровая логика:

* + Определение возможных ходов реализовано в методах **get\_possible\_moves** и **get\_possible\_jumps.**
  + Проверка победных условий выполняется методом **check\_winner.**
  + Искусственный интеллект для игры за черные шашки реализован через алгоритм минимакс с альфа-бета отсечением и реализован в методе **make\_computer\_move** класса **Board**.
  + Обработка ходов и проверка их корректности выполняется в методе **make\_move.**

**2. 2. Описание интерфейса программы**

1. Окно входа в игру

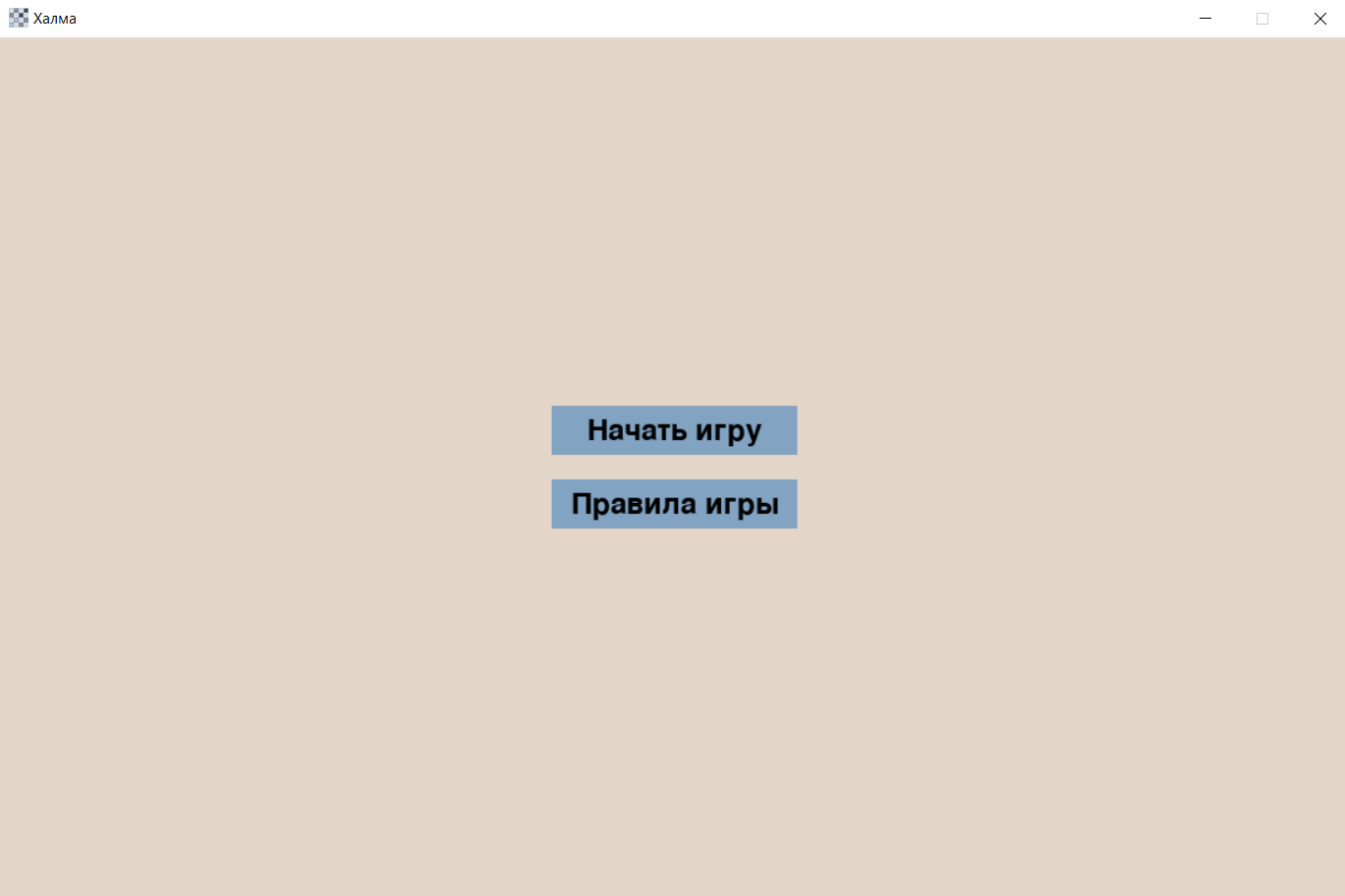


Рисунок Начальный экран

1. Окно правил игры

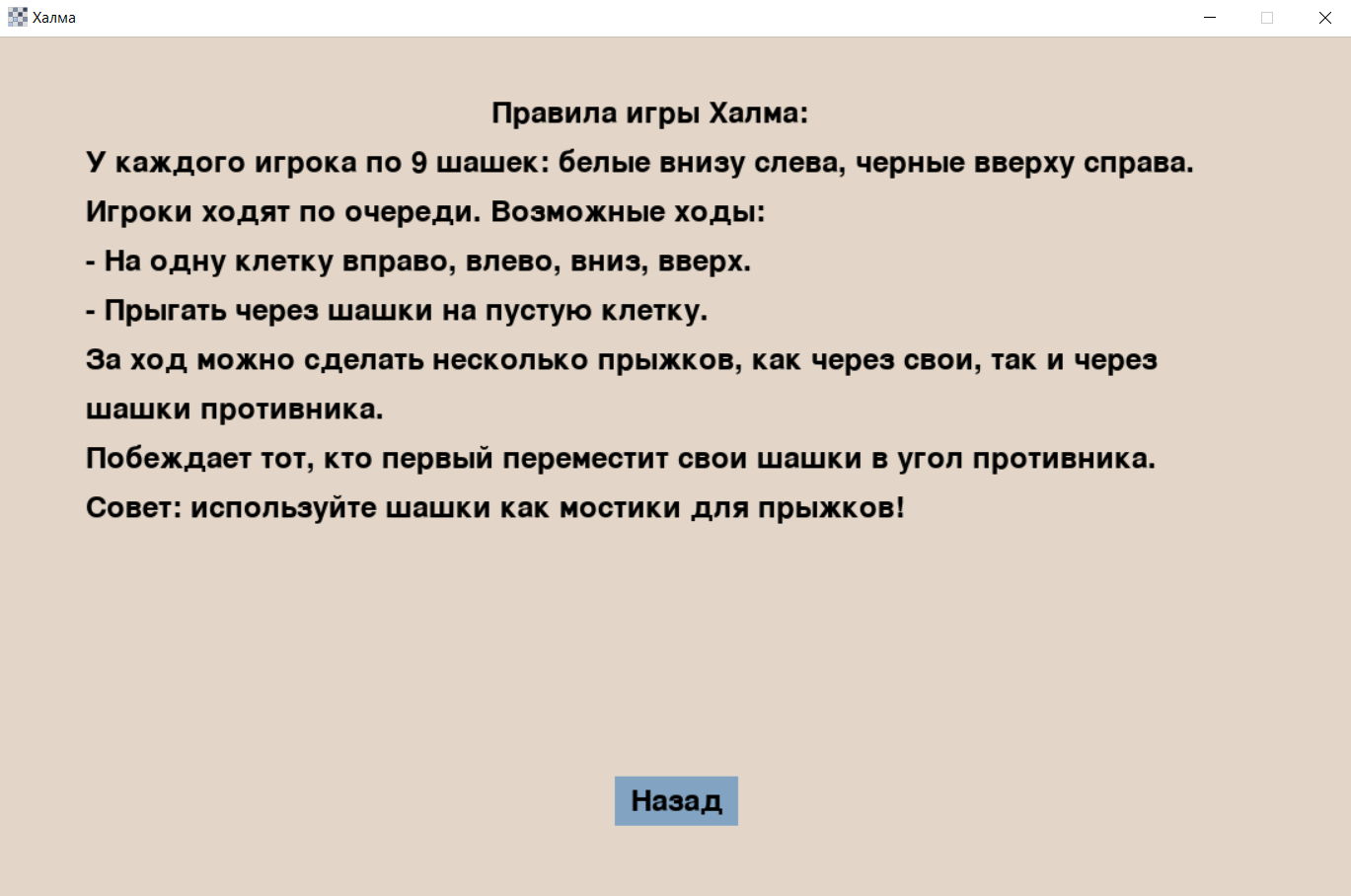


Рисунок Окно "Правила"

3. Окно регистрации и авторизации

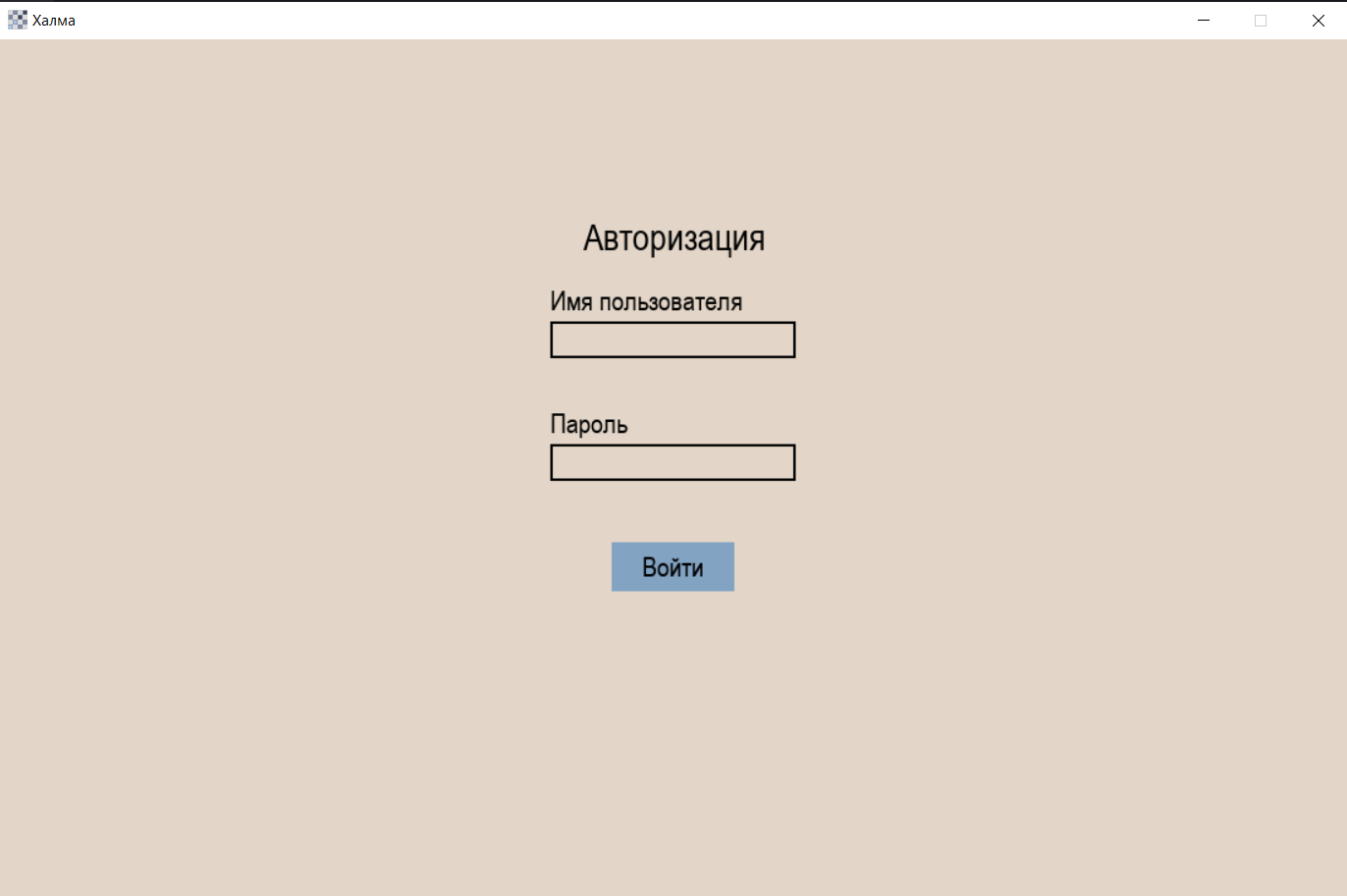


Рисунок Окно авторизации пользователя

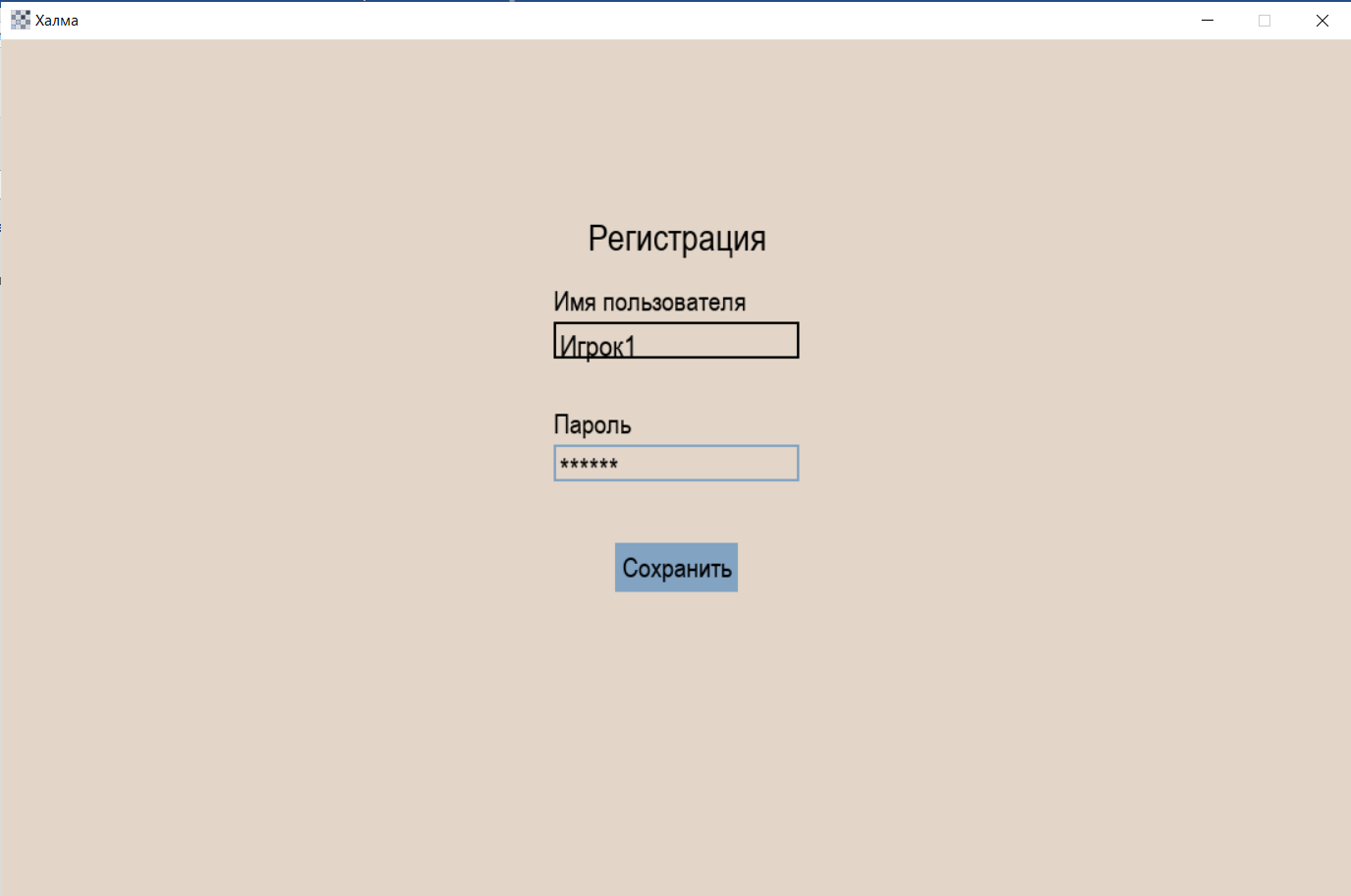


Рисунок Окно регистрации пользователя

Для начала игры необходимо зарегистрироваться или авторизоваться. Если введены не корректные данные, возникнет сообщение об ошибке. После успешной авторизации пользователь переходит в окно Игры.

**2.3. Игра**

После авторизации в новом окне запускается игра: происходит отрисовка доски и начальная расстановка шашек.



Рисунок Окно "Игра"

При наведении мыши на клетку игрового поля, клетка подсвечивается синей рамкой, также подсвечиваются шашки, куда возможно сделать ход.

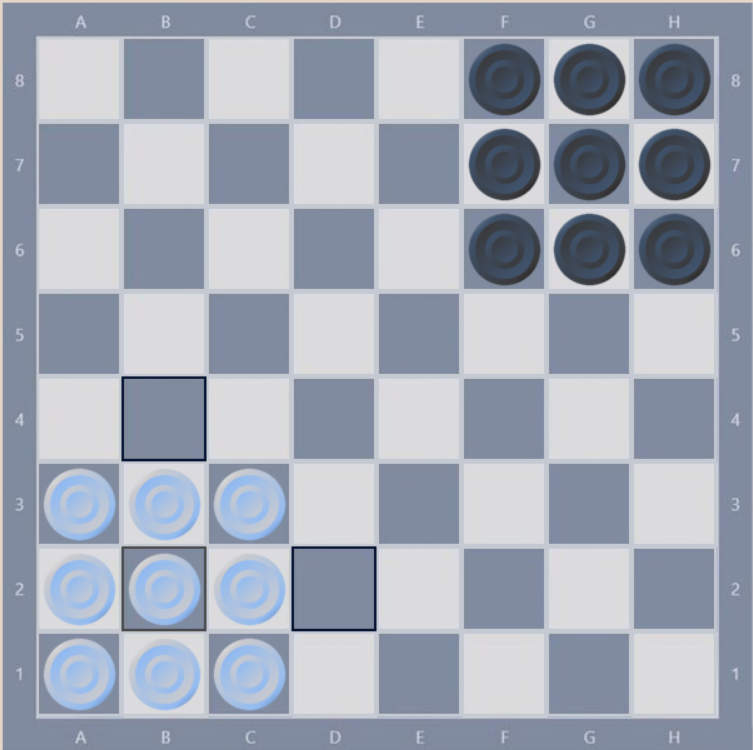


Рисунок Подсветка выбранной шашки и возможных ходов

Если ход выбранной шашкой в заданную клетку возможен, то шашка передвигается и ход передаётся компьютеру. При возможности осуществить множественное «перепрыгивание» ходы также подсвечиваются, шашки через которые перепрыгнули не исчезают.

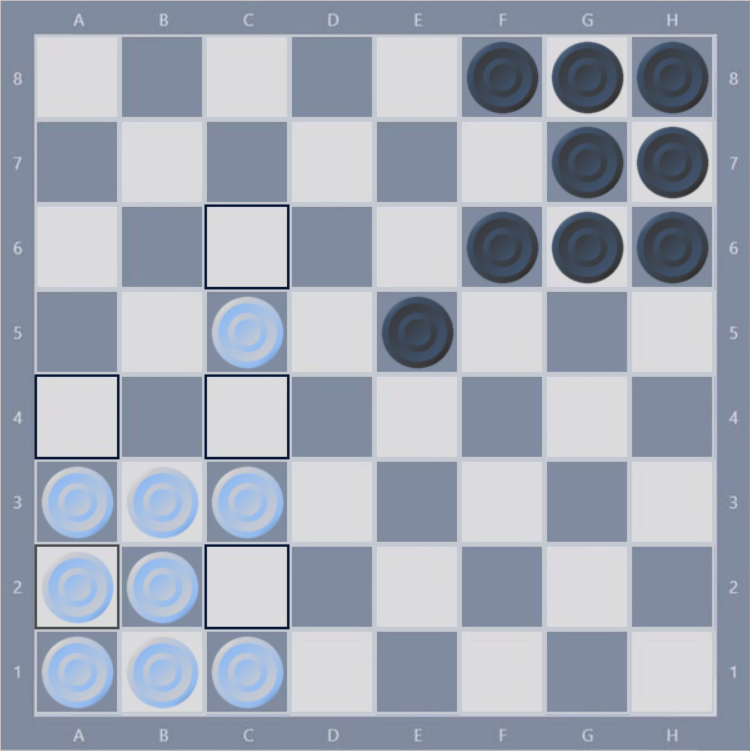


Рисунок Подсветка выбранной шашки и множественных прыжков

Для возможности отслеживать ход игры слева от доски находится поле истории ходов.



Рисунок Игровое поле с отображением истории ходов

При перемещении всех шашек одного из игроков в дом противника игра заканчивается с уведомлением о победе.

**2.4. Средства контроля ввода/вывода:**

1. Проверка авторизации: функция **login**() в классе LoginScreen контролирует минимальную длину пароля (3 символа), проверяет существование пользователя в базе данных и корректность введенных учетных данных. При неверных данных выводится сообщение об ошибке.
2. Проверка шифрования: метод **caesar\_cipher**() обеспечивает корректное шифрование и дешифрование паролей пользователей со сдвигом в 6 позиций.
3. Проверка победителя: метод **check**\_**winner**() в классе Board отслеживает, достигли ли белые шашки верхнего правого угла или черные шашки нижнего левого угла. При достижении условий победы игра завершается с соответствующим сообщением.
4. Проверка возможности хода: функция **can\_jump**() в классе Checker проверяет допустимость прыжка, включая проверку занятости целевой клетки, правильность направления прыжка и наличие шашки для перепрыгивания.
5. Проверка доступных ходов: метод **get\_possible\_moves**() определяет все возможные ходы для выбранной шашки, включая простые перемещения на соседние клетки и прыжки через другие шашки.
6. Проверка хода компьютера: функция **make\_computer\_move**() с алгоритмом минимакс проверяет и оценивает возможные ходы, используя матрицу весов для определения оптимального хода.
7. Проверка анимации: методы **handle\_click**() и **update\_animation**() контролируют возможность выполнения хода во время анимации и следят за корректностью анимации перемещения шашек.
8. Проверка границ: функция **get\_possible\_jumps**() следит, чтобы все ходы оставались в пределах игрового поля 8x8 клеток.
9. Проверка истории ходов: функция **handle\_scroll**() контролирует корректность отображения и прокрутки истории ходов, включая форматирование записей в виде "A1 → B2".
10. Проверка ресурсов: при запуске игры проверяется наличие необходимых изображений (шашки, доска, иконка) и звуковых файлов.

**2.5 Особенности реализации приложения**

Массив поля:

initial\_position = [

[0,0,0,0,0,-1,-1,-1],

[0,0,0,0,0,-1,-1,-1],

[0,0,0,0,0,-1,-1,-1],

[0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0],

[1,1,1,0,0,0,0,0],

[1,1,1,0,0,0,0,0],

[1,1,1,0,0,0,0,0]]

Массив весов поля:

weight\_pole = [

[7, 6, 5, 4, 3, 2, 1, 0],

[8, 7, 6, 5, 4, 3, 2, 1],

[9, 8, 7, 6, 5, 4, 3, 2],

[10, 9, 8, 7, 6, 5, 4, 3],

[11, 10, 9, 8, 7, 6, 5, 4],

[14, 14, 14, 9, 8, 7, 6, 5],

[14, 14, 14, 10, 9, 8, 7, 6],

[14, 14, 14, 11, 10, 9, 8, 7]

]

**3. Обращение к программе**

Pygame - это кросс-платформенная библиотека Python для создания игр и мультимедийных приложений, предоставляющая широкие возможности для работы с графикой, звуком и пользовательским вводом. В данной используются следующие основные компоненты Pygame:

Графический интерфейс:

* + pygame.display - модуль для управления окном и отображением графики.
  + pygame.Rect - класс для работы с прямоугольными областями.
  + pygame.font - модуль для работы с текстом и шрифтами.

Обработка событий:

* + pygame.event - модуль для обработки пользовательского ввода (клики мыши, нажатия клавиш).

Дополнительные элементы:

* + Кнопки и текстовые поля реализованы с помощью pygame.draw.
  + Модуль для воспроизведения фоновой музыки и звуковых эффектов pygame.mixer.

**4. Сообщения**

1) *«Пароль должен содержать минимум 3 символа»* в случае ввода некорректного пароля.

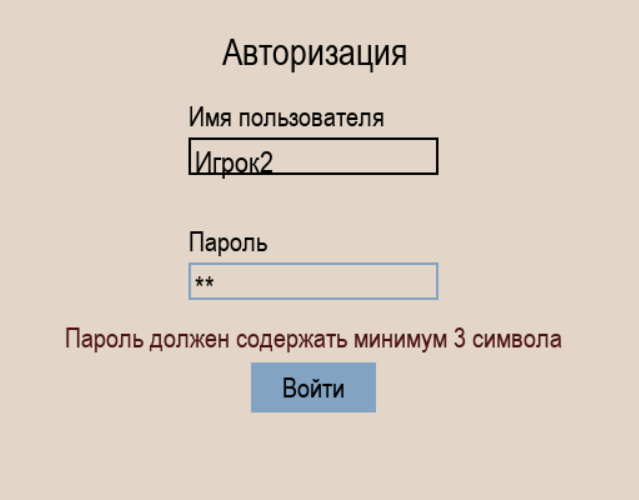


Рисунок Ошибка ввода пароля

2) *«Зарегистрироваться»* в случае отсутствия указанной учётной записи или неверного ввода логина/ пароля.

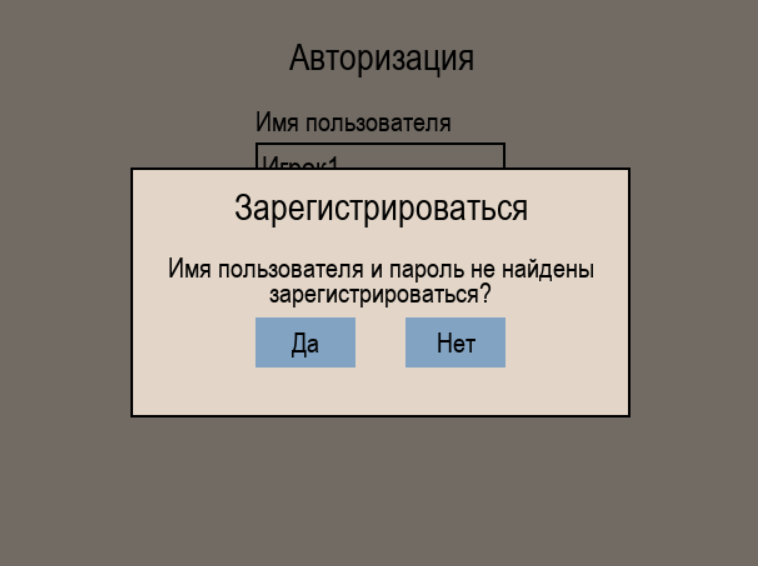


Рисунок Окно предложения регистрации пользователя

3) *«Такой ход сейчас не доступен!»* в случае попытки пользователя сделать ход в недоступную ячейку.



Рисунок Уведомление о невозможном ходе

4) *«Хотите закрыть игру?»* в случае нажатия стрелки выхода.

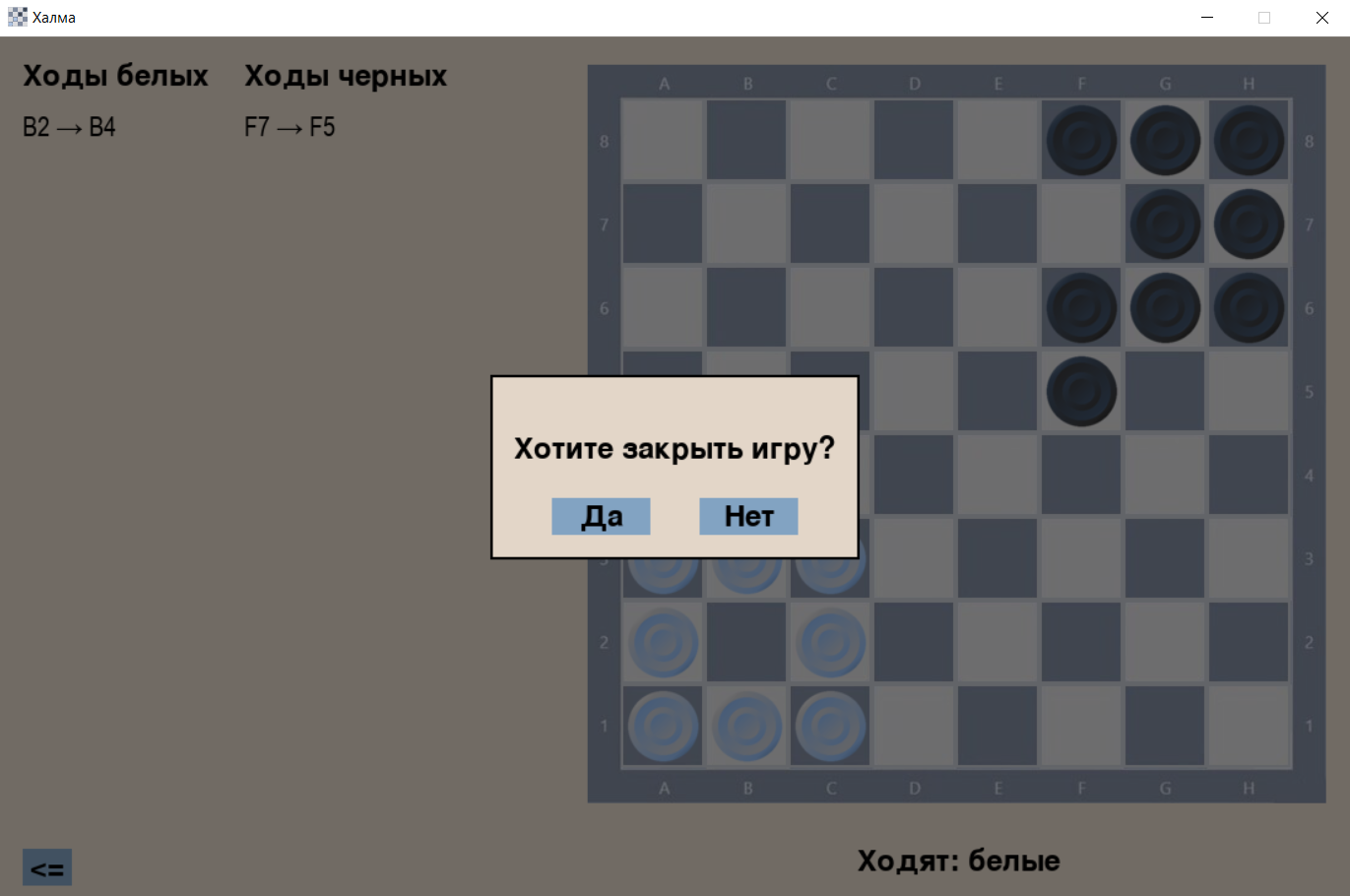


Рисунок Окно закрытия игры до ее окончания

5) *«Победили черные!»* в случае окончании игры при условии проигрыша пользователя.

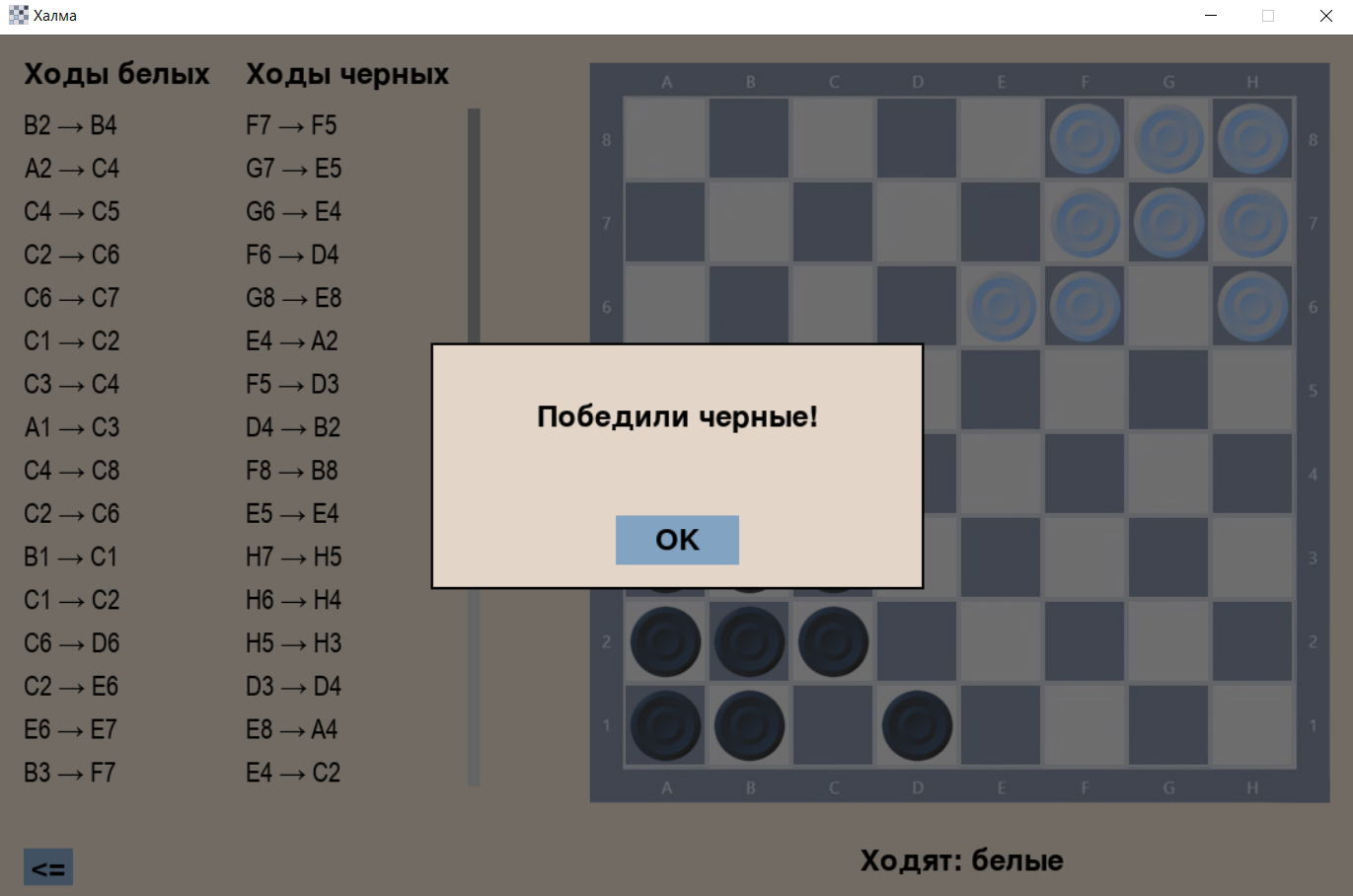


Рисунок Уведомление о победе черных

6) *«Победили белые!»* в случае окончании игры при условии выигрыша пользователя.

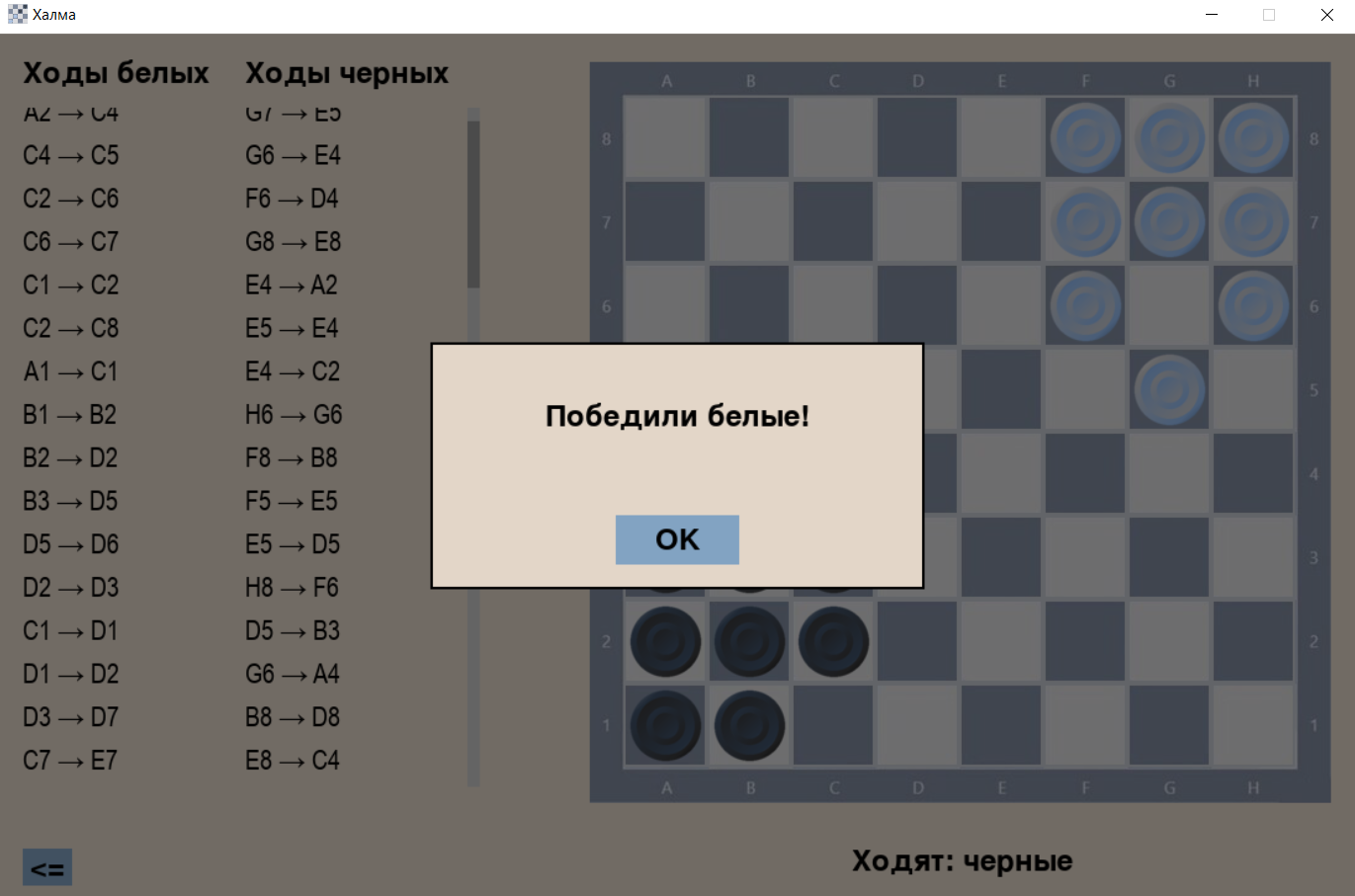


Рисунок Уведомление о победе белых

**5. Компиляция файла в формате Еxe**

Для создания exe файла используется библиотека **PyInstaller. Для использования библиотеки необходимо запустить в Терминале функцию: <**pip install pyinstaller>. После этого нужно выполнить команду: <puinstaller halma.py>. После чего появляется папка dist, в которой будет храниться наш итоговый файл.

Чтобы создать однофайловое приложение необходимо объединить все используемые дополнительные файлы в формате команды Терминала: <pyinstaller --onefile --add-data "desk.jpg;." --add-data "white.png;." --add-data "black.png;." --add-data "new\_icon.png;." --add-data "music\_fon.mp3;." --noconsole halma.py>.

После осуществления этой команды появится готовое однофайловое приложение, работающее в фоновом режиме.

**Тестовая документация**

Для корректности приложения необходимо провести функциональное тестирование компонентов приложения.

Необходимые тест-кейсы:

1. Тесты авторизации и регистрации:
   * Регистрация нового пользователя.

Ожидаемый результат: Программа создает нового пользователя и сохраняет зашифрованные данные в файл users.txt.

Реальный результат: Программа успешно создает пользователя и сохраняет данные с шифрованием методом Цезаря.

* + Авторизация существующего пользователя.

Ожидаемый результат: Программа проверяет учетные данные и предоставляет доступ к игре.

Реальный результат: Программа корректно авторизует пользователя и запускает игру.

* + Попытка регистрации с паролем менее 3 символов.

Ожидаемый результат: Программа отклоняет регистрацию и показывает сообщение об ошибке.

Реальный результат: Программа показывает сообщение "Пароль должен содержать минимум 3 символа" и не допускает начать игру.

* + Проверка шифрования паролей.

Ожидаемый результат: Программа сохраняет пароли в зашифрованном виде методом Цезаря со сдвигом 6.

Реальный результат: Программа корректно шифрует и сохраняет пароли в файле users.txt.

1. Тесты игрового поля:
   * Корректность начальной расстановки шашек.

Ожидаемый результат: Программа размещает 9 белых шашек внизу слева и 9 черных вверху справа квадратом 3х3 клетки.

Реальный результат: Программа корректно расставляет все шашки на начальные позиции.

* + Проверка размеров доски.

Ожидаемый результат: Программа создает доску 8x8 клеток с размером клетки 68 пикселей.

Реальный результат: Программа отображает доску с корректными размерами.

* + Проверка корректности отображения белых и черных шашек.

Ожидаемый результат: Программа отображает белые и черные шашки с соответствующими изображениями.

Реальный результат: Программа корректно отображает шашки при наличии файлов изображений.

* + Проверка подсветки выбранной шашки.

Ожидаемый результат: Программа подсвечивает выбранную шашку синим цветом.

Реальный результат: Программа обводит контур клетки, в которой находится выбранная шашка.

* + Проверка отображения возможных ходов.

Ожидаемый результат: Программа подсвечивает доступные ходы темно-синим цветом.

Реальный результат: Программа корректно отображает все возможные ходы.

1. Тесты механики движения:
   * Простой ход на одну клетку.

Ожидаемый результат: Программа позволяет перемещать шашку на соседнюю клетку в любом направлении.

Реальный результат: Программа корректно обрабатывает простые ходы.

* + Прыжок через одну шашку.

Ожидаемый результат: Программа позволяет перепрыгивать через соседнюю шашку на свободное поле.

Реальный результат: Программа корректно обрабатывает прыжки через шашки.

* + Множественные прыжки за один ход.

Ожидаемый результат: Программа позволяет совершать несколько последовательных прыжков за один ход.

Реальный результат: Программа корректно обрабатывает серию прыжков как через свои, так и через шашки противника.

* + Попытка недопустимого хода.

Ожидаемый результат: Программа блокирует недопустимый ход и показывает сообщение об ошибке.

Реальный результат: Программа показывает сообщение "Такой ход сейчас не доступен!".

* + Проверка очередности ходов.

Ожидаемый результат: Программа обеспечивает строгое чередование ходов белых и черных.

Реальный результат: Программа корректно контролирует очередность ходов.

1. Тесты бота:
   * Проверка выбора оптимального хода.

Ожидаемый результат: Программа выбирает ход с наилучшей оценкой по матрице весов.

Реальный результат: Программа успешно выбирает оптимальные ходы.

* + Проверка работы алгоритма минимакс.

Ожидаемый результат: Программа анализирует ходы на указанную глубину с учетом ответов противника.

Реальный результат: Программа корректно применяет алгоритм минимакс.

* + Проверка альфа-бета отсечения.

Ожидаемый результат: Программа оптимизирует перебор путем отсечения заведомо худших вариантов.

Реальный результат: Программа эффективно применяет альфа-бета отсечение.

* + Проверка оценочной функции.

Ожидаемый результат: Программа правильно оценивает позиции по матрице весов.

Реальный результат: Программа корректно использует оценочную функцию.

1. Тесты завершения игры:
   * Победа белых.

Ожидаемый результат: Программа определяет победу белых при занятии всех целевых позиций и показывает сообщение.

Реальный результат: Программа корректно определяет победу белых и отображает окно победителя.

* + Победа черных.

Ожидаемый результат: Программа определяет победу черных при занятии всех целевых позиций и показывает сообщение.

Реальный результат: Программа корректно определяет победу черных и отображает окно победителя.

* + Проверка корректного отображения диалогового окна победителя.

Ожидаемый результат: Программа отображает окно с указанием победителя и кнопкой "ОК".

Реальный результат: Программа корректно отображает окно победителя.

1. Тесты истории ходов:
   * Корректность записи хода белых.

Ожидаемый результат: Программа записывает ходы белых в формате "A1 → B2" в левую колонку.

Реальный результат: Программа корректно записывает и отображает ходы белых.

* + Корректность записи хода черных.

Ожидаемый результат: Программа записывает ходы черных в формате "A1 → B2" в правую колонку.

Реальный результат: Программа корректно записывает и отображает ходы черных.

* + Проверка формата записи.

Ожидаемый результат: Программа использует буквенно-цифровую нотацию для записи ходов.

Реальный результат: Программа корректно применяет формат записи.

* + Работа скроллинга истории ходов.

Ожидаемый результат: Программа обеспечивает плавную прокрутку при большом количестве ходов.

Реальный результат: Программа позволяет просматривать историю ходов с помощью скроллинга.

1. Тесты интерфейса:
   * Работа кнопки выхода.

Ожидаемый результат: Программа показывает окно подтверждения при нажатии кнопки выхода.

Реальный результат: Программа корректно обрабатывает нажатие кнопки выхода.

* + Отображение окна подтверждения выхода.

Ожидаемый результат: Программа отображает окно с вопросом «Хотите закрыть игру?» и кнопками "Да"/"Нет".

Реальный результат: Программа корректно отображает окно подтверждения выхода.

* + Отображение текущего игрока.

Ожидаемый результат: Программа показывает, чей сейчас ход (белые/черные).

Реальный результат: Программа корректно отображает текущего игрока.

* + Отображение сообщений об ошибках.

Ожидаемый результат: Программа показывает сообщения об ошибках.

Реальный результат: Программа корректно отображает и скрывает сообщения об ошибках.

* + Проверка отображения правил игры.

Ожидаемый результат: Программа отображает правила игры в читаемом формате с кнопкой возврата.

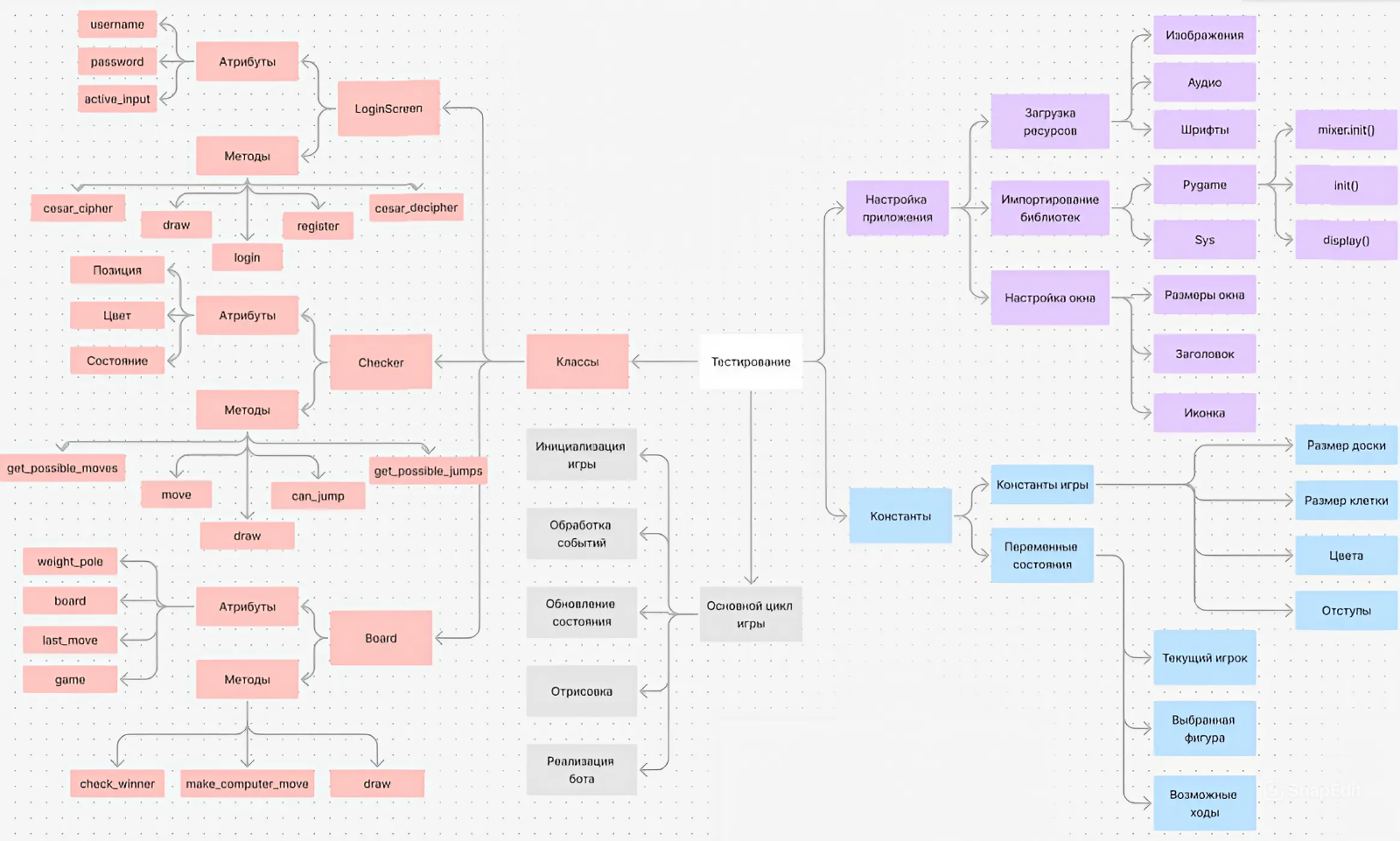
Реальный результат: Программа корректно отображает правила игры.

1. Тесты дополнительных файлов:
   * Загрузка ресурсов (изображения, звуки).

Ожидаемый результат: Программа загружает все необходимые изображения и звуковые файлы при запуске.

Реальный результат: Программа успешно загружает ресурсы при их наличии в директории.

Реализация mind\_map по проведенным тестам:



**Листинг программы**

import pygame  
import sys  
  
cell\_size = 68  
board\_size = 8  
window\_size\_w = 1100  
window\_size\_h = 700  
board\_offset = window\_size\_w - (cell\_size \* board\_size) - 50  
board\_offset\_y = 50  
white = (227, 214, 200)  
black = (0, 0, 0)  
gray = (130, 163, 194)  
fps = 60  
  
  
class LoginScreen:  
 def \_\_init\_\_(self, screen):  
 self.screen = screen  
 self.font\_header = pygame.font.SysFont('Arial', 30)  
 self.font\_entry = pygame.font.SysFont('Arial', 24)  
 self.font\_label = pygame.font.SysFont('Arial', 22)  
  
 self.username = ""  
 self.password = ""  
 self.active\_input = None  
 self.show\_register\_popup = False  
 self.is\_registration = False  
 self.game\_started = False  
 self.error\_message = ""  
 self.error\_timer = 0  
  
 self.username\_rect = pygame.Rect(window\_size\_w // 2 - 100, 230, 200, 30)  
 self.password\_rect = pygame.Rect(window\_size\_w // 2 - 100, 330, 200, 30)  
 self.login\_btn = pygame.Rect(window\_size\_w // 2 - 50, 410, 100, 40)  
  
 self.popup\_rect = pygame.Rect(window\_size\_w // 2 - 200, window\_size\_h // 2 - 100, 400, 200)  
 self.yes\_btn = pygame.Rect(window\_size\_w // 2 - 100, window\_size\_h // 2 + 20, 80, 40)  
 self.no\_btn = pygame.Rect(window\_size\_w // 2 + 20, window\_size\_h // 2 + 20, 80, 40)  
  
 def caesar\_cipher(self, text, shift=6):  
 result = ""  
 for char in text:  
 if 'А' <= char <= 'Я' or 'а' <= char <= 'я':  
 ascii\_offset = ord('а') if char.islower() else ord('А')  
 shifted = (ord(char) - ascii\_offset + shift) % 32 + ascii\_offset  
 result += chr(shifted)  
 elif 'A' <= char <= 'Z' or 'a' <= char <= 'z':  
 ascii\_offset = ord('a') if char.islower() else ord('A')  
 shifted = (ord(char) - ascii\_offset + shift) % 26 + ascii\_offset  
 result += chr(shifted)  
 else:  
 result += char  
 return result  
  
 def caesar\_decipher(self, text, shift=6):  
 return self.caesar\_cipher(text, -shift)  
  
 def handle\_event(self, event):  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if not self.show\_register\_popup:  
 if self.username\_rect.collidepoint(event.pos):  
 self.active\_input = "username"  
 elif self.password\_rect.collidepoint(event.pos):  
 self.active\_input = "password"  
 elif self.login\_btn.collidepoint(event.pos):  
 if self.is\_registration:  
 self.register\_user()  
 else:  
 self.login()  
 else:  
 self.active\_input = None  
 elif self.show\_register\_popup:  
 if self.yes\_btn.collidepoint(event.pos):  
 self.is\_registration = True  
 elif self.no\_btn.collidepoint(event.pos):  
 self.show\_register\_popup = False  
 self.username = ""  
 self.password = ""  
  
 elif event.type == pygame.KEYDOWN and self.active\_input and not self.show\_register\_popup:  
 if event.key == pygame.K\_BACKSPACE:  
 if self.active\_input == "username":  
 self.username = self.username[:-1]  
 else:  
 self.password = self.password[:-1]  
 elif event.key == pygame.K\_RETURN:  
 if self.is\_registration:  
 self.register\_user()  
 else:  
 self.login()  
 else:  
 if self.active\_input == "username":  
 self.username += event.unicode  
 else:  
 self.password += event.unicode  
  
 def login(self):  
 if len(self.password) < 3:  
 self.error\_message = "Пароль должен содержать минимум 3 символа"  
 self.error\_timer = 1500  
 return False  
  
 try:  
 with open('users.txt', 'r', encoding='utf-8') as file:  
 for line in file:  
 encrypted\_line = line.strip()  
 decrypted\_line = self.caesar\_decipher(encrypted\_line)  
 stored\_username, stored\_password = decrypted\_line.split(':')  
 if stored\_username.strip() == self.username and stored\_password.strip() == self.password:  
 self.game\_started = True  
 return True  
 self.show\_register\_popup = True  
 return False  
 except FileNotFoundError:  
 with open('users.txt', 'w', encoding='utf-8') as file:  
 pass  
 self.show\_register\_popup = True  
 return False  
  
 def register\_user(self):  
 if len(self.password) < 3:  
 self.error\_message = "Пароль должен содержать минимум 3 символа"  
 self.error\_timer = 1500  
 return False  
  
 if self.username and self.password:  
 user\_data = f"{self.username}:{self.password}"  
 encrypted\_data = self.caesar\_cipher(user\_data)  
 with open('users.txt', 'a', encoding='utf-8') as file:  
 file.write(encrypted\_data + '\n')  
 return True  
 return False  
  
 def draw(self):  
 if self.game\_started:  
 return  
  
 header\_text = "Регистрация" if self.is\_registration else "Авторизация"  
 header = self.font\_header.render(header\_text, True, black)  
 header\_rect = header.get\_rect(center=(window\_size\_w // 2, 160))  
 self.screen.blit(header, header\_rect)  
  
 username\_label = self.font\_label.render("Имя пользователя", True, black)  
 self.screen.blit(username\_label, (window\_size\_w // 2 - 100, 200))  
  
 pygame.draw.rect(self.screen, gray if self.active\_input == "username" else black,  
 self.username\_rect, 2)  
 username\_text = self.font\_entry.render(self.username, True, black)  
 self.screen.blit(username\_text, (self.username\_rect.x + 5, self.username\_rect.y + 5))  
  
 password\_label = self.font\_label.render("Пароль", True, black)  
 self.screen.blit(password\_label, (window\_size\_w // 2 - 100, 300))  
  
 pygame.draw.rect(self.screen, gray if self.active\_input == "password" else black,  
 self.password\_rect, 2)  
 hidden\_password = "\*" \* len(self.password)  
 password\_text = self.font\_entry.render(hidden\_password, True, black)  
 self.screen.blit(password\_text, (self.password\_rect.x + 5, self.password\_rect.y + 5))  
  
 pygame.draw.rect(self.screen, gray, self.login\_btn)  
 button\_text = "Сохранить" if self.is\_registration else "Войти"  
 login\_text = self.font\_label.render(button\_text, True, black)  
 login\_text\_rect = login\_text.get\_rect(center=self.login\_btn.center)  
 self.screen.blit(login\_text, login\_text\_rect)  
  
 if self.error\_message and self.error\_timer > 0:  
 error\_surface = self.font\_label.render(self.error\_message, True, (77, 19, 19))  
 error\_rect = error\_surface.get\_rect(center=(window\_size\_w // 2, 390))  
 self.screen.blit(error\_surface, error\_rect)  
 self.error\_timer -= 1  
  
 if self.show\_register\_popup:  
 overlay = pygame.Surface((window\_size\_w, window\_size\_h))  
 overlay.fill((0, 0, 0))  
 overlay.set\_alpha(128)  
 self.screen.blit(overlay, (0, 0))  
  
 pygame.draw.rect(self.screen, white, self.popup\_rect)  
 pygame.draw.rect(self.screen, black, self.popup\_rect, 2)  
  
 popup\_header = self.font\_header.render("Зарегистрироваться", True, black)  
 popup\_header\_rect = popup\_header.get\_rect(center=(window\_size\_w // 2, window\_size\_h // 2 - 70))  
 self.screen.blit(popup\_header, popup\_header\_rect)  
  
 popup\_text = self.font\_label.render("Имя пользователя и пароль не найдены", True, black)  
 popup\_text2 = self.font\_label.render("зарегистрироваться?", True, black)  
 popup\_text\_rect = popup\_text.get\_rect(center=(window\_size\_w // 2, window\_size\_h // 2 - 20))  
 popup\_text\_rect2 = popup\_text2.get\_rect(center=(window\_size\_w // 2, window\_size\_h // 2))  
 self.screen.blit(popup\_text, popup\_text\_rect)  
 self.screen.blit(popup\_text2, popup\_text\_rect2)  
  
 pygame.draw.rect(self.screen, gray, self.yes\_btn)  
 pygame.draw.rect(self.screen, gray, self.no\_btn)  
  
 yes\_text = self.font\_label.render("Да", True, black)  
 no\_text = self.font\_label.render("Нет", True, black)  
  
 yes\_text\_rect = yes\_text.get\_rect(center=self.yes\_btn.center)  
 no\_text\_rect = no\_text.get\_rect(center=self.no\_btn.center)  
  
 self.screen.blit(yes\_text, yes\_text\_rect)  
 self.screen.blit(no\_text, no\_text\_rect)  
  
  
class Checker:  
 def \_\_init\_\_(self, row, col, player, game):  
 self.row = row  
 self.col = col  
 self.x = col \* cell\_size + board\_offset + cell\_size // 2 - game.white\_checker\_img.get\_width() // 2  
 self.y = row \* cell\_size + board\_offset\_y + cell\_size // 2 - game.white\_checker\_img.get\_height() // 2  
 self.selected = False  
 self.player = player  
 self.game = game  
  
 self.animating = False  
 self.animation\_path = []  
 self.current\_path\_index = 0  
 self.animation\_progress = 0  
 self.start\_x = self.x  
 self.start\_y = self.y  
 self.target\_x = self.x  
 self.target\_y = self.y  
  
 def get\_path(self, start\_row, start\_col, end\_row, end\_col):  
 path = [(start\_row, start\_col)]  
 visited = set()  
 found\_path = []  
  
 def find\_jump\_path(x, y, target\_x, target\_y, current\_path):  
 nonlocal found\_path  
  
 if found\_path:  
 return  
  
 if (x, y) == (target\_x, target\_y):  
 found\_path = current\_path.copy()  
 return  
  
 visited.add((x, y))  
  
 for dx, dy in [(0, -2), (-2, 0), (0, 2), (2, 0)]:  
 new\_x = x + dx  
 new\_y = y + dy  
  
 if (0 <= new\_x < board\_size and 0 <= new\_y < board\_size and  
 (new\_x, new\_y) not in visited):  
  
 middle\_x = x + dx // 2  
 middle\_y = y + dy // 2  
  
 if (self.game.board.board[middle\_x][middle\_y] is not None and  
 self.game.board.board[new\_x][new\_y] is None):  
  
 current\_path.append((new\_x, new\_y))  
 find\_jump\_path(new\_x, new\_y, target\_x, target\_y, current\_path)  
 if not found\_path:  
 current\_path.pop()  
  
 visited.remove((x, y))  
  
 if abs(start\_row - end\_row) + abs(start\_col - end\_col) == 1:  
 path.append((end\_row, end\_col))  
 return path  
  
 find\_jump\_path(start\_row, start\_col, end\_row, end\_col, path)  
 return found\_path if found\_path else path  
  
 def can\_jump(self, start\_row, start\_col, end\_row, end\_col):  
 if self.game.board.board[end\_row][end\_col] is not None:  
 return False  
  
 if not (start\_row == end\_row or start\_col == end\_col):  
 return False  
  
 if start\_row == end\_row:  
 mid\_col = (start\_col + end\_col) // 2  
 return (abs(start\_col - end\_col) == 2 and  
 self.game.board.board[start\_row][mid\_col] is not None)  
 else:  
 mid\_row = (start\_row + end\_row) // 2  
 return (abs(start\_row - end\_row) == 2 and  
 self.game.board.board[mid\_row][start\_col] is not None)  
  
 def get\_possible\_jumps(self, row, col, visited=None):  
 if visited is None:  
 visited = set()  
  
 possible\_moves = set()  
  
 def test\_jump(x, y, target\_x, target\_y, visited\_positions):  
 if (x, y) not in visited\_positions:  
 visited\_positions.add((x, y))  
  
 if (x, y) != (row, col):  
 possible\_moves.add((x, y))  
  
 for dx, dy in [(0, -2), (-2, 0), (0, 2), (2, 0)]:  
 new\_x = x + dx  
 new\_y = y + dy  
  
 if 0 <= new\_x < board\_size and 0 <= new\_y < board\_size:  
 middle\_x = x + dx // 2  
 middle\_y = y + dy // 2  
  
 if (self.game.board.board[middle\_x][middle\_y] is not None and  
 self.game.board.board[new\_x][new\_y] is None and  
 (new\_x, new\_y) not in visited\_positions):  
 test\_jump(new\_x, new\_y, target\_x, target\_y, visited\_positions)  
  
 test\_jump(row, col, None, None, visited)  
  
 return possible\_moves  
  
 def get\_possible\_moves(self, row, col):  
 moves = set()  
  
 for d\_row, d\_col in [(0, 1), (0, -1), (1, 0), (-1, 0)]:  
 new\_row = row + d\_row  
 new\_col = col + d\_col  
  
 if (0 <= new\_row < board\_size and  
 0 <= new\_col < board\_size and  
 self.game.board.board[new\_row][new\_col] is None):  
 moves.add((new\_row, new\_col))  
  
 jumps = self.get\_possible\_jumps(row, col)  
 moves.update(jumps)  
  
 return moves  
  
 def start\_animation(self, path):  
 self.animating = True  
 self.animation\_path = path  
 self.current\_path\_index = 0  
 self.animation\_progress = 0  
 self.start\_x = self.x  
 self.start\_y = self.y  
 if path:  
 next\_row, next\_col = path[0]  
 self.target\_x = next\_col \* cell\_size + board\_offset + cell\_size // 2 - self.game.white\_checker\_img.get\_width() // 2  
 self.target\_y = next\_row \* cell\_size + board\_offset\_y + cell\_size // 2 - self.game.white\_checker\_img.get\_height() // 2  
  
 def update\_animation(self):  
 if self.animating:  
 self.animation\_progress += 0.1  
 if self.animation\_progress >= 1:  
 self.x = self.target\_x  
 self.y = self.target\_y  
 self.current\_path\_index += 1  
  
 if self.current\_path\_index < len(self.animation\_path):  
 self.animation\_progress = 0  
 self.start\_x = self.x  
 self.start\_y = self.y  
 next\_row, next\_col = self.animation\_path[self.current\_path\_index]  
 self.target\_x = next\_col \* cell\_size + board\_offset + cell\_size // 2 - self.game.white\_checker\_img.get\_width() // 2  
 self.target\_y = next\_row \* cell\_size + board\_offset\_y + cell\_size // 2 - self.game.white\_checker\_img.get\_height() // 2  
 else:  
 self.animating = False  
 self.game.animating = False  
 if self.animation\_path:  
 final\_row, final\_col = self.animation\_path[-1]  
 self.row = final\_row  
 self.col = final\_col  
 else:  
 t = self.animation\_progress  
 t = t \* t \* (3 - 2 \* t)  
 self.x = self.start\_x + (self.target\_x - self.start\_x) \* t  
 self.y = self.start\_y + (self.target\_y - self.start\_y) \* t  
  
 def draw(self):  
 self.update\_animation()  
 if self.player == 1:  
 self.game.screen.blit(self.game.white\_checker\_img, (self.x, self.y))  
 else:  
 self.game.screen.blit(self.game.black\_checker\_img, (self.x, self.y))  
 if self.selected:  
 pygame.draw.rect(self.game.screen, (74, 75, 77),  
 (self.col \* cell\_size + board\_offset,  
 self.row \* cell\_size + board\_offset\_y,  
 cell\_size, cell\_size), 2)  
  
 def move(self, row, col):  
 self.row = row  
 self.col = col  
 self.x = col \* cell\_size + board\_offset + cell\_size // 2 - self.game.white\_checker\_img.get\_width() // 2  
 self.y = row \* cell\_size + board\_offset\_y + cell\_size // 2 - self.game.white\_checker\_img.get\_height() // 2  
  
  
class Board:  
 def \_\_init\_\_(self, game):  
 self.game = game  
 self.reset\_board()  
 self.weight\_pole = [  
 [7, 6, 5, 4, 3, 2, 1, 0],  
 [8, 7, 6, 5, 4, 3, 2, 1],  
 [9, 8, 7, 6, 5, 4, 3, 2],  
 [10, 9, 8, 7, 6, 5, 4, 3],  
 [11, 10, 9, 8, 7, 6, 5, 4],  
 [14, 14, 14, 9, 8, 7, 6, 5],  
 [14, 14, 14, 10, 9, 8, 7, 6],  
 [14, 14, 14, 11, 10, 9, 8, 7]  
 ]  
  
 def reset\_board(self):  
 self.board = [[None for \_ in range(board\_size)] for \_ in range(board\_size)]  
 self.init\_board()  
 self.last\_move = None  
  
 def init\_board(self):  
 initial\_position = [  
 [0, 0, 0, 0, 0, -1, -1, -1],  
 [0, 0, 0, 0, 0, -1, -1, -1],  
 [0, 0, 0, 0, 0, -1, -1, -1],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 0, 0, 0],  
 [1, 1, 1, 0, 0, 0, 0, 0],  
 [1, 1, 1, 0, 0, 0, 0, 0],  
 [1, 1, 1, 0, 0, 0, 0, 0]  
 ]  
  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if initial\_position[row][col] != 0:  
 self.board[row][col] = Checker(row, col, initial\_position[row][col], self.game)  
  
 def check\_winner(self):  
 white\_win = True  
 for row in range(3):  
 for col in range(5, 8):  
 if not self.board[row][col] or self.board[row][col].player != 1:  
 white\_win = False  
 break  
 if not white\_win:  
 break  
  
 black\_win = True  
 for row in range(5, 8):  
 for col in range(3):  
 if not self.board[row][col] or self.board[row][col].player != -1:  
 black\_win = False  
 break  
 if not black\_win:  
 break  
  
 if white\_win:  
 return 1  
 elif black\_win:  
 return -1  
 return 0  
  
 def draw(self, possible\_moves):  
 self.game.screen.fill(white)  
 desk\_rect = self.game.desk\_img.get\_rect()  
 desk\_rect.left = board\_offset - 27  
 desk\_rect.top = board\_offset\_y - 27  
 self.game.screen.blit(self.game.desk\_img, desk\_rect)  
  
 for row in range(board\_size):  
 for col in range(board\_size):  
 pygame.draw.rect(self.game.screen, (197, 201, 209),  
 (col \* cell\_size + board\_offset,  
 row \* cell\_size + board\_offset\_y,  
 cell\_size, cell\_size), 2)  
 if (row, col) in possible\_moves:  
 pygame.draw.rect(self.game.screen, (6, 21, 51),  
 (col \* cell\_size + board\_offset,  
 row \* cell\_size + board\_offset\_y,  
 cell\_size, cell\_size), 2)  
  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if self.board[row][col]:  
 self.board[row][col].draw()  
  
 def make\_computer\_move(self):  
 if self.game.animating:  
 return False  
  
 def minimax(board, depth, alpha, beta, maximizing\_player):  
 winner = self.check\_winner()  
 if winner == -1:  
 return float('inf'), None, None, None, None  
 elif winner == 1:  
 return float('-inf'), None, None, None, None  
  
 if maximizing\_player:  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if board[row][col] and board[row][col].player == -1:  
 possible\_moves = board[row][col].get\_possible\_moves(row, col)  
  
 for new\_row, new\_col in possible\_moves:  
 temp = board[new\_row][new\_col]  
 board[new\_row][new\_col] = board[row][col]  
 board[row][col] = None  
  
 if self.check\_winner() == -1:  
 board[row][col] = board[new\_row][new\_col]  
 board[new\_row][new\_col] = temp  
 return float('inf'), row, col, new\_row, new\_col  
  
 board[row][col] = board[new\_row][new\_col]  
 board[new\_row][new\_col] = temp  
  
 if depth == 0:  
 score = 0  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if board[row][col]:  
 if board[row][col].player == -1:  
 score += self.weight\_pole[row][col]  
 else:  
 score -= self.weight\_pole[7 - row][7 - col]  
 return score, None, None, None, None  
  
 if maximizing\_player:  
 max\_eval = float('-inf')  
 best\_move = None  
 best\_start = None  
 moves\_found = False  
  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if board[row][col] and board[row][col].player == -1:  
 possible\_moves = board[row][col].get\_possible\_moves(row, col)  
  
 if row >= 5 and col <= 2:  
 filtered\_moves = set()  
 for new\_row, new\_col in possible\_moves:  
 if new\_row >= 5 and new\_col <= 2:  
 filtered\_moves.add((new\_row, new\_col))  
 possible\_moves = filtered\_moves  
  
 for new\_row, new\_col in possible\_moves:  
 moves\_found = True  
 temp = board[new\_row][new\_col]  
 board[new\_row][new\_col] = board[row][col]  
 board[row][col] = None  
  
 eval, \_, \_, \_, \_ = minimax(board, depth - 1, alpha, beta, False)  
  
 board[row][col] = board[new\_row][new\_col]  
 board[new\_row][new\_col] = temp  
  
 if eval > max\_eval or best\_move is None:  
 max\_eval = eval  
 best\_move = (new\_row, new\_col)  
 best\_start = (row, col)  
  
 alpha = max(alpha, eval)  
 if beta <= alpha:  
 break  
  
 if not moves\_found or best\_start is None or best\_move is None:  
 score = 0  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if board[row][col]:  
 if board[row][col].player == -1:  
 score += self.weight\_pole[row][col]  
 else:  
 score -= self.weight\_pole[7 - row][7 - col]  
 return score, None, None, None, None  
  
 return max\_eval, best\_start[0], best\_start[1], best\_move[0], best\_move[1]  
  
 else:  
 min\_eval = float('inf')  
 best\_move = None  
 best\_start = None  
 moves\_found = False  
  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if board[row][col] and board[row][col].player == 1:  
 possible\_moves = board[row][col].get\_possible\_moves(row, col)  
  
 for new\_row, new\_col in possible\_moves:  
 moves\_found = True  
 temp = board[new\_row][new\_col]  
 board[new\_row][new\_col] = board[row][col]  
 board[row][col] = None  
  
 eval, \_, \_, \_, \_ = minimax(board, depth - 1, alpha, beta, True)  
  
 board[row][col] = board[new\_row][new\_col]  
 board[new\_row][new\_col] = temp  
  
 if eval < min\_eval or best\_move is None:  
 min\_eval = eval  
 best\_move = (new\_row, new\_col)  
 best\_start = (row, col)  
  
 beta = min(beta, eval)  
 if beta <= alpha:  
 break  
  
 if not moves\_found or best\_start is None or best\_move is None:  
 score = 0  
 for row in range(board\_size):  
 for col in range(board\_size):  
 if board[row][col]:  
 if board[row][col].player == -1:  
 score += self.weight\_pole[row][col]  
 else:  
 score -= self.weight\_pole[7 - row][7 - col]  
 return score, None, None, None, None  
  
 return min\_eval, best\_start[0], best\_start[1], best\_move[0], best\_move[1]  
  
 eval\_score, start\_row, start\_col, end\_row, end\_col = minimax(self.board, 3, float('-inf'), float('inf'), True)  
  
 if None in (start\_row, start\_col, end\_row, end\_col):  
 return False  
  
 checker = self.board[start\_row][start\_col]  
 if checker is None:  
 return False  
  
 path = checker.get\_path(start\_row, start\_col, end\_row, end\_col)  
 if not path:  
 return False  
  
 checker.start\_animation(path)  
 self.game.animating = True  
 self.board[start\_row][start\_col] = None  
 self.board[end\_row][end\_col] = checker  
 checker.move(end\_row, end\_col)  
 self.last\_move = ((start\_row, start\_col), (end\_row, end\_col))  
 return True  
  
  
class Game:  
  
 def \_\_init\_\_(self):  
 pygame.init()  
 pygame.mixer.init()  
  
 self.screen = pygame.display.set\_mode((window\_size\_w, window\_size\_h))  
 pygame.display.set\_caption("Халма")  
  
 icon = pygame.image.load("new\_icon.png")  
 pygame.display.set\_icon(icon)  
  
 pygame.mixer.music.load('music\_fon.mp3')  
 pygame.mixer.music.set\_volume(0.01)  
 pygame.mixer.music.play(-1)  
 self.clock = pygame.time.Clock()  
  
 self.desk\_img = pygame.image.load("desk.jpg")  
 self.white\_checker\_img = pygame.image.load("white.png")  
 self.black\_checker\_img = pygame.image.load("black.png")  
  
 self.font = pygame.font.Font(None, 36)  
  
 self.board = Board(self)  
 self.selected\_checker = None  
 self.possible\_moves = set()  
 self.current\_player = 1  
 self.error\_message = ""  
 self.error\_timer = 0  
 self.is\_computer\_player = True  
 self.animating = False  
 self.animation\_progress = 0  
 self.show\_exit\_confirmation = False  
 self.show\_winner\_popup = False  
 self.winner = 0  
 self.login\_screen = LoginScreen(self.screen)  
 self.logged\_in = False  
  
 self.RULES\_TEXT = """  
 Правила игры Халма:  
  
  
 У каждого игрока по 9 шашек: белые внизу слева, черные вверху справа.  
  
 Игроки ходят по очереди. Возможные ходы:  
 - На одну клетку вправо, влево, вниз, вверх.  
 - Прыгать через шашки на пустую клетку.  
  
 За ход можно сделать несколько прыжков, как через свои, так и через  
 шашки противника.  
  
 Побеждает тот, кто первый переместит свои шашки в угол противника.  
  
 Совет: используйте шашки как мостики для прыжков!  
  
"""  
  
 self.moves\_history = []  
 self.moves\_font = pygame.font.SysFont('Arial', 22)  
 self.col\_to\_letter = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H'}  
 self.row\_to\_number = {0: '8', 1: '7', 2: '6', 3: '5', 4: '4', 5: '3', 6: '2', 7: '1'}  
  
 self.moves\_scroll\_y = 0  
 self.max\_visible\_moves = 15  
 self.scroll\_speed = 25  
 self.moves\_area\_height = 550  
  
 def add\_move\_to\_history(self, start\_pos, end\_pos, player):  
 start\_notation = f"{self.col\_to\_letter[start\_pos[1]]}{self.row\_to\_number[start\_pos[0]]}"  
 end\_notation = f"{self.col\_to\_letter[end\_pos[1]]}{self.row\_to\_number[end\_pos[0]]}"  
 move\_text = f"{start\_notation} → {end\_notation}"  
 self.moves\_history.append((move\_text, player))  
  
 def handle\_scroll(self, event):  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 content\_height = len(self.moves\_history) \* 35  
  
 if content\_height > self.moves\_area\_height:  
 scroll\_bar\_rect = pygame.Rect(380, 60, 10, self.moves\_area\_height)  
  
 if scroll\_bar\_rect.collidepoint(event.pos):  
 relative\_y = event.pos[1] - 60  
 scroll\_ratio = relative\_y / self.moves\_area\_height  
 self.moves\_scroll\_y = min(content\_height - self.moves\_area\_height,  
 max(0, int(content\_height \* scroll\_ratio)))  
 else:  
 if event.button == 4:  
 self.moves\_scroll\_y = max(0, self.moves\_scroll\_y - self.scroll\_speed)  
 elif event.button == 5:  
 max\_scroll = content\_height - self.moves\_area\_height  
 self.moves\_scroll\_y = min(max\_scroll, self.moves\_scroll\_y + self.scroll\_speed)  
  
 def draw\_moves\_history(self):  
 white\_header = self.font.render("Ходы белых", True, black)  
 black\_header = self.font.render("Ходы черных", True, black)  
 self.screen.blit(white\_header, (20, 20))  
 self.screen.blit(black\_header, (200, 20))  
  
 total\_height = max(self.moves\_area\_height, len(self.moves\_history) \* 35)  
 moves\_surface = pygame.Surface((350, total\_height))  
 moves\_surface.fill(white)  
  
 white\_y = 0  
 black\_y = 0  
  
 for move\_text, player in self.moves\_history:  
 text = self.moves\_font.render(move\_text, True, black)  
 if player == 1:  
 moves\_surface.blit(text, (0, white\_y))  
 white\_y += 35  
 else:  
 moves\_surface.blit(text, (180, black\_y))  
 black\_y += 35  
  
 content\_height = max(white\_y, black\_y)  
  
 if content\_height > self.moves\_area\_height:  
 self.moves\_scroll\_y = min(content\_height - self.moves\_area\_height,  
 max(0, self.moves\_scroll\_y))  
  
 visible\_rect = pygame.Rect(0, self.moves\_scroll\_y, 350, self.moves\_area\_height)  
 self.screen.blit(moves\_surface, (20, 60), visible\_rect)  
  
 if content\_height > self.moves\_area\_height:  
 scroll\_height = max(30, self.moves\_area\_height \* self.moves\_area\_height / content\_height)  
 scroll\_pos = (self.moves\_scroll\_y \* (self.moves\_area\_height - scroll\_height) /  
 (content\_height - self.moves\_area\_height))  
  
 pygame.draw.rect(self.screen, (200, 200, 200),  
 (380, 60, 10, self.moves\_area\_height))  
  
 pygame.draw.rect(self.screen, (150, 150, 150),  
 (380, 60 + scroll\_pos, 10, scroll\_height))  
  
 def show\_winner\_dialog(self):  
 overlay = pygame.Surface((window\_size\_w, window\_size\_h))  
 overlay.fill((0, 0, 0))  
 overlay.set\_alpha(128)  
 self.screen.blit(overlay, (0, 0))  
  
 dialog\_width = 400  
 dialog\_height = 200  
 dialog\_x = (window\_size\_w - dialog\_width) // 2  
 dialog\_y = (window\_size\_h - dialog\_height) // 2  
  
 dialog\_rect = pygame.Rect(dialog\_x, dialog\_y, dialog\_width, dialog\_height)  
 pygame.draw.rect(self.screen, white, dialog\_rect)  
 pygame.draw.rect(self.screen, black, dialog\_rect, 2)  
  
 winner\_text = "Победили белые!" if self.winner == 1 else "Победили черные!"  
 text = self.font.render(winner\_text, True, black)  
 text\_rect = text.get\_rect(center=(window\_size\_w // 2, dialog\_y + 60))  
 self.screen.blit(text, text\_rect)  
  
 ok\_button = pygame.Rect(dialog\_x + (dialog\_width - 100) // 2, dialog\_y + dialog\_height - 60, 100, 40)  
 pygame.draw.rect(self.screen, gray, ok\_button)  
 ok\_text = self.font.render("OK", True, black)  
 ok\_rect = ok\_text.get\_rect(center=ok\_button.center)  
 self.screen.blit(ok\_text, ok\_rect)  
  
 return ok\_button  
  
 def wrap\_text(self, text, font, max\_width):  
 words = text.split(' ')  
 lines = []  
 current\_line = []  
 current\_width = 0  
  
 for word in words:  
 word\_surface = font.render(word + ' ', True, black)  
 word\_width = word\_surface.get\_width()  
  
 if current\_width + word\_width <= max\_width:  
 current\_line.append(word)  
 current\_width += word\_width  
 else:  
 lines.append(' '.join(current\_line))  
 current\_line = [word]  
 current\_width = word\_width  
  
 if current\_line:  
 lines.append(' '.join(current\_line))  
 return lines  
  
 def show\_rules(self):  
 rules\_running = True  
 padding = 50  
 line\_spacing = 40  
 max\_width = window\_size\_w - (padding \* 2)  
  
 while rules\_running:  
 self.screen.fill(white)  
  
 y = padding  
 for line in self.RULES\_TEXT.split('\n'):  
 if line.strip():  
 if line.strip()[0].isdigit():  
 text\_surface = self.font.render(line, True, black)  
 self.screen.blit(text\_surface, (padding, y))  
 y += line\_spacing  
 else:  
 wrapped\_lines = self.wrap\_text(line, self.font, max\_width)  
 for wrapped\_line in wrapped\_lines:  
 text\_surface = self.font.render(wrapped\_line, True, black)  
 self.screen.blit(text\_surface, (padding, y))  
 y += line\_spacing  
  
 back\_button = pygame.Rect(window\_size\_w // 2 - 50, window\_size\_h - 100, 100, 40)  
 pygame.draw.rect(self.screen, gray, back\_button)  
 back\_text = self.font.render("Назад", True, black)  
 back\_text\_rect = back\_text.get\_rect(center=back\_button.center)  
 self.screen.blit(back\_text, back\_text\_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 back\_button.collidepoint(event.pos):  
 rules\_running = False  
  
 def show\_menu(self):  
 menu\_running = True  
 while menu\_running:  
 self.screen.fill(white)  
  
 start\_button = pygame.Rect(window\_size\_w // 2 - 100, window\_size\_h // 2 - 50, 200, 40)  
 rules\_button = pygame.Rect(window\_size\_w // 2 - 100, window\_size\_h // 2 + 10, 200, 40)  
  
 pygame.draw.rect(self.screen, gray, start\_button)  
 pygame.draw.rect(self.screen, gray, rules\_button)  
  
 start\_text = self.font.render("Начать игру", True, black)  
 rules\_text = self.font.render("Правила игры", True, black)  
  
 start\_text\_rect = start\_text.get\_rect(center=start\_button.center)  
 rules\_text\_rect = rules\_text.get\_rect(center=rules\_button.center)  
  
 self.screen.blit(start\_text, start\_text\_rect)  
 self.screen.blit(rules\_text, rules\_text\_rect)  
  
 pygame.display.flip()  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.mixer.music.stop()  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if start\_button.collidepoint(event.pos):  
 menu\_running = False  
 while not self.logged\_in:  
 self.screen.fill(white)  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.mixer.music.stop()  
 pygame.quit()  
 sys.exit()  
 self.login\_screen.handle\_event(event)  
 if event.type == pygame.MOUSEBUTTONDOWN and self.login\_screen.login\_btn.collidepoint(  
 event.pos):  
 if self.login\_screen.login():  
 self.logged\_in = True  
  
 self.login\_screen.draw()  
 pygame.display.flip()  
  
 return self.show\_opponent\_selection()  
 if rules\_button.collidepoint(event.pos):  
 self.show\_rules()  
  
 def reset\_game\_state(self):  
 self.board.reset\_board()  
 self.selected\_checker = None  
 self.possible\_moves = set()  
 self.current\_player = 1  
 self.error\_message = ""  
 self.error\_timer = 0  
 self.animating = False  
 self.animation\_progress = 0  
 self.show\_winner\_popup = False  
 self.winner = 0  
 self.moves\_history = []  
 self.moves\_scroll\_y = 0  
  
 def show\_opponent\_selection(self):  
 self.is\_computer\_player = True  
 self.reset\_game\_state()  
 return True  
  
 def show\_exit\_dialog(self):  
 overlay = pygame.Surface((window\_size\_w, window\_size\_h))  
 overlay.fill((0, 0, 0))  
 overlay.set\_alpha(128)  
 self.screen.blit(overlay, (0, 0))  
  
 dialog\_width = 300  
 dialog\_height = 150  
 dialog\_x = (window\_size\_w - dialog\_width) // 2  
 dialog\_y = (window\_size\_h - dialog\_height) // 2  
  
 dialog\_rect = pygame.Rect(dialog\_x, dialog\_y, dialog\_width, dialog\_height)  
 pygame.draw.rect(self.screen, white, dialog\_rect)  
 pygame.draw.rect(self.screen, black, dialog\_rect, 2)  
  
 text = self.font.render("Хотите закрыть игру?", True, black)  
 text\_rect = text.get\_rect(center=(window\_size\_w // 2, dialog\_y + 60))  
 self.screen.blit(text, text\_rect)  
  
 yes\_button = pygame.Rect(dialog\_x + 50, dialog\_y + 100, 80, 30)  
 no\_button = pygame.Rect(dialog\_x + 170, dialog\_y + 100, 80, 30)  
  
 pygame.draw.rect(self.screen, gray, yes\_button)  
 pygame.draw.rect(self.screen, gray, no\_button)  
  
 yes\_text = self.font.render("Да", True, black)  
 no\_text = self.font.render("Нет", True, black)  
  
 yes\_rect = yes\_text.get\_rect(center=yes\_button.center)  
 no\_rect = no\_text.get\_rect(center=no\_button.center)  
  
 self.screen.blit(yes\_text, yes\_rect)  
 self.screen.blit(no\_text, no\_rect)  
  
 return yes\_button, no\_button  
  
 def main\_game(self):  
 exit\_button = pygame.Rect(20, window\_size\_h - 40, 40, 30)  
  
 while True:  
 self.clock.tick(fps)  
  
 winner = self.board.check\_winner()  
 if winner != 0:  
 self.winner = winner  
 self.show\_winner\_popup = True  
  
 if self.show\_winner\_popup:  
 ok\_button = self.show\_winner\_dialog()  
 pygame.display.flip()  
  
 waiting\_for\_ok = True  
 while waiting\_for\_ok:  
 for event in pygame.event.get():  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if ok\_button.collidepoint(event.pos):  
 self.reset\_game\_state()  
 self.show\_exit\_confirmation = False  
 return  
 elif event.type == pygame.QUIT:  
 pygame.mixer.music.stop()  
 pygame.quit()  
 sys.exit()  
  
 if self.current\_player == -1 and self.is\_computer\_player and not self.animating:  
 if self.board.make\_computer\_move():  
 last\_move = self.board.last\_move  
 if last\_move:  
 self.add\_move\_to\_history(last\_move[0], last\_move[1], -1)  
 self.current\_player = 1  
 self.error\_message = ""  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.mixer.music.stop()  
 pygame.quit()  
 sys.exit()  
  
 self.handle\_scroll(event)  
  
 if event.type == pygame.MOUSEBUTTONDOWN and not self.animating:  
 mouse\_pos = event.pos  
  
 if exit\_button.collidepoint(mouse\_pos):  
 self.show\_exit\_confirmation = True  
 elif self.show\_exit\_confirmation:  
 yes\_button, no\_button = self.show\_exit\_dialog()  
 if yes\_button.collidepoint(mouse\_pos):  
 self.reset\_game\_state()  
 self.show\_exit\_confirmation = False  
 return  
 elif no\_button.collidepoint(mouse\_pos):  
 self.show\_exit\_confirmation = False  
 else:  
 if self.current\_player == 1 or not self.is\_computer\_player:  
 col = (mouse\_pos[0] - board\_offset) // cell\_size  
 row = (mouse\_pos[1] - board\_offset\_y) // cell\_size  
  
 if 0 <= row < board\_size and 0 <= col < board\_size:  
 self.handle\_click(row, col)  
  
 self.screen.fill(white)  
 self.board.draw(self.possible\_moves)  
 self.draw\_moves\_history()  
 self.draw\_interface(exit\_button)  
 if self.show\_exit\_confirmation:  
 self.show\_exit\_dialog()  
 pygame.display.flip()  
  
 def handle\_click(self, row, col):  
 if self.selected\_checker:  
 can\_move = False  
  
 if (self.board.board[row][col] is None and  
 ((abs(self.selected\_checker.row - row) == 1 and self.selected\_checker.col == col) or  
 (abs(self.selected\_checker.col - col) == 1 and self.selected\_checker.row == row))):  
 can\_move = True  
 elif (row, col) in self.possible\_moves:  
 can\_move = True  
  
 if can\_move:  
 start\_pos = (self.selected\_checker.row, self.selected\_checker.col)  
  
 path = self.selected\_checker.get\_path(self.selected\_checker.row, self.selected\_checker.col, row, col)  
 self.selected\_checker.start\_animation(path)  
 self.animating = True  
  
 self.board.board[self.selected\_checker.row][self.selected\_checker.col] = None  
 self.board.board[row][col] = self.selected\_checker  
  
 self.add\_move\_to\_history(start\_pos, (row, col), self.current\_player)  
  
 self.current\_player = -self.current\_player  
 self.error\_message = ""  
 else:  
 self.error\_message = "Такой ход сейчас не доступен!"  
 self.error\_timer = 300  
  
 self.selected\_checker.selected = False  
 self.selected\_checker = None  
 self.possible\_moves = set()  
 elif self.board.board[row][col] and self.board.board[row][col].player == self.current\_player:  
 self.selected\_checker = self.board.board[row][col]  
 self.selected\_checker.selected = True  
 self.possible\_moves = self.selected\_checker.get\_possible\_moves(row, col)  
  
 def draw\_interface(self, exit\_button):  
 pygame.draw.rect(self.screen, gray, exit\_button)  
 exit\_text = self.font.render("<=", True, black)  
 exit\_text\_rect = exit\_text.get\_rect(center=exit\_button.center)  
 self.screen.blit(exit\_text, exit\_text\_rect)  
  
 player\_text = f"Ходят: {'белые' if self.current\_player == 1 else 'черные'}"  
 text\_surface = self.font.render(player\_text, True, black)  
 text\_rect = text\_surface.get\_rect(center=(window\_size\_w - 320, window\_size\_h - 30))  
 self.screen.blit(text\_surface, text\_rect)  
  
 if self.error\_message and self.error\_timer > 0:  
 error\_surface = self.font.render(self.error\_message, True, (77, 19, 19))  
 error\_rect = error\_surface.get\_rect(center=(window\_size\_w - 320, window\_size\_h - 60))  
 self.screen.blit(error\_surface, error\_rect)  
 self.error\_timer -= 1  
  
  
def main():  
 game = Game()  
 while True:  
 if game.show\_menu():  
 game.main\_game()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()