МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  |  |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  на курсовую работу  по дисциплине «Алгоритмы и структуры данных»  Тема «Компьютерная логическая игра  «Турецкие шашки-Поддавки»» | | | | | | |
|  | | |  | | | | Руководитель разработки:  Кандидат технических наук, доцент кафедры «ИВК»  Шишкин Вадим Викторинович  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | |
|  | | | | Исполнитель  студент гр. ИСТбд-22  Марочкина А.Д.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | |
|  | | 2024 | | | | | | |

Содержание

Аннотация 2

Техническое задание 3

Пояснительная записка. 10

Руководство программиста XX

Аннотация

Данный документ представляет собой пояснительную записку на курсовую работу на тему «Турецкие шашки-Поддавки». 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»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | ТЕХНИЧЕСКОЕ ЗАДАНИЕ  на курсовую работу  по дисциплине «Алгоритмы и структуры данных»  Тема «Компьютерная логическая игра «Турецкие шашки- Поддавки» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Марочкина А.Д.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**Введение:**

Название программы: «Турецкие шашки-Поддавки».

Приложение предназначено для игры в «Турецкие шашки-Поддавки» с компьютером. Основные правила игры «Турецкие шашки-Поддавки» заключаются в следующем:

*Доска и начальная расстановка*

Для игры в Турецкие шашки-Поддавки используется прямоугольная доска размером 8×8 клеток. Соперникам перед началом игры предоставляется по 16 шашек, одному — белых, другому — чёрных. Шашки расставляются на второй и третьей от игрока горизонталях, по 8 шашек в ряд, при этом первая от игрока горизонталь остаётся свободной.

*Правила ходов*

* Простая шашка ходит на одно поле вперёд, влево, вправо.
* Дамка ходит на любое количество пустых полей вперёд, назад, вправо, влево.

*Правила взятия шашек соперника*

* Если у игрока при его ходе есть возможность взятия (боя) шашек противника, он обязан бить. Бой возможен только тогда, когда поле за шашкой противника свободно. Если с новой позиции шашки, побившей шашку противника, можно бить дальше, бой продолжается (за один ход можно побить несколько шашек противника).
* Если есть несколько вариантов боя, игрок обязан выбрать тот, при котором берётся наибольшее количество шашек противника. Это относится к взятию и шашками, и дамками.
* Если есть несколько вариантов боя с равным количеством взятых шашек, игрок вправе выбрать любой из них.
* Простая шашка бьёт шашку противника, стоящую спереди, справа или слева (бить назад запрещено), перескакивая через неё на следующее поле по вертикали или горизонтали.
* Дамка бьёт шашки противника, стоящие от неё через любое количество пустых клеток спереди, сзади, справа и слева, если следующее за шашкой поле свободно. Как и простая шашка, дамка может за один ход побить несколько шашек противника.
* В турецких шашках отсутствует правило турецкого удара: при взятии шашки снимаются с доски одна за другой по ходу боя, но при этом дамка не имеет права во время ударного хода по вертикали или горизонтали изменить его направление на противоположное, то есть на 180°.

*Превращение в дамку*

* Простая шашка, вступившая на восьмую горизонталь, становится дамкой.
* Простая шашка становится дамкой после завершения хода. Если она попадает на восьмую горизонталь в результате взятия и может бить дальше, как простая шашка, она продолжает бить и становится дамкой по завершении хода. Продолжить бить как дамка на этом же ходу она не может.

*Завершение игры и определение победителя*

* Выигрывает тот, кто сам отдаст все свои шашки или позволит их запереть противнику. Если на доске осталось по одной шашке — объявляется ничья.

Программное приложение должно обеспечивать следующие возможности:

* 1. Удобный и понятный графический интерфейс с возможностью управления шашками.
  2. Отдельный раздел, в котором будет осуществляться доступ к правилам и описанию игры, возможность выхода, возможность начать игру сначала.
  3. Поле для входа/регистрации.

1. **Основания для разработки**

Учебная программа по дисциплине «Алгоритмы и структуры данных» по специальности 09.03.02 «Информационные системы и технологии».

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

**2.1. Функциональное назначение**

Основное назначение приложения - предоставление интерактивной среды, позволяющей играть в «Турецкие шашки-Поддавки» против другого игрока.

Приложение должно обеспечивать:

1. Визуальное представление: отображение шашек на графическом интерфейсе
2. Управление шашками: возможность делать ходза определенный цвет.

Автоматизируемые процессы:

1. Обработка действий игрока и обеспечение корректного выполнения правил игры.
2. Алгоритм для смены хода, для обеспечения игры с другим игроком.
3. Предоставления специального раздела, в котором будет осуществляться доступ к правилам и описанию игры.
4. Раздел для выполнения входа/регистрации.

**2.2. Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно иметь модульную организацию. Модули разделены на блоки, каждый из которых отвечает за конкретный аспект работы программы. Основные модули программы:

1. Модуль игрового поля и интерфейса: отвечает за графическое использование игровых полей, шашек; обработка взаимодействия пользователя с элементами на экране.
2. Модуль логики игры: содержит основные правила игры «Турецкие шашки-Поддавки», включая условия перемещения шашек, захвата фигурки и выполнения игры; обработка всех ходов игрока, проверяя их на соответствие правилам
3. Модуль входа/регистрации: реализует алгоритм позволяющий пользователю совершить вход или пройти регистрацию.
4. Модуль правил: отвечает за раздел, где находятся основные правила игры для пользователя.
   * 1. Требования к составу функций приложения

Основные функции:

1. Функции игрового процесса: отвечают за создание игры, расстановку шашек на игровом поле, обработку ходов пользователя и проверку условий на завершение и выполнение игры.
2. Функции входа/регистрации: отвечают вход или регистрацию пользователя.
3. Функции правил: отвечают за отдельный раздел с объяснением основных правил игры.

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

Приложение должно обеспечивать интуитивно понятный и удобный пользовательский интерфейс, поддерживающий основной функционал игры, а также обеспечивать удобную структуру для хранения и обмена данными.

1. Основной интерфейс игры:

1) Чёткое обеспечение игрового поля с визуально различающимися клетками, обеспечивающее видимость возможных ходов и текущих положений шашек.

2) Панель управления, содержащая: текстовые поля для ввода имени пользователя и пароля, а также кнопки входа или кнопки регистрации пользователя, а также кнопка «Правила игры».

3. Система правил:

1) Отдельный раздел, где будут хранится основные правила для игры в «Турецкие шашки-Поддавки».

4. Алгоритм обмена данными:

1) Обмен между модулями: модули игрового интерфейса и логики игры предоставляют данные о ходах и состоянии игровых полей через интерфейсные события (например, клик мыши, выбор шашек).

**2.3 Требования к надежности**

Работоспособность:

1. Приложение должно корректно работать на заданных платформах (Windows, Linux, macOS) без сбоев и багов.
2. Графический интерфейс должен быть отзывчивым и реагировать на действия пользователя.
3. Все функции приложения должны доступно выполняться без ошибок.

Восстановление при сбоях:

1. В случае возникновения ошибки, приложение должно вывести сообщение об ошибке, после устранения ошибки, перезапустить приложение.
   1. **Требования к информационной и программной совместимости**

Операционная система: Windows, Linux, MacOS.

Язык программирования: Python.

Среда разработки: PyCharm.

Библиотеки: tkinter – библиотека для создания графического интерфейса, PIL – библиотека для работы с растровой графикой.

База данных: использование файла формата .json для хранения имени пользователя и пароля для входа в приложение.

* 1. **Требования к маркировке и упаковке**

Определяются заданием на курсовую работу.

**2.6 Требования к транспортированию и хранению**

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются.

2.6.2 Условия хранения

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

2.6.3 Сроки хранения

Срок хранения — до окончания срока учебы.

1. **Требования к программной документации**

Определяются заданием на курсовую работу.

1. **Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приемки**

Определяются заданием на курсовую работу.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  на курсовую работу  по дисциплине «Алгоритмы и структуры данных»  Тема «Компьютерная логическая игра «Турецкие шашки-Поддавки» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Марочкина А.Д.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**Введение**

Название приложения «Турецкие шашки-Поддавки». «Турецкие шашки-Поддавки» – одна из разновидностей игры в [шашки](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D1%88%D0%BA%D0%B8), характерной особенностью которой, в отличие от большинства вариантов шашек, является то, что ходы и взятия шашками делаются не по диагоналям, а по вертикалям и горизонталям.

* 1. **Проектная часть**
  2. **Постановка задачи на разработку приложения**

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

* 1. **Математические методы**

В основе игры лежит дискретная сетка и предполагает анализ позиций шашек на поле.

Игровое поле моделируется в виде двумерной сетки размерами *8×8*. Каждая ячейка определяется координатами (row, col), где:

row – строка;

col – столбец;

По сути, доска представляет собой граф, где: клетки – это узлы графа, а возможные ходы и захваты шашек – это ребра графа.

Обоснование выбора данной модели:

- достаточно проста в реализации;

- удобно реализовать проверку на доступность хода или захвата фигуры противника;

- модель легко масштабируется.

**1.3 Архитектура и алгоритмы**

**1.3.1 Архитектура**

Игра построена с использованием объектно-ориентированной архитектуры, где основным объектом является игровое поле.

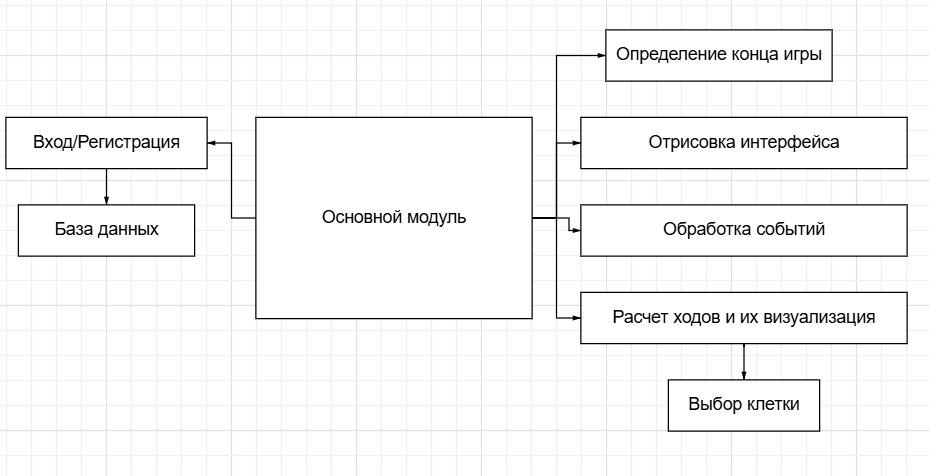


Рис.1. Архитектура приложения

Основные структуры данных и их взаимодействие:

**Классы:** Класс доски управляет общим состоянием игры и использует объекты(шашки) для представления шашек и их свойств. Класс регистрации отвечает за вход/регистрацию пользователя.

**Списки и объекты:** Массив хранит в себе объекты(шашки), а методы манипулируют этими объектами через индексацию массива.

**Словари:** Используются для передачи информации о доступности хода.

**Постоянные:** Используются для обозначения цветов, границ игрового поля.

**Файловая структура:** Применяется для хранения данных о пользователе(его имя и пароль).

**1.3.2 Алгоритм поиска ходов**

На рис. 2 представлен алгоритм поиска хода

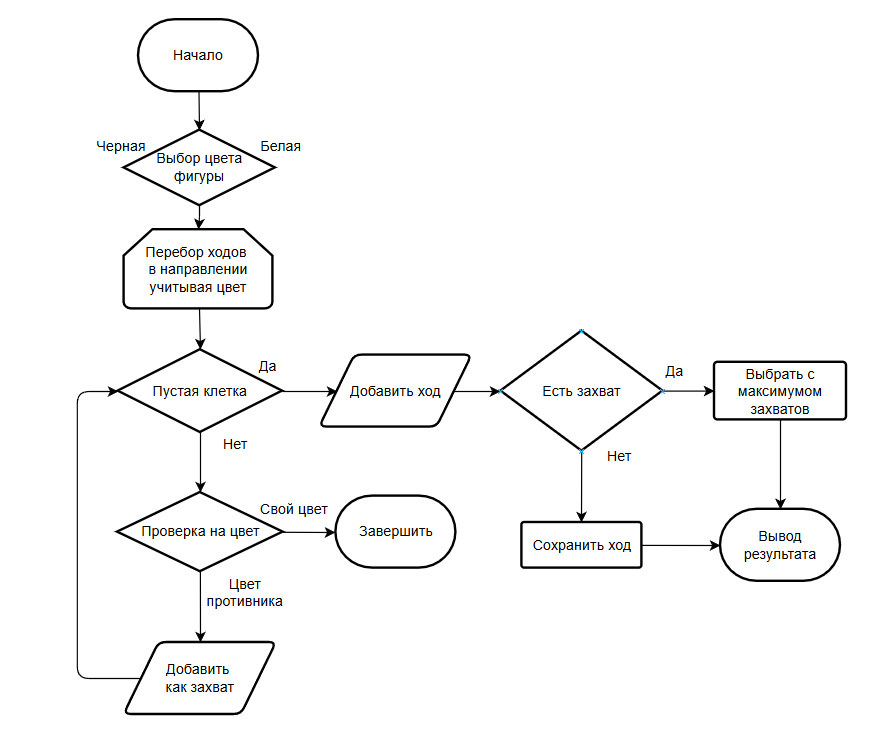


Рис.2 Алгоритм поиска хода

**1.3.3 Алгоритм входа/регистрации**

На рис. 3 представлен алгоритм входа/регистрации

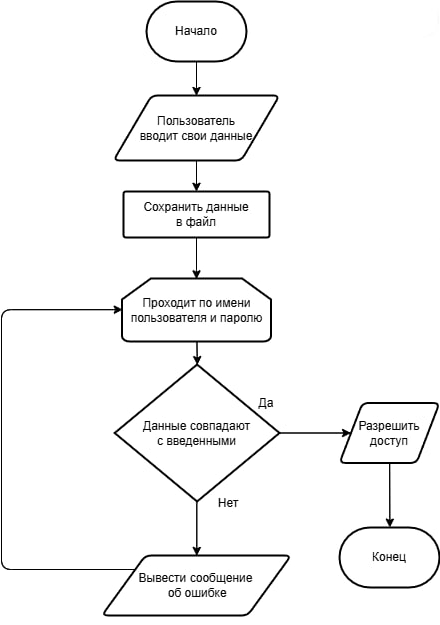


Рис. 3 Алгоритм входа/регистрации

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | РУКОВОДСТВО ПРОГРАММИСТА  на курсовую работу  по дисциплине «Алгоритмы и структуры данных»  Тема «Компьютерная логическая игра “Турецкие шашки-Поддавки”» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Марочкина А.Д.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**Введение**

Данный проект представляет собой полноценную игру в «Турецкие шашки-Поддавки» с графическим интерфейсом, который делает игровой процесс удобным и привлекательным. Игра включает в себя широкий спектр возможностей, позволяя пользователям наслаждаться игрой друг против друга.

* + - 1. **Назначения и условия применения программы.**

**1.1 Назначения и функции, выполняемые приложением.**

Основные функции:

* Поддержка игры между двумя игроками.
* авторизация и регистрация пользователей с использованием системы проверки имени и пароля, а также хэширование пароля.
* Возможность начать игру заново или посмотреть правила игры.
* Визуализация игрового поля с возможностью перемещения игровых элементов (шашек) по клеткам, а так же подсветкой возможных ходов.

Реализация правил игры:

* Игроки ходят по очереди, перемещая свои фигуры.
* Если у игрока появляется возможность захватить фигуру противника, то после совершения данного хода с поля удаляется захваченная фигура.
* Отображение игровых сообщений (Победа, поражение или ничья).
* Ведение базы данных для хранения данных о пользователе

**1.2 Условия, необходимые для использования приложения.**

Операционная система: *Windows* 10/11.

Язык программирования: *Python* (рекомендуется версия 3.8+).

Среда разработки: *PyCharm*.

Библиотеки:

1. *tkinter –* используется для создания графического интерфейса пользователя;
2. *hashlib*  – используется для хеширования данных.
3. *OS* – для предоставления работы с файловой системой компьютера.
   * + 1. **Характеристики программы.**

**2.1 Характеристика приложения.**

Количество строк программного кода: ~432 строк.

Реализовано 5 классов: *Game, Draw, GameBoard, AuthForm, ChechersGame.*

Описание работы приложения:

* 1. При запуске файла ‘main.py’, приложение открывает окно Регистрации/входа. Пользователя предоставляется возможность ввести логин и пароль для успешной регистрации или входа. Также он может сразу ознакомиться с правилами игры.

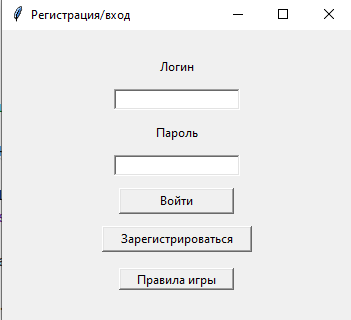


Рисунок 1 – Окно входа

* 1. После успешного входа или регистрации открывается игровое поле(рисунок 2).

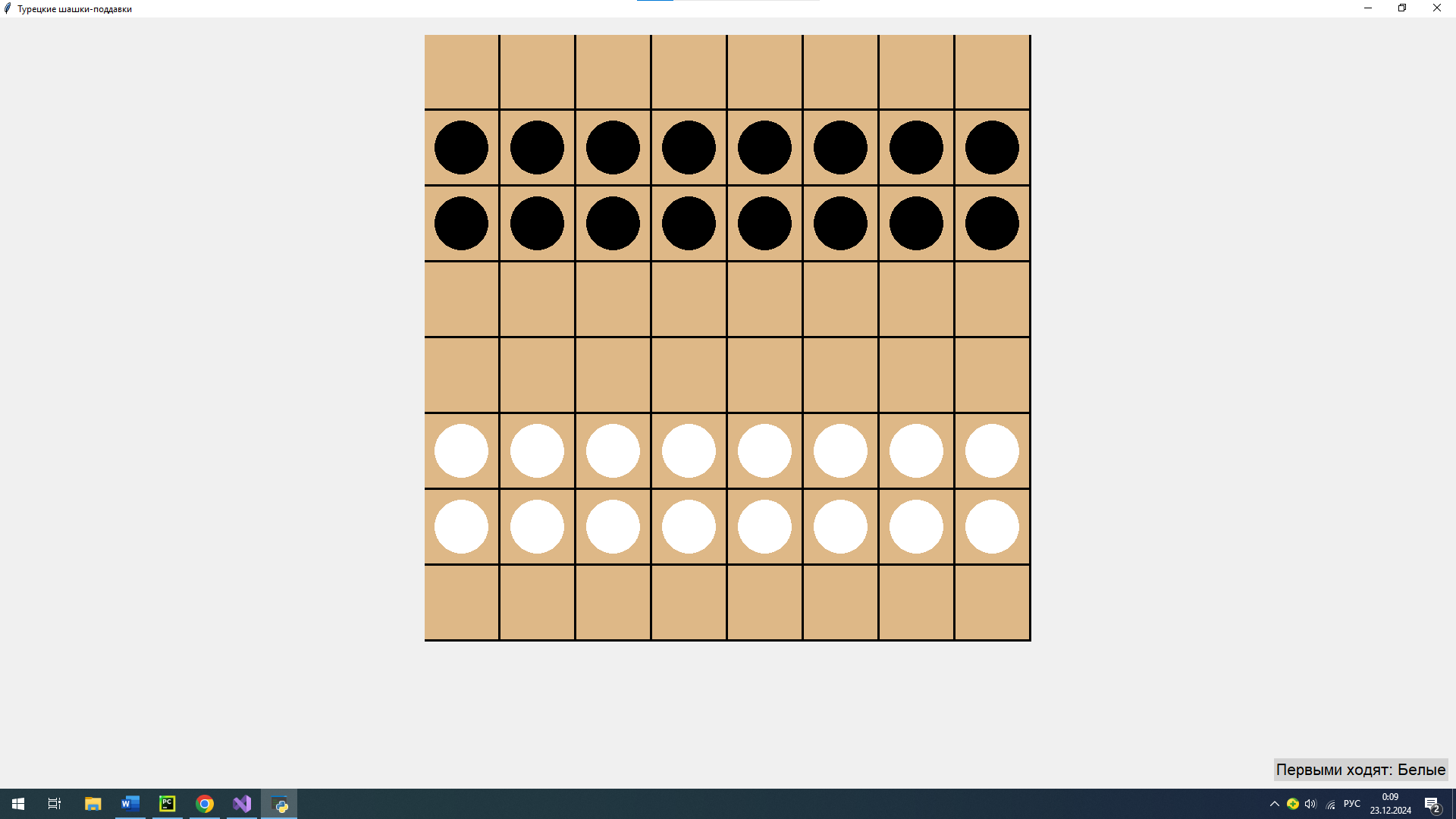


Рисунок 2 – Графический интерфейс игры

* 1. Пользователь поочередно с другим игроком (например другом), перемещает фигуры, следуя правилам игры (рисунок 3)

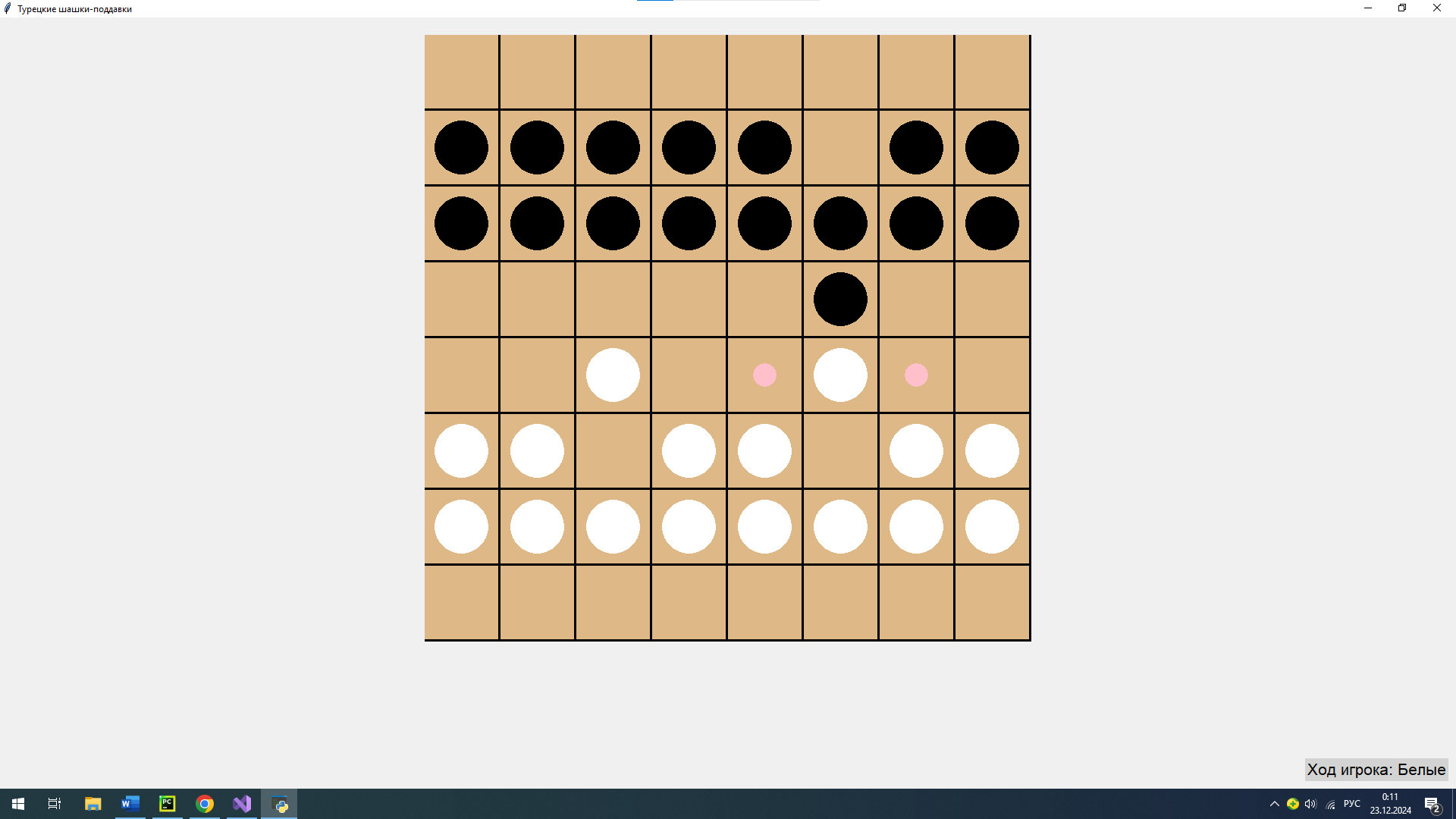
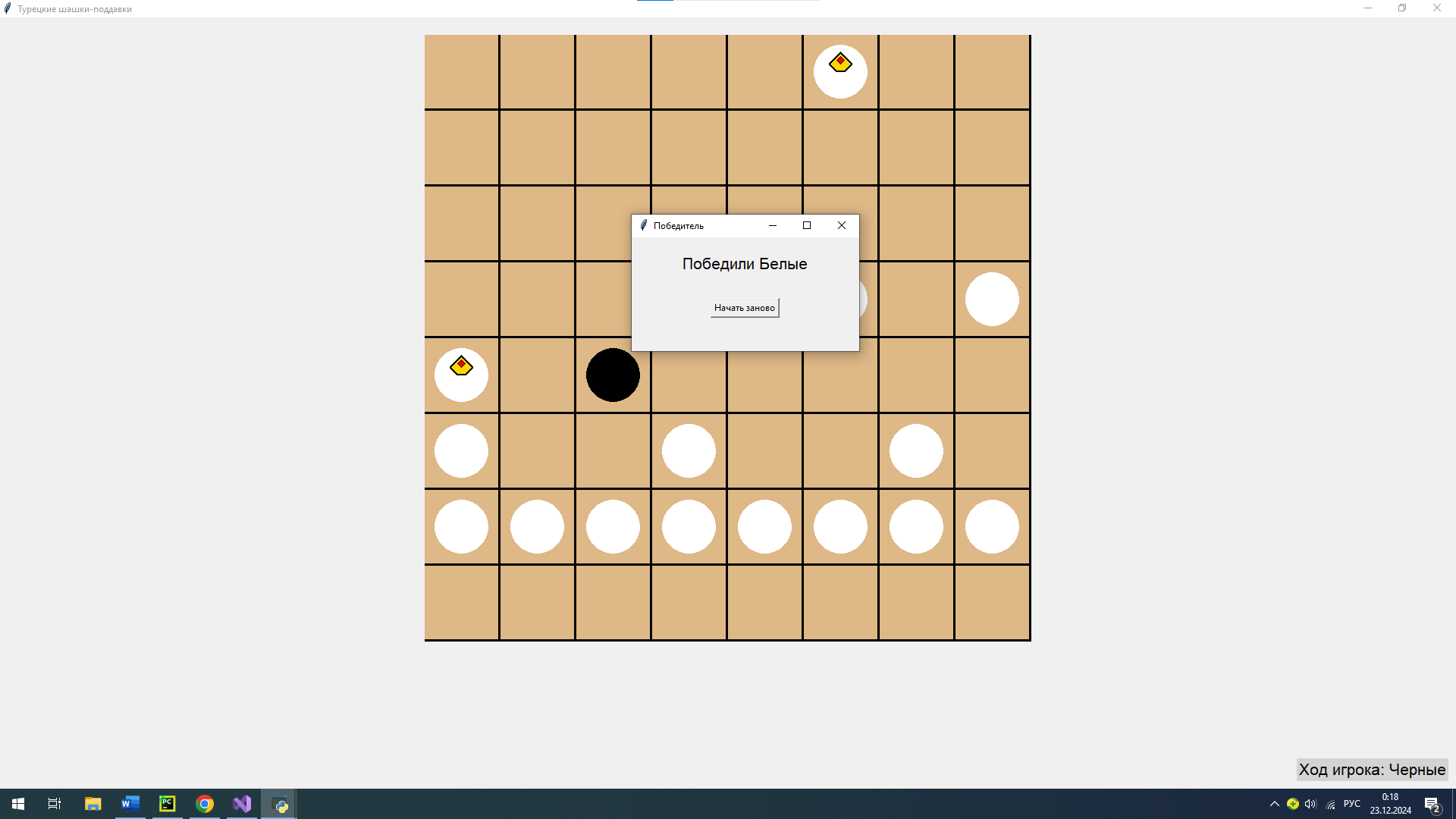


Рисунок 3 – Ходы пользователей

* 1. При завершении игры открывается окно с сообщением о победители партии, а также возможность начать игру заново. (рисунок 4)

  
Рисунок 4 – Сообщение о победе белых в соответствии с правилами игры (игрок, у которого на доске осталась одна фигура, не может продолжить игру).

**2.2 Особенности реализации приложения.**

Списки используются для хранения данных о положении шашек на игровом поле. Они обеспечивают быструю и удобную обработку игровых данных. Строки файлов применяются для хранения имен пользователей и их паролей. Словари используются для передачи информации о доступности хода. Постоянные обеспечивают цвета и размер игрового поля.

* + - 1. **Обращение к программе.**

Методы файла ‘main.py’:

Методы класса *Game*:

\_\_init\_\_(self, canvas, turn\_label) - Инициализирует игровое поле (canvas), метку хода (turn\_label) и вызывает метод reset() для установки начального состояния игры.

reset(self) - Сбрасывает состояние игры до начального. Создает новую игровую доску (GameBoard), задает начальный ход белым и сбрасывает список доступных ходов и обновляет текст метки turn\_label.

update(self) - Отвечает за обновление графического интерфейса игры. Перерисовывает игровое поле, а также подсвечивает доступные ходы.

move\_piece(self, row, col) - Выполняет перемещение выбранной шашки на новую позицию. Проверяет, является ли выбранное место допустимым ходом. Перемещает шашку, удаляет захваченные фигуры (если есть) и переключает ход. Если перемещение невозможно, возвращает False.

highlight\_valid\_moves(self, moves) - Отображает допустимые ходы для выбранной шашки. Рисует круги розового цвета (Pink) на клетках, куда можно переместить шашку.

select(self, row, col) - Обрабатывает выбор шашки и попытку её перемещения. Если шашка уже выбрана, вызывает метод move\_piece для её перемещения. При успешном выборе новой шашки сохраняет её как selected и определяет все возможные ходы (valid\_moves) с использованием метода get\_valid\_moves. Обновляет текст на экране, если игрок пытается выбрать не свою шашку.

switch\_turn(self) - Переключает ход между игроками. Сбрасывает список допустимых ходов (valid\_moves). Меняет текущего игрока (белые или черные). Обновляет текст метки turn\_label, чтобы указать текущего игрока.

show\_winner\_window(self, winner) - Отображает всплывающее окно с информацией о победителе. Центрирует окно на экране, а также добавляет кнопку для сброса игры.

reset\_and\_close\_winner\_window(self) - Закрывает окно с победителем и перезапускает игру.

Методы класса *Draw*:

\_\_init\_\_(self, row, col, color) - Инициализирует новый объект шашки с заданными координатами, цветом и базовыми параметрами. Устанавливает начальные координаты (row, col) и цвет (color) шашки.

draw(self, canvas) - Отображает шашку на игровом поле. Рисует внешний контур для создания визуального эффекта выделения. Отображает основную часть шашки в её цвете (self.color). Если шашка является дамкой, вызывает метод draw\_crown для отображения короны.

draw\_crown(self, canvas) – Рисует золотую корону и красный бриллиант поверх фигуры.

calculate\_position(self) - Вычисляет пиксельные координаты центра шашки на основе её строки и столбца. Центр клетки вычисляется с использованием размера клетки (100) и текущих координат (row, col).

move(self, row, col) - Перемещает шашку в новую позицию на доске. Изменяет строку и столбец шашки. Пересчитывает её пиксельные координаты вызовом calculate\_position.

make\_queen(self) - Преобразует шашку в дамку. Отображается корона в True.

Методы класса Game*Board*:

\_\_init\_\_(self, canvas) - Создает объект класса Game*Board*, устанавливает его параметры, такие как доска (self.gameboard), количество шашек, и холст для отображения. Вызывает метод initialize\_gameboard() для генерации начальной расстановки шашек. Изначально задается 16 шашек для каждой стороны. Ведется учет "дамок" через self.white\_queens и self.black\_queens.

initialize\_gameboard(self) - Создает начальную расстановку шашек на доске. Все ячейки доски инициализируются значением 0 (если клетка пуста). Располагает черные шашки на втором и третьем ряду (строки 1-2), а белые — на шестом и седьмом ряду (строки 5-6).

draw\_gameboard(self) - Рисует шахматную доску на холсте. Используется 1 цвет (burlywood). Удаляем все элементы с холста перед перерисовкой, чтобы избежать наложений.

move(self, piece, row, col) - Перемещает шашку piece на новую позицию (строку row и столбец col). Если шашка достигает противоположного конца доски, она становится дамкой (make\_queen). Увеличивает счетчик дамок соответствующего цвета (self.black\_queens или self. white\_queens).

remove\_pieces(self, pieces)- Удаляет из игры указанные шашки (при взятии). Обновляет счетчики оставшихся шашек для каждого цвета (self.white\_pieces или self.black\_pieces).

get\_piece(self, row, col) - Возвращает шашку, находящуюся на позиции (строка row, столбец col). Если на указанной позиции пусто, возвращается 0.

scan\_vertical(self, start, stop, step, color, col, skipped=[]) - Проверяет возможные движения вдоль вертикали (вверх или вниз). Учитывает наличие шашек противника и позволяет выполнять взятия.

scan\_horizontal(self, start, stop, step, color, row, skipped=[]) - Проверяет возможные движения вдоль горизонтали (влево или вправо). Поддерживает логику взятий, аналогично вертикальному движению.

get\_valid\_moves(self, piece) - Определяет все допустимые ходы для шашки piece, включая простые ходы и взятия. При наличии возможных взятий возвращает только их. Для дамок применяет отдельную логику проверки.

scan\_vertical\_queen(self, start, stop, step, color, row, skipped=[]) - Проверяет возможные ходы дамки вдоль горизонтали.

determine\_winner(self) - Возвращает результат игры, основываясь на оставшихся шашках каждого цвета. Определяет победителя или фиксирует ничью.

def get\_row\_col(event) - Преобразует координаты мыши в индексы строки и столбца игровой доски.

hash\_password(password) - Хеширует пароль с использованием алгоритма SHA-256.

check\_user\_exists(username) - Проверяет, существует ли пользователь с данным именем в файле users.txt.

authenticate(username, password) - Проверяет, соответствует ли указанный пароль хешу, сохраненному для данного пользователя.

add\_user(username, password) - Добавляет нового пользователя в файл users.txt, если имя пользователя еще не занято.

Методы класса *AuthForm:*

\_\_init\_\_(self, root) - Инициализирует окно для входа и регистрации: Создает элементы интерфейса: поля для ввода логина и пароля, кнопки для входа, регистрации и открытия правил игры. Центрирует окно на экране, используя размеры экрана пользователя.

login(self) - Проверяет введенные имя пользователя и пароль. Если данные верны (проверка через authenticate), отображает сообщение об успешном входе.Далее закрывает окно авторизации и запускает игру (start\_checkers\_game). Если данные неверны, отображает сообщение об ошибке. Использует messagebox для отображения всплывающих сообщений.

Методы класса *Resize*:

\_\_init\_\_(self, root) - Инициализирует объект класса, связывая его с корневым окном (root) приложения.

resize\_and\_center(self, window\_width, window\_height) - Устанавливает размеры окна (window\_width, window\_height) и автоматически размещает его в центре экрана.

register(self) - Проверяет введенные данные: Если поля пусты, отображает сообщение об ошибке. Если логин уже занят, показывает ошибку. Проверяет, существует ли пользователь с таким логином через функцию check\_user\_exists. Если пользователь существует, показывает ошибку. Если нет, добавляет нового пользователя через add\_user, хешируя его пароль. Использует messagebox для уведомлений об успешной регистрации или ошибках.

show\_rules(self) - Открывает файл с правилами игры, проверяет наличие файла с правилами игры (правила шашек.pdf). Если файл существует, открывает его с помощью стандартного приложения (os.startfile). Если файл не найден, показывает сообщение об ошибке.

Методы класса CheckersApp:

\_\_init\_\_(self, root) - Создает интерфейс игры, включая: Холст для игрового поля, метку с указанием текущего хода turn\_label. Устанавливает размеры окна через пиксели(длины и ширины окна). Инициализирует объект класса Game, который управляет логикой игры. Привязывает событие щелчка мыши по игровому полю к методу on\_click. Вызывает метод update\_game для начального обновления интерфейса.

on\_click(self, event) - Использует функцию get\_row\_col для преобразования координат мыши в строку и столбец. Определяет, по какому квадрату доски был произведен клик, и передает эту информацию в игровую логику (self.game.select).

Методы для запуска:

start\_checkers\_game() - Создает объект класса CheckersApp и запускает главный цикл Tk. Запускает игровой интерфейс приложения.

root.mainloop() - Запускает начальный интерфейс с выбором входа или регистрации..

Методы для работы с пользователем:

check\_user\_exists - Сравнивает введенный пароль с паролем из файла. Проверяет, существует ли пользователь с указанным именем и паролем в файле users.txt

add\_user- Регистрирует нового пользователя в файле users.txt. Если пользователь уже существует, возвращает False. Если логин свободен, добавляет запись и сохраняет файл.

* + - 1. **Сообщения.**

Перечисляются сообщения, выдаваемые по результатам контроля корректности ввода/вывода.

На рисунке 5 представлено окно ошибки при попытке входа без введенных данных пользователя.

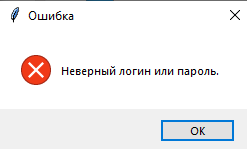


Рисунок 5 – Ошибка “Неверный логин или пароль”

На рисунке 6 представлено окно ошибки при попытке ввода неправильных данных.

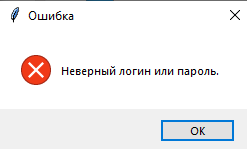


Рисунок 6 – Ошибка “Неверный логин или пароль”

На рисунке 7 представлено окно ошибки при попытке пройти регистрацию с пустыми полями.

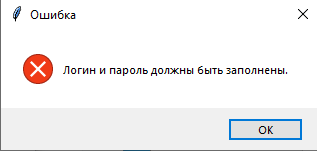


Рисунок 7 – Ошибка “Логин и пароль должны быть заполнены”

На рисунке 8 представлено окно об успешной регистрации.

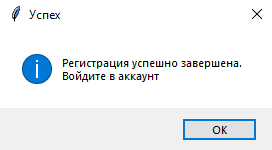


Рисунок 8 – “Регистрация успешно завершена. Войдите в аккаунт”

На рисунке 9 представлена ошибка при попытке зарегистрировать пользователя с уже существующем именем.

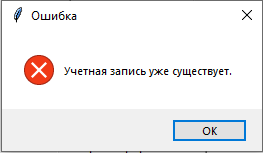


Рисунок 9 – “Учетная запись уже существует”

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | |  | | ТЕСТОВАЯ ДОКУМЕНТАЦИЯ  на курсовую работу  по дисциплине «Алгоритмы и структуры данных»  Тема «Компьютерная логическая игра “Турецкие шашки-Поддавки”» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Марочкина А.Д.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

* 1. **Введение.**

Турецкие шашки-Поддавки— это одна из популярных вариаций шашечной игры, отличающаяся уникальными правилами и стратегиями. Важной частью разработки любой программы, симулирующей эту игру, является тестирование, которое гарантирует корректность реализации игровых правил, стабильность функционала и удобство использования приложения.

Данная тестовая документация предназначена для описания подхода к тестированию программы, обеспечивающей игру в Турецкие шашки-Поддавки. В ней детализированы все аспекты проверки: от базовой проверки правил игры до тестирования сложных стратегических ситуаций, пользовательского интерфейса и производительности.

* 1. **Области тестирования.**

В рамках тестирования разработанного приложения были выделены ключевые области такие как:

**2.1 Проверка соответствию правил.**

Убедиться, что все игровые механики реализованы правильно, включая начальную расстановку, ходы, захват фигур и условия окончания игры.

#### **2.2 Тестирование интерфейса**

* Тестирование отображения шахматной доски.
* Проверка реакции интерфейса на ввод пользователя (например, отображение возможных ходов).

#### **2.3 Работа с пользовательским вводом**

* Проверка на обработку корректного и некорректного ввода:
  + Попытка переместить фигуру в недопустимую клетку.
  + Клик по пустой клетке без выбора фигуры.
* Проверка реакции программы на экстремальные сценарии, например, многократное нажатие.
  1. **Методы тестирования.**

**3.1 Модульное тестирование.**

На данном этапе тестирования проверяется корректность работы отдельных, изолированных модулей (или функций) кода. Цель — убедиться, что каждый модуль выполняет свою задачу в соответствии со спецификацией. В “Турецких шашках” модульным тестирование может являться:

* Генерация допустимых ходов для каждой фигуры: Проверяется, что алгоритм корректно генерирует все возможные допустимые ходы для каждой фигуры с учетом текущей позиции на доске и правил игры.
* Обработка кликов на игровой доске: Проверяется, что при клике на конкретную клетку доски игра корректно распознает выбранную фигуру и обрабатывает ход, корректно обрабатывая ситуации, когда клетка пуста, занята фигурой союзника или противника.
* Валидация ходов: Проверка того, что функция, которая проверяет легальность хода, корректно отслеживает все правила шахмат (например, не попадает ли король под шах, не блокируется ли ход собственной фигурой).

**3.2 Общее тестирование.**

На этом уровне проверяется работа всей системы в целом, включая пользовательский интерфейс и выполнение сложных игровых ситуаций. Целью является - убедиться, что игра работает как законченное программное обеспечение и соответствует всем требованиям.

* Проверка полного игрового сценария: Воспроизведение нескольких игровых партий с разными исходами (победа, поражение, ничья).
* Тестирование графического интерфейса: Проверка удобства и отзывчивости интерфейса. Включает в себя проверку навигации по меню, отзывчивость кнопок, ясность сообщений об ошибках.

**3.3 Тестирование производительности.**

Проверка устойчивости приложения к высоким нагрузкам и длительному использованию. Цель — выявление узких мест и потенциальных ошибок, которые могут возникнуть при интенсивной эксплуатации.

* Оценка использования процессора и памяти программы во время игры, чтобы убедиться, что игра не потребляет слишком много системных ресурсов, особенно на слабых устройствах.
* Проверка работы игры при длительном игровом процессе, например, на протяжении нескольких часов.
* Проверка игры при интенсивных действиях, например, при большом количестве взятий.

**4. Набор тест-кейсов**

| **ID** | **Название** | **Шаги** | **Ожидаемый результат** | **Итоги** |
| --- | --- | --- | --- | --- |
| ТК1 | Кнопка «Правила игры» | 1)Запустить программу  2)Нажать на кнопку «Правила игры» | Откроется файл с правилами игры | Файл открылся  Пройден |
| TK2 | Проверка начальных позиций | 1) Запустить программу  2) Проверить как отрисовано игровое.  3) Проверить на правильных ли позициях стоят шашки в соответствии с правилами игры | 1) Доска отрисована верно  2)Белые шашки стоят на 6 и 7 вертикали  3) Черные шашки стоят на 2 и 3 вертикали | Все запущено корректно. Все фигуры на своих местах. Пройден |
| ТК3 | Ход шашки (по вертикали) | 1)Запустить программу  2)Выбрать шашку, находящуюся на доске.  3) Переместить ее на свободную клетку вперед или вниз в зависимости от того, какого она цвета | Шашка переместиться на указанную клетку вперед или вниз | Шашка переместилась. Пройден |
| ТК4 | Ход шашки (по горизонтали) | 1)Запустить программу  2)Выбрать шашку, находящуюся на доске.  3)Переместить ее вперед для того, чтобы дать ей возможность переместиться по горизонтали.  4) Затем выполнить движение по горизонтали на свободную клетку (влево или вправо) | Шашка после того, как у нее появится возможность движения по горизонтали, сможет переместиться влево или вправо | Шашка переместилась. Пройден |
| ТК5 | Ход шашки, если на ее пути шашка одинакового цвета | 1)Запустить программу 2)Выбрать шашку. 3)Переместить ее вперед, затем переместить шашку противника.  4)Переместить вторую шашку радом со своей первой.  5)На следующий свой ход попробовать переместить первую шашку в сторону шашки рядом (которую мы переместили в 3 пункте) | Первая шашка не сможет переместиться в сторону шашки своего цвета | Пройден |
| ТК6 | Преобразование в дамку | 1)Запустить программу  2)Довести одну из шашек до последней (0 или 7 по индексу) горизонтали.  3)Проверить произошла ли отрисовка короны на шашке. | При достижении нужной горизонтали на обычной шашке появится корона | Пройден |
| ТК7 | Проверка хода дамки | 1)Запустить программу  2)Довести одну из шашек до последней (0 или 7 по индексу) горизонтали.  3)Проверить появилась ли картинка короны на шашке. 4)Выбрать дамку и попробовать совершить ход | При выборе дамки появятся поля на которые она может переместиться и при нажатии на доступную клетку, дамка перемещается на него | Пройден |
| ТК8 | Проверка взятия шашки противника | 1)Запустить программу  2)Совершив 2 хода, поставить шашку противника на пути вашей шашки  3)Совершить захват | Шашка противника удаляется с доски, и ваша шашка становится за шашкой противника | Пройден |
| ТК9 | Проверка взятия дамкой | 1)Запустить программу  2)Превратить обычную шашку в дамку  3)Поставить шашку противника на пути дамки  4)Совершить захват дамкой | Шашка противника удаляется с доски, и дамка занимает клетку за этой шашкой | Пройден |
| ТК10 | Проверка отрисовки доступного хода | 1)Запустить программу  2)Выбрать шашку  3)Посмотреть на то, какие клетки доступны для хода  4)Сверить возможные ходы с правилами и сделать вывод | Доступные ходы отрисованы согласно правилам игры | Пройден |
| ТК11 | Проверка на совершение недопустимого хода | 1)Запустить программу  2)Выбрать шашку  3)Попробовать переместить на любую клетку кроме подсвеченной | Невозможно совершить ход | Пройден |
| ТК12 | Проверка хода дамки если на ее пути шашка такого же цвета | 1)Запустить программу  2)Превратить шашку в дамку  3)Поставить на пути дамки шашку того же цвета, что и дамка  4)Попробовать сделать ход через шашку своего цвета | Невозможно совершить ход | Пройден |
| ТК13 | Проверка победы одной из сторон | 1)Запустить программу  2)Захватить все шашки противника или оставить одну | Появится сообщение о победы одной из сторон | Пройден |

**Код программы**

import tkinter as tk  
from tkinter import messagebox  
import hashlib  
import os  
  
class Game:  
 def \_\_init\_\_(self, canvas, turn\_label):  
 self.canvas = canvas  
 self.turn\_label = turn\_label  
 self.reset()  
  
 def reset(self):  
 self.selected = None  
 self.board = GameBoard(self.canvas)  
 self.turn = "white"  
 self.valid\_moves = {}  
 self.turn\_label.config(text="Первыми ходят: Белые")  
 self.update()  
  
 def update(self):  
 self.board.draw\_gameboard()  
 self.highlight\_valid\_moves(self.valid\_moves)  
  
 def move\_piece(self, row, col):  
 if self.selected and (row, col) in self.valid\_moves:  
 self.board.move(self.selected, row, col)  
 skipped = self.valid\_moves.get((row, col), [])  
 if skipped:  
 self.board.remove\_pieces(skipped)  
 self.switch\_turn()  
 self.update()  
 return True  
 return False  
   
 def highlight\_valid\_moves(self, moves):  
 for move in moves:  
 row, col = move  
 x = col \* 100 + 100 // 2  
 y = row \* 100 + 100 // 2  
 self.canvas.create\_oval(x - 15, y - 15, x + 15, y + 15, fill="pink", outline="pink")  
  
 def select(self, row, col):  
 if self.selected:  
 if not self.move\_piece(row, col):  
 self.selected = None  
 self.select(row, col)  
 else:  
 piece = self.board.get\_piece(row, col)  
 if piece != 0 and piece.color == self.turn:  
 self.selected = piece  
 self.valid\_moves = self.board.get\_valid\_moves(piece)  
 self.update()  
 return True  
  
 def switch\_turn(self):  
 self.valid\_moves = {}  
 if self.turn == "white":  
 self.turn = "black"  
 self.turn\_label.config(text="Ход игрока: Черные")  
 else:  
 self.turn = "white"  
 self.turn\_label.config(text="Ход игрока: Белые")  
  
 winner = self.board.determine\_winner()  
 if winner:  
 self.show\_winner\_window(winner)  
  
 def show\_winner\_window(self, winner):  
 self.winner\_window = tk.Toplevel(self.canvas.winfo\_toplevel())  
 self.winner\_window.title("Победитель")  
 """Центрирует заданное окно на экране."""  
 screen\_width = self.winner\_window.winfo\_screenwidth()  
 screen\_height = self.winner\_window.winfo\_screenheight()  
 x = (screen\_width // 2) - 75  
 y = (screen\_height // 2) - 75  
 self.winner\_window.geometry(f"{300}x{150}+{x}+{y}")  
  
 tk.Label(self.winner\_window, text=winner, font=("Arial", 16)).pack(pady=20)  
 tk.Button(self.winner\_window, text="Начать заново", command=self.reset\_and\_close\_winner\_window).pack(pady=10)  
  
 def reset\_and\_close\_winner\_window(self):  
 self.winner\_window.destroy()  
 self.reset()  
  
class Draw:  
 def \_\_init\_\_(self, row, col, color):  
 self.row = row  
 self.col = col  
 self.color = color  
 self.queen = False  
 self.calculate\_position()  
  
 def draw(self, canvas):  
 canvas.create\_oval(  
 self.x - 35, self.y - 35,  
 self.x + 35, self.y + 35,  
 fill=self.color, outline=""  
 )  
  
 if self.queen:  
 self.draw\_crown(canvas)  
  
 def draw\_crown(self, canvas):  
 crown\_points = [  
 (self.x - 15, self.y - 10), (self.x - 5, self.y - 20),   
 (self.x, self.y - 25), (self.x + 5, self.y - 20),   
 (self.x + 15, self.y - 10), (self.x + 10, self.y - 5),   
 (self.x + 5, self.y), (self.x - 5, self.y),   
 (self.x - 10, self.y - 5)   
 ]  
 canvas.create\_polygon(crown\_points, fill="gold", outline="black", width=2)  
  
 diamond\_points = [  
 (self.x, self.y - 20), (self.x + 5, self.y - 15),  
 (self.x, self.y - 10), (self.x - 5, self.y - 15)   
 ]  
 canvas.create\_polygon(diamond\_points, fill="red", outline="black", width=1)  
   
 def calculate\_position(self):  
 self.x = 100 \* self.col + 50  
 self.y = 100 \* self.row + 50  
  
 def move(self, row, col):  
 self.row, self.col = row, col  
 self.calculate\_position()  
  
 def make\_queen(self):  
 self.queen = True  
  
class GameBoard:   
 def \_\_init\_\_(self, canvas):  
 self.canvas = canvas  
 self.grid = []   
 self.white\_pieces = 16  
 self.black\_pieces = 16   
 self.white\_queens = 0  
 self.black\_queens = 0   
 self.initialize\_gameboard()   
   
 def initialize\_gameboard(self):   
 for row in range(8):  
 self.grid.append([])  
 for col in range(8):  
 if 1 <= row <= 2:  
 self.grid[row].append(Draw(row, col, "black"))  
 elif 5 <= row <= 6:  
 self.grid[row].append(Draw(row, col, "white"))  
 else:  
 self.grid[row].append(0)  
   
 def draw\_gameboard(self):  
 self.canvas.delete("all")  
 for row in range(8):  
 for col in range(8):  
 x0, y0 = col \* 100, row \* 100  
 x1, y1 = x0 + 100, y0 + 100  
 self.canvas.create\_rectangle(x0, y0, x1, y1, fill="burlywood", outline="black", width=3)  
 piece = self.grid[row][col]  
 if piece:  
 piece.draw(self.canvas)  
  
 def move(self, piece, row, col):   
 self.grid[piece.row][piece.col], self.grid[row][col] = 0, piece  
 piece.move(row, col)  
 if row in (0, 7):  
 piece.make\_queen()  
 if piece.color == "black":  
 self.black\_queens += 1  
 else:  
 self.white\_queens += 1  
  
 def remove\_pieces(self, pieces):  
 for piece in pieces:  
 self.grid[piece.row][piece.col] = 0  
 if piece.color == "white":  
 self.white\_pieces -= 1  
 else:  
 self.black\_pieces -= 1  
  
 def get\_piece(self, row, col):  
 return self.grid[row][col] if 0 <= row < 8 and 0 <= col < 8 else None  
  
 def scan\_vertical(self, start, stop, step, color, col, skipped=[]):  
 moves = {}  
 last = []  
 for r in range(start, stop, step):  
 current = self.grid[r][col]  
 if current == 0:  
 if skipped and not last:  
 break  
 elif skipped:  
 moves[(r, col)] = skipped + last  
 else:  
 moves[(r, col)] = last  
 if last:  
 new\_skipped = skipped + last  
 moves.update(self.scan\_vertical(r + step, stop, step, color, col, skipped=new\_skipped))  
 moves.update(self.scan\_horizontal(col - 1, -1, -1, color, r, skipped=new\_skipped))  
 moves.update(self.scan\_horizontal(col + 1, 8, 1, color, r, skipped=new\_skipped))  
 break  
 elif current.color == color:  
 break  
 else:  
 if last:  
 break  
 last = [current]  
 return moves  
   
 def scan\_vertical\_queen(self, start, stop, step, color, col, skipped=[]):  
 moves = {}  
 last = []  
 for r in range(start, stop, step):  
 current = self.grid[r][col]  
 if current == 0:  
 if skipped:  
 moves[(r, col)] = skipped + last  
 else:  
 moves[(r, col)] = last  
 if last:  
 new\_skipped = skipped + last  
 moves.update(self.scan\_vertical(r + step, stop, step, color, col, skipped=new\_skipped))  
 moves.update(self.scan\_horizontal(col - 1, -1, -1, color, r, skipped=new\_skipped))  
 moves.update(self.scan\_horizontal(col + 1, 8, 1, color, r, skipped=new\_skipped))  
 elif current.color == color:  
 break  
 else:  
 if last:  
 break  
 last = [current]  
 return moves  
  
 def scan\_horizontal(self, start, stop, step, color, row, skipped=[]):  
 moves = {}  
 last = []  
 for c in range(start, stop, step):  
 current = self.grid[row][c]  
 if current == 0:  
 if skipped and not last:  
 break  
 elif skipped:  
 moves[(row, c)] = skipped + last  
 else:  
 moves[(row, c)] = last  
 if last:  
 new\_skipped = skipped + last  
 moves.update(self.scan\_horizontal(c + step, stop, step, color, row, skipped=new\_skipped))  
 moves.update(self.scan\_vertical(row - 1, -1, -1, color, c, skipped=new\_skipped))  
 moves.update(self.scan\_vertical(row + 1, 8, 1, color, c, skipped=new\_skipped))  
 break  
 elif current.color == color:  
 break  
 else:  
 if last:  
 break  
 last = [current]  
 return moves  
  
 def scan\_horizontal\_queen(self, start, stop, step, color, row, skipped=[]):  
 moves = {}  
 last = []  
 for c in range(start, stop, step):  
 current = self.grid[row][c]  
 if current == 0:  
 if skipped:  
 moves[(row, c)] = skipped + last  
 else:  
 moves[(row, c)] = last  
 if last:  
 new\_skipped = skipped + last  
 moves.update(self.scan\_horizontal(c + step, stop, step, color, row, skipped=new\_skipped))  
 moves.update(self.scan\_vertical(row - 1, -1, -1, color, c, skipped=new\_skipped))  
 moves.update(self.scan\_vertical(row + 1, 8, 1, color, c, skipped=new\_skipped))  
 elif current.color == color:  
 break  
 else:  
 if last:  
 break  
 last = [current]  
 return moves  
   
 def get\_valid\_moves(self, piece):  
 moves = {}  
 row, col = piece.row, piece.col  
  
 if piece.color == "white" and not piece.queen:  
 moves.update(self.scan\_vertical(row - 1, -1, -1, piece.color, col))   
 moves.update(self.scan\_horizontal(col - 1, -1, -1, piece.color, row))   
 moves.update(self.scan\_horizontal(col + 1, 8, 1, piece.color, row))  
 elif piece.color == "black" and not piece.queen:  
 moves.update(self.scan\_vertical(row + 1, 8, 1, piece.color, col))  
 moves.update(self.scan\_horizontal(col - 1, -1, -1, piece.color, row))  
 moves.update(self.scan\_horizontal(col + 1, 8, 1, piece.color, row))  
 elif piece.queen:  
 moves.update(self.scan\_vertical\_queen(row - 1, -1, -1, piece.color, col))   
 moves.update(self.scan\_vertical\_queen(row + 1, 8, 1, piece.color, col))  
 moves.update(self.scan\_horizontal\_queen(col - 1, -1, -1, piece.color, row))   
 moves.update(self.scan\_horizontal\_queen(col + 1, 8, 1, piece.color, row))  
  
 capture\_moves = {pos: skipped for pos, skipped in moves.items() if skipped}  
 if capture\_moves:  
 max\_captures = max(len(skipped) for skipped in capture\_moves.values())  
 return {pos: skipped for pos, skipped in capture\_moves.items() if len(skipped) == max\_captures}  
  
 return moves  
   
 def determine\_winner(self):  
 if self.black\_pieces <= 1:  
 return "Победили Белые"  
 elif self.white\_pieces <= 1:  
 return "Победили Черные"  
 elif self.white\_pieces == 1 and self.black\_pieces == 1:  
 return "Ничья"  
 return None  
  
def get\_row\_col(event):  
 return event.y // 100, event.x // 100  
  
def hash\_password(password):  
 return hashlib.sha256(password.encode('utf-8')).hexdigest()  
  
def check\_user\_exists(username):  
 if os.path.exists("users.txt"):  
 with open("users.txt", "r") as file:  
 for line in file:  
 parts = line.strip().split(':')  
 if len(parts) == 2 and parts[0] == username:  
 return True  
 return False  
  
def authenticate(username, password):  
 hashed\_password = hash\_password(password)  
 if os.path.exists("users.txt"):  
 with open("users.txt", "r") as file:  
 for line in file:  
 parts = line.strip().split(':')  
 if len(parts) == 2 and parts[0] == username and parts[1] == hashed\_password:  
 return True  
 return False  
  
def add\_user(username, password):  
 if check\_user\_exists(username):  
 return False  
 hashed\_password = hash\_password(password)  
 with open("users.txt", "a") as file:  
 file.write(f"{username}:{hashed\_password}\n")  
 return True  
  
class AuthForm:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Регистрация/вход")  
 self.frame = tk.Frame(root)  
 self.frame.pack(pady=20)  
  
 tk.Label(self.frame, text="Логин").pack(padx=6, pady=6)  
 self.login\_entry = tk.Entry(self.frame, bd=2)  
 self.login\_entry.pack(padx=6, pady=6)  
  
 tk.Label(self.frame, text="Пароль").pack(padx=6, pady=6)  
 self.password\_entry = tk.Entry(self.frame, bd=2, show='\*')  
 self.password\_entry.pack(padx=6, pady=6)  
  
 tk.Button(self.frame, text="Войти", command=self.login, width=15).pack(padx=6, pady=6)  
 tk.Button(self.frame, text="Зарегистрироваться", command=self.register, width=20).pack(padx=6, pady=6)  
 tk.Button(self.frame, text="Правила игры", command=self.show\_rules, width=15).pack(padx=6, pady=10)  
  
 """Центрирует окно на экране."""  
 screen\_width = self.root.winfo\_screenwidth()  
 screen\_height = self.root.winfo\_screenheight()  
 x = (screen\_width - 350) // 2  
 y = (screen\_height - 325) // 2  
 self.root.geometry(f"{350}x{290}+{x}+{y}")  
  
 def login(self):  
 username = self.login\_entry.get()  
 password = self.password\_entry.get()  
 if authenticate(username, password):  
 messagebox.showinfo("Успех!", "Вы вошли в свой аккаунт")  
 self.root.destroy()  
 start\_checkers\_game()  
 else:  
 messagebox.showerror("Ошибка", "Неверный логин или пароль.")  
  
 def register(self):  
 username = self.login\_entry.get()  
 password = self.password\_entry.get()  
 if not username or not password:  
 messagebox.showerror("Ошибка", "Логин и пароль должны быть заполнены.")  
 elif check\_user\_exists(username):  
 messagebox.showerror("Ошибка", "Учетная запись уже существует.")  
 else:  
 add\_user(username, password)  
 messagebox.showinfo("Успех", "Регистрация успешно завершена.\nВойдите в аккаунт")  
  
 def show\_rules(self):  
 rules\_file = "правила шашек.pdf"  
 if os.path.exists(rules\_file):  
 os.startfile(rules\_file)  
 else:  
 messagebox.showerror("Ошибка", "Файл с правилами не найден.")  
  
class CheckersGame:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Турецкие шашки-поддавки")  
  
 self.button\_frame = tk.Frame(root).pack(pady=10)  
  
 self.turn\_label = tk.Label(text="Первыми ходят: Белые", font=("Arial", 16), bg="lightgray")  
 self.turn\_label.pack(side=tk.BOTTOM, anchor=tk.NE, padx=10, pady=10)  
  
 self.canvas = tk.Canvas(root, width=800, height=800)  
 self.canvas.pack()  
 self.game = Game(self.canvas, self.turn\_label)  
  
 root.state('zoomed')  
 self.canvas.bind("<Button-1>", self.on\_click)  
 self.update\_game()  
  
 def on\_click(self, event):  
 row, col = get\_row\_col(event)  
 self.game.select(row, col)  
  
def start\_checkers\_game():  
 root = tk.Tk()  
 CheckersGame(root)  
 root.mainloop()  
  
root = tk.Tk()  
AuthForm(root)  
root.mainloop()