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”» | | | | | | |
|  | | |  | | | | Руководитель разработки:  Кандидат технических наук, доцент кафедры «ИВК»  Шишкин Вадим Викторинович  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | |
|  | | | | Исполнитель  студент гр. ИСТбд-22  Линкевич А.К.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | |
|  | | 2024 | | | | | | |

Содержание

Аннотация 2

Техническое задание 3

Пояснительная записка. 10

Руководство программиста 16

Тестовая документация 36

Листинг программы 50

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”» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Линкевич А.К.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**Введение:**

Название программы: «Турецкие шашки».

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

*Доска и начальная расстановка*

Для игры в турецкие шашки 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*

* 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 смог уничтожить все шашки противника, либо лишить их возможности хода (“запереть”), либо тот, кто остался с несколькими своими простыми шашками против одной простой шашки противника.
* 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» 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», 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.

3) Простая и доступная панель управления, содержащая: кнопки «Правила», «Начать сначала», «Выход».

3. Система правил:

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

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”» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Линкевич А.К.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**Введение**

Название приложения «Турецкие шашки», условное обозначение «Türk daması». «Турецкие шашки» – 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.

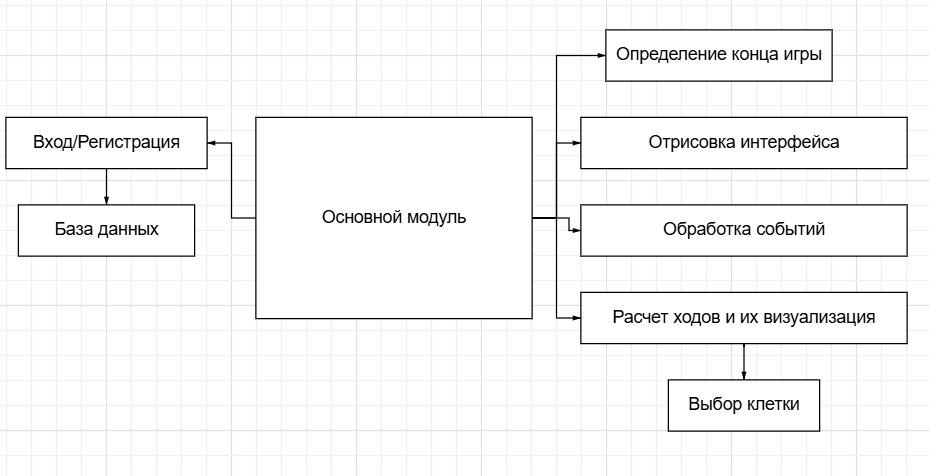


Рис.1. Архитектура приложения

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

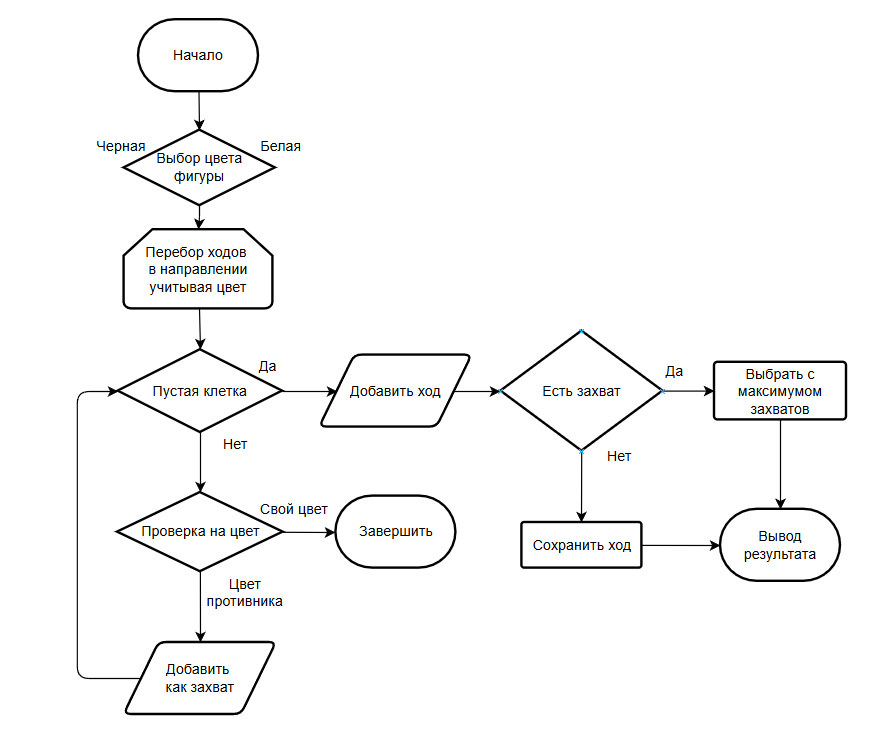


Рис.2 Алгоритм поиска хода

**1.3.3 Алгоритм входа/регистрации**

На рис. 3 представлен алгоритм входа/регистрации

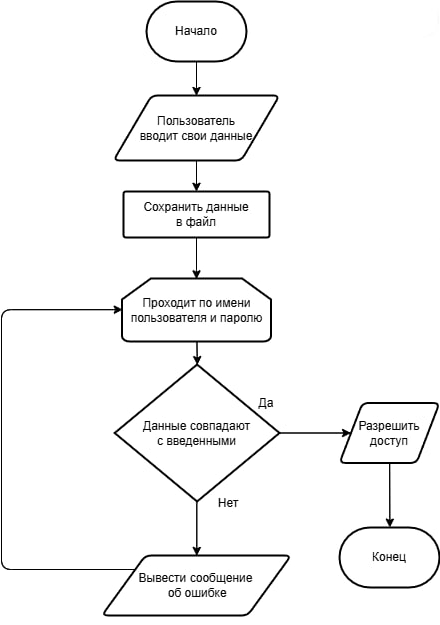


Рис. 3 Алгоритм входа/регистрации

**2. Источники, использованные при разработке**

* + - 1. Шишкин В.В., Афонин Д.С. Разработка логических компьютерных игр с графическим интерфейсом в среде питон. Ульяновск, УлГТУ 2023
      2. Вирт Н. Алгоритмы + структуры данных = программы: Пер. с англ. / Н. Вирт. – Москва : Мир, 1985. – 406 с.
      3. Турецкие шашки [Электронный ресурс]. – Режим доступа: URL: https://lotos-khv.ru/game/games/turk.pdf

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”» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Линкевич А.К.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

**Введение**

Данный проект представляет собой полноценную игру в “Турецкие шашки” 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.
* Возможность начать игру заново или посмотреть правила игры.
* Визуализация игрового поля с возможностью перемещения игровых элементов (кругов) 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;

Преимущества использования *tkinter:*

**Встроенность:** tkinter поставляется вместе с Python, поэтому его не нужно устанавливать отдельно, что делает его удобным и доступным.

**Кроссплатформенность:** Работает на большинстве популярных операционных систем, таких как Windows, macOS и Linux.

**Гибкость:** Подходит как для небольших скриптов, так и для более сложных приложений.

1. *json*  – используется для хеширования данных.

Преимущества использования *json:*

**Встроенность:** json поставляется вместе с Python, поэтому его не нужно устанавливать отдельно, что делает его удобным и доступным.

**Стандарт данных:** JSON — это легкий и широко используемый формат для обмена данными между приложениями.

**Кроссплатформенность:** Данные в формате JSON легко интегрируются с приложениями, написанными на других языках программирования.

1. *PIL* – для использования растровой графики.

Преимущества использования *PIL:*

**Редактирование:** Предоставляет функции для изменения размера, поворота, обрезки, наложения текста и других операций с изображениями.

**Совместимость:** Работает совместно с другими библиотеками Python, например, tkinter, для отображения изображений в интерфейсе.

**Обработка изображений:** Поддерживает множество форматов изображений (JPEG, PNG, BMP, GIF и другие).

1. *OS* – для предоставления работы с файловой системой компьютера.
   * + 1. **Характеристики программы.**

**2.1 Характеристика приложения.**

Количество строк программного кода: ~618 строк.

Реализовано 8 классов: *Resize, LoginRegisterApp, LoginForm, RegisterForm, CheckersApp, Board, Game, Piece.*

Код проекта разделен на несколько файлов и классов, которые отвечают за различные аспекты работы программы

* ‘main.py’: основной файл, отвечающий за запуск программы, и реализацию регистрации.
* ‘board.py’: файл, который содержит реализацию класса *Board*, отвечающий за отрисовку доски и фигур, проверку доступности ходов, проверку на окончание игры.
* ‘game.py’: отвечает за логику приложения, смену ходов, перезапуска игры, отображение правил и смену хода
* ‘constants.py’: содержит константы для игры: цвета фигур и клеток доски, размеры доски.
* ‘piece.py’: отвечает за фигуру на поле, ее позицию на поле, отрисовка фигуры и короны на ней, если фигура является дамкой.

Описание работы приложения:

* 1. При запуске файла ‘main.py’, приложение запускается с приветственного окна, предлагающего выбор: совершить вход или пройти регистрацию (рисунок 1).

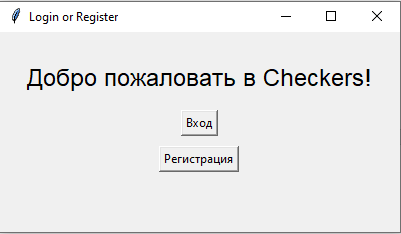


Рисунок 1 – Приветственное окно

* 1. При нажатии на кнопку “Вход”, пользователю предоставляется право ввести его имя и пароль для успешного входа. В случае если пользователь нажал на кнопку случайно, дается возможность вернуться назад(рисунок 2).

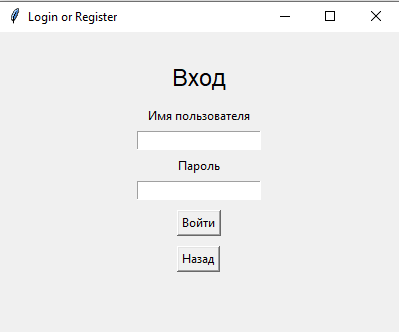


Рисунок 2 – Окно входа

* 1. Если у пользователя отсутствует учетная запись, то предоставляется возможность регистрации (рисунок 3).

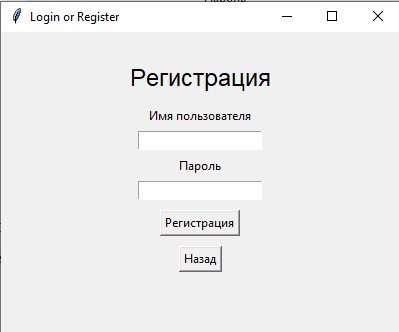


Рисунок 3 – Окно регистрации

* 1. После успешного входа или регистрации открывается игровое поле(рисунок 4).

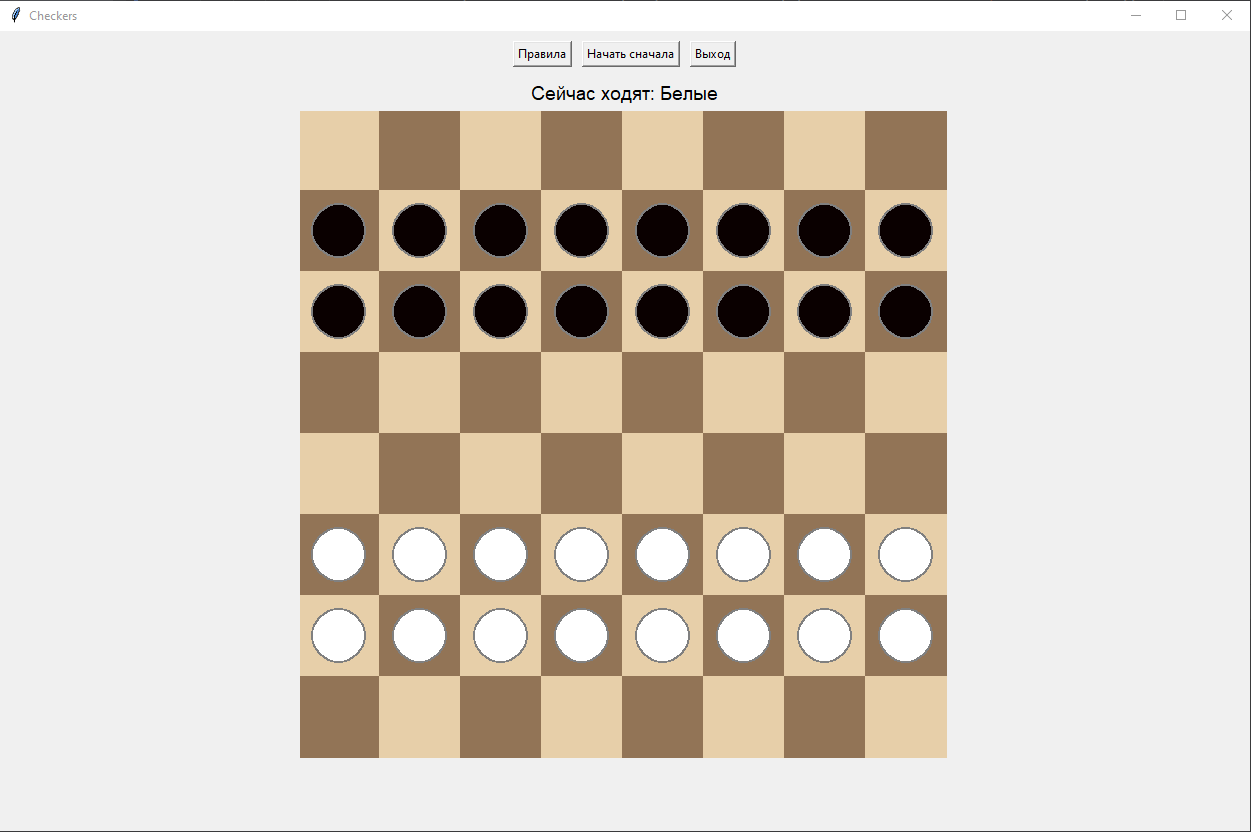


Рисунок 4 – Графический интерфейс игры

* 1. Пользователь поочередно с другим игроком (например другом), перемещает фигуры, следуя правилам игры (рисунок 5)

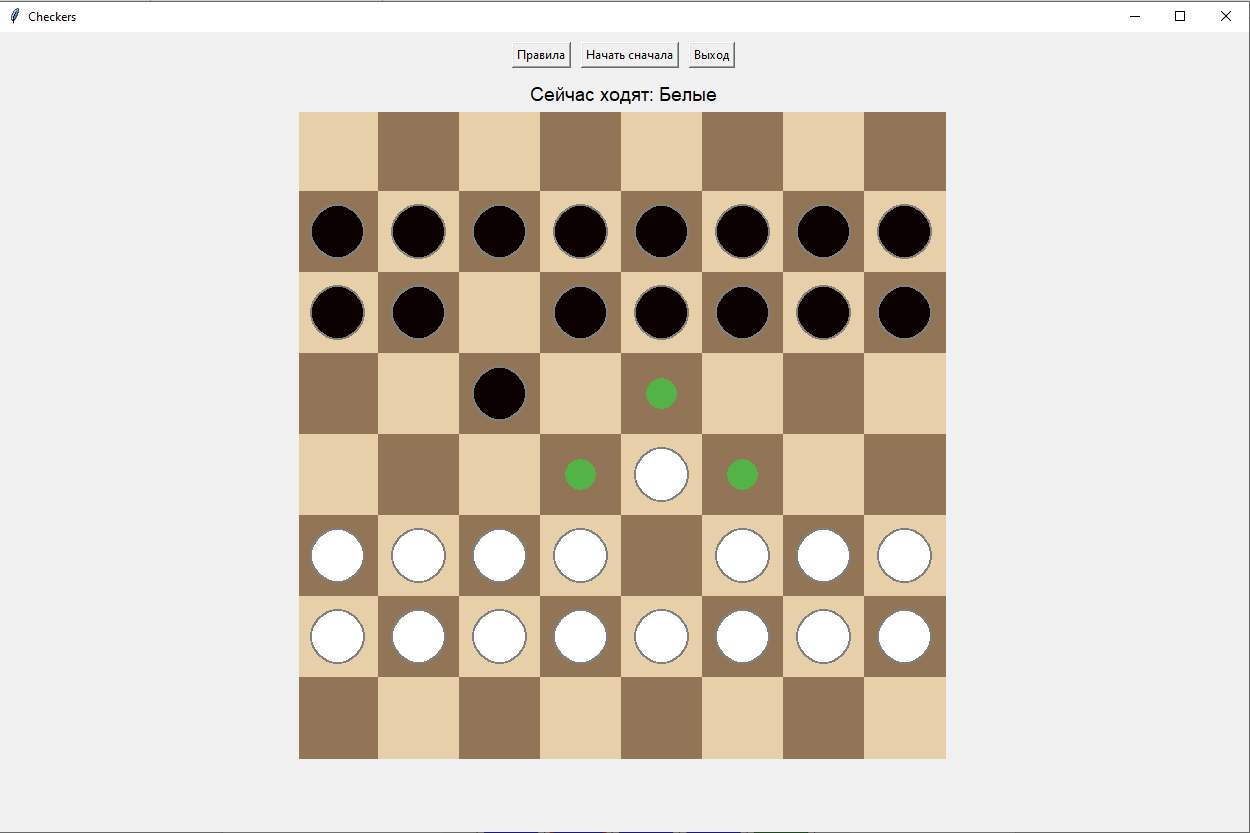
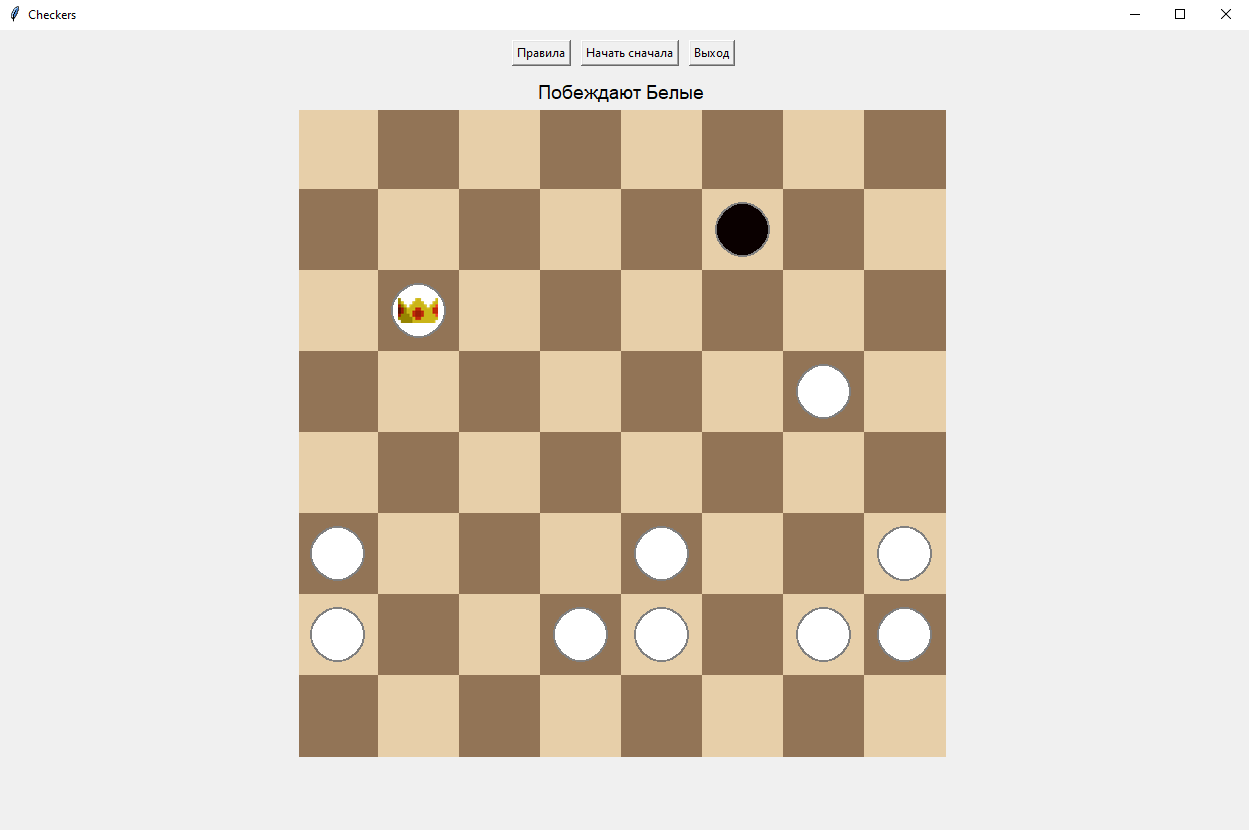


Рисунок 5 – Ходы пользователей

* 1. При завершении игры над игровым полем показывается сообщение о завершении игровой партии (рисунок 6)

  
Рисунок 6 – 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.**

### **Основные структуры данных и взаимодействие:**

Для файла “main.py”:

1. **Классы и наследование**
   * Код использует классы для организации логики. Классы Resize, LoginRegisterApp, LoginForm, RegisterForm, и CheckersApp являются ключевыми компонентами.
   * Наследование: Класс Resize предоставляет функциональность изменения размеров и центрирования окна, а остальные классы наследуют его (LoginRegisterApp, LoginForm, RegisterForm, CheckersApp), чтобы избежать дублирования кода.
   * Каждый из классов инкапсулирует определённую функциональность:
     + Resize: Управляет размером и положением окна.
     + LoginRegisterApp: Обеспечивает выбор между входом и регистрацией.
     + LoginForm: Реализует форму входа.
     + RegisterForm: Реализует форму регистрации.
     + CheckersApp: Отвечает за логику и интерфейс игры.
2. **Графический интерфейс (Tkinter)**
   * Используется библиотека tkinter для работы с графическим интерфейсом.
   * Основные элементы интерфейса:
     + Frame, Label, Button, Canvas, Entry.
   * Интерфейс управляется через методы классов, которые обрабатывают события (например, нажатия кнопок и клики мыши).
3. **Файловая система (JSON-файл)**
   * Для хранения данных пользователей (имя пользователя и пароль) используется файл users.json.
   * Методы:
     + validate\_user: Проверяет, существует ли пользователь и совпадает ли пароль.
     + add\_user: Добавляет нового пользователя в файл.
   * Эти методы реализуют взаимодействие между пользовательским интерфейсом и системой хранения данных.
4. **Игровая логика (Checkers)**
   * Класс CheckersApp управляет логикой игры в шашки.
   * Он взаимодействует с классом Game (предположительно, он определён где-то в другом месте кода) для управления состоянием игры.
   * Метод get\_row\_col\_from\_mouse определяет координаты клика на игровом поле и преобразует их в логические координаты (строка, столбец).
5. **События (Event Handling)**
   * Обработка событий реализована через методы, связанные с кнопками (command) и кликами мыши (<Button-1>).

Для файла “board.py”:

* + 1. **Матрица board (двумерный список):**
* Представляет игровое поле.
* Каждый элемент board[row][col] может быть либо экземпляром класса Piece (игровая фишка), либо 0 (пустое место).
* Используется для управления состоянием игрового поля: размещения, перемещения и удаления фишек.
  + 1. **Класс Piece:**
* Этот класс представляет отдельную фишку. Она имеет атрибуты, такие как row, col, color (цвет фишки) и статус королевы (king).
* Методы класса Piece используются для изменения позиции фишки и её преобразования в королеву.
  + 1. **Словари moves (например, в методах \_traverse\_\*):**
* Хранят возможные ходы для фишки в виде пар {(row, col): [skipped\_pieces]}, где ключ — координаты целевой клетки, а значение — список фишек, которые будут "перепрыгнуты" (съедены) в ходе этого хода.
  + 1. **Константы:**
* ROWS, COLS, SQUARE\_SIZE, BROWN, BROWN2 и цвета (BLACK, WHITE) задают параметры игрового поля, размера клеток и цвет фишек.
  + 1. **Объект canvas:**
* Используется для графического отображения игрового поля и фишек. Он взаимодействует с методами draw\_squares и draw.

Для файла “piece.py”:

* 1. **Класс Piece**:
* Представляет шашку на игровом поле.
* Свойства:
  + **row и col** — положение шашки в координатах доски.
  + **color** — цвет шашки (например, белый или чёрный).
  + **king** — булевый флаг, указывающий, является ли шашка дамкой.
  + **x и y** — экранные координаты шашки для графического интерфейса (рассчитываются через calc\_pos).
  + **crown\_image** — изображение короны, используемое для отображения дамки.
* Константы:
  + **PADDING** — отступ для визуального размера шашки (создаёт эффект пространства внутри клетки).
  + **OUTLINE** — толщина внешней обводки шашки.
  1. **Константы (SQUARE\_SIZE, GREY)**:
* SQUARE\_SIZE определяет размер клетки на доске.
* GREY используется для внешней обводки шашки.
  1. **Модуль tkinter**:
* Обеспечивает графическое отображение шашек (используется canvas для рисования).
  1. **Библиотека PIL**:
* Используется для обработки изображения короны, которая отображается на дамке.
  + - 1. **Обращение к программе.**

Методы файла ‘main.py’:

get\_row\_col\_from\_mouse(event) - Определяет строку и столбец на доске игры по координатам мыши.

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.

Методы класса *LoginRegisterApp:*

\_\_init\_\_(self, root) - Создает графический интерфейс с выбором между кнопками "Вход" и "Регистрация". Устанавливает заголовок окна и вызывает метод resize\_and\_center для удобного позиционирования. Включает приветственное сообщение и кнопки для навигации.

show\_login(self) - Вызывает класс LoginForm для отображения нового интерфейса. Уничтожает текущую рамку интерфейса (self.frame) и заменяет ее формой входа.

show\_register(self) - Вызывает класс RegisterForm для отображения нового интерфейса. Уничтожает текущую рамку интерфейса (self.frame) и заменяет ее формой регистрации.

start\_game(self) - Завершает текущее приложение (self.root.destroy) и вызывает start\_checkers\_game(). Закрывает текущее окно приложения и запускает игровой интерфейс "Турецкие шашки".

Методы класса *LoginForm*:

\_\_init\_\_(self, root, success\_callback) - Создает форму входа, включающую: Поля для ввода имени пользователя и пароля. Кнопки "Войти" и "Назад". Использует success\_callback для выполнения действий при успешном входе.

login(self) - Проверяет введенные имя пользователя и пароль. Если данные верны (проверка через validate\_user), вызывает success\_callback для перехода к следующему шагу (игра). Если данные неверны, отображает сообщение об ошибке. Использует messagebox для отображения всплывающих сообщений.

go\_back(self) - Уничтожает текущую рамку (self.frame) и вызывает класс LoginRegisterApp. Возвращает пользователя к начальному интерфейсу (выбор входа или регистрации).

Методы класса Register Form:

\_\_init\_\_(self, root, success\_callback) - Создает форму входа, включающую: Поля для ввода имени пользователя и пароля. Кнопки "Войти" и "Назад". Использует success\_callback для выполнения действий при успешной регистрации.

register(self) - Проверяет введенные данные: Если поля пусты, отображает сообщение об ошибке. Если имя пользователя уже занято, показывает ошибку. Если регистрация успешна, сохраняет данные (через add\_user) и вызывает success\_callback. Использует messagebox для уведомлений об успешной регистрации или ошибках.

go\_back(self) - Уничтожает текущую рамку (self.frame) и вызывает класс LoginRegisterApp. Возвращает пользователя к начальному интерфейсу (выбор входа или регистрации).

Методы класса CheckersApp:

\_\_init\_\_(self, root) - Создает интерфейс игры, включая: Холст для игрового поля. Метку с указанием текущего хода. Кнопки для отображения правил, сброса игры и выхода. Устанавливает размеры окна через resize\_and\_center.

on\_click(self, event) - Использует функцию get\_row\_col\_from\_mouse 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).

update\_game(self) - Использует метод self.game.update для обновления графики и проверки текущего состояния. Обновляет состояние игры: Если есть победитель, отображает его. Если игра продолжается, обновляет графику и планирует следующее обновление через 60 кадров в секунду.

quit(self) - Закрывает окно приложения.

Методы для работы с пользователем:

validate\_user(username, password) - Сравнивает введенный пароль с паролем из файла. Проверяет, существует ли пользователь с указанным именем и паролем в файле users.json

add\_user(username, password) - Регистрирует нового пользователя в файле users.json. Если пользователь уже существует, возвращает False. Если имя свободно, добавляет запись и сохраняет файл.

Методы для запуска:

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

main() - Запускает начальный интерфейс с выбором входа или регистрации. Создает объект класса LoginRegisterApp и запускает главный цикл Tk.

Методы класса *Board* файла ‘board.py’:

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

create\_board(self) - Создает начальную расстановку шашек на доске. Все ячейки доски инициализируются объектами Piece или 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\_squares(self) - Рисует шахматную доску на холсте. Использует два цвета (BROWN и BROWN2) для чередующихся клеток. Полностью очищает холст перед перерисовкой, чтобы избежать наложений.

draw(self) - Рисует все шашки на доске поверх клеток, созданных методом draw\_squares. Обходит каждый элемент self.board и вызывает метод draw для каждого объекта Piece.

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\_king). Увеличивает счетчик дамок соответствующего цвета (self.white\_kings или self.black\_kings).

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

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

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

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.

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

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

\_traverse\_vertical\_king(self, start, stop, step, color, col, skipped=[]) - Проверяет возможные ходы дамки вдоль вертикали. Учитывает полную свободу движения дамки (без ограничения на количество клеток).

\_traverse\_horizontal\_king(self, start, stop, step, color, row, skipped=[]) - Проверяет возможные ходы дамки вдоль горизонтали.

Методы класса *Game* файла ‘game.py’:

reset(self) - Сбрасывает состояние игры до начального. Вызывает метод \_init для инициализации доски и других игровых параметров. Сбрасывает текущий ход на белых и обновляет текст метки turn\_label.

select(self, row, col) - Обрабатывает выбор шашки и попытку её перемещения. Если шашка уже выбрана, вызывает метод \_move 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.

\_move(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.

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

change\_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.

Методы класса *Piece* файла ‘piece.py’:

\_\_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. Флаг king определяет, является ли шашка дамкой. Рассчитывает пиксельные координаты для отображения на доске с помощью calc\_pos. Загружает и изменяет размер изображения короны для отображения на дамке.

calc\_pos(self) - Вычисляет пиксельные координаты центра шашки на основе её строки и столбца. Центр клетки вычисляется с использованием размера клетки (SQUARE\_SIZE) 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 calc\_pos.

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, поверх неё добавляется изображение короны.

make\_king(self) - Преобразует шашку в дамку. Устанавливает флаг king в True.

* + - 1. **Сообщения.**

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

На рисунке 7 представлено окно ошибки при попытке входа без введенных данных пользователя.

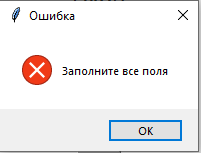


Рисунок 7 – Ошибка “Заполните все поля”

На рисунке 8 представлено окно ошибки при попытке ввода неправильных данных.

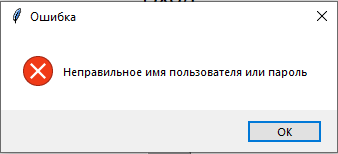


Рисунок 8 – Ошибка “Неправильное имя пользователя или пароль”

На рисунке 9 представлено окно ошибки при попытке пройти регистрацию с пустыми полями.

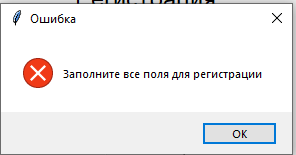


Рисунок 9 – Ошибка “Заполните все поля для регистрации”

На рисунке 10 представлено окно об успешной регистрации.

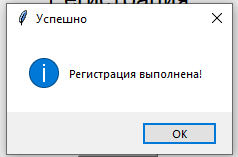


Рисунок 10 – “Регистрация выполнена!”

На рисунке 11 представлена ошибка при попытке зарегистрировать пользователя с уже существующем именем.

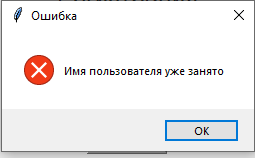


Рисунок 10 – “Имя пользователя уже занято”

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

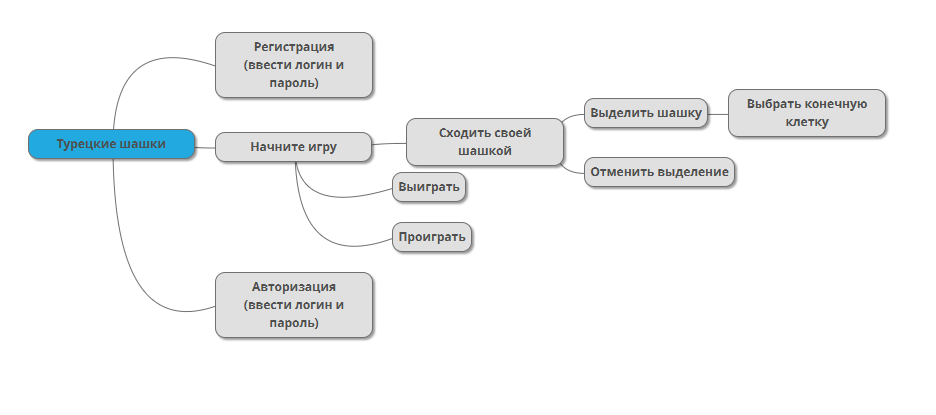
Кафедра «Измерительно-вычислительные комплексы»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  |  |
| |  |  | | --- | --- | | ***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”» | | | | | | |
|  | | |  | |  | | | |
|  | | Исполнитель  студент гр. ИСТбд-22  Линкевич А.К.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | 2024 | | | | | | |

* 1. **Введение.**

Турецкие шашки — 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. В ней детализированы все аспекты проверки: от базовой проверки правил игры до тестирования сложных стратегических ситуаций, пользовательского интерфейса и производительности.



Mind-map “Турецких шашек”

* 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** |
| --- | --- | --- | --- | --- |
| TK1 | 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 |
| ТК2 | 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 |
| ТК3 | 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 |
| ТК4 | 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 |
| ТК5 | Преобразование в дамку | 1)Запустить программу  2)Довести одну из шашек до последней (0 или 7 по индексу) горизонтали.  3)Проверить появилась ли картинка короны на шашке. | При достижении нужной горизонтали на обычной шашке появится корона | Пройден |
| ТК6 | 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 |
| ТК7 | 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 |
| ТК8 | 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 |
| ТК9 | Кнопка “Правила” | 1)Запустить программу  2)Нажать на кнопку “Правила” | Появляется окно с правилами игры | Пройден |
| ТК10 | Кнопка “Начать сначала” | 1)Запустить программу  2)Нажать на кнопку “Начать сначала” | Текущее состояние доски сбрасывается до начального состояния | Пройден |
| ТК11 | Кнопка “Выход” | 1)Запустить программу  2)Нажать на кнопку “Выход” | Приложение закроется | Пройден |
| ТК12 | 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 |
| ТК13 | Проверка на совершение невозможного хода | 1)Запустить программу  2)Выбрать шашку  3)Попробовать переместить на любую клетку кроме подсвеченной | Невозможно совершить ход | Пройден |
| ТК14 | 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 |
| ТК15 | Проверка победы одной из сторон | 1)Запустить программу  2)Захватить все шашки противника или оставить одну | Появится сообщение о победы одной из сторон | Пройден |
| ТК16 | Проверка на ничью | 1)Запустить программу  2)Довести игру до возникновения ничьи | Появляется сообщение о ничьей | Пройден |
| ТК17 | Проверка на клик по пустой клетке | 1)Запустить программу  2)Нажать на клетку на доске | Появится сообщение о том, что нужно выбрать шашку определенного цвета | Пройден |
| ТК18 | Регистрация нового пользователя | 1)Запустить программу  2)Нажать на кнопку регистрации 3)Ввести данные требующиеся от пользователя  4)Войти в приложение | Появится сообщение о том, что пользователь зарегистрирован | Пройден |
| ТК19 | Получение ошибки о регистрации | 1)Запустить программу  2)Нажать на кнопку регистрации  3)Нажать на кнопку зарегистри-роваться  не вводя данные | Появится сообщение о том, что нужно заполнить данные | Пройден |
| ТК20 | Попытка регистрации с уже существующем именем пользователя | 1)Запустить программу  2)Нажать на кнопку регистрации  3)Попробовать ввести уже существующее имя пользователя (предварительно  зарегистрировать) | Появится сообщение о том, что пользователь с таким именем уже есть | Пройден |
| ТК21 | Попытка успешного входа | 1)Запустить программу  2)Нажать на кнопку входа  3)Попробовать ввести уже существующее данные пользователя (предварительно  зарегистрировать) | Откроется окно игры | Пройден |
| ТК22 | Попытка неуспешного входа | 1)Запустить программу  2)Нажать на кнопку входа  3)Попробовать ввести несуществующее данные пользователя | Появляется ошибка о том, что имя пользователя или пароль неверные | Пройден |

**Чек-лист для формы регистрации**

|  |  |  |
| --- | --- | --- |
| **Описание** | **Пример** | **Результат** |
| Проверка поля “имя пользователя”/ ”пароль” |  |  |
| Ввод логина/пароля | Test/Test | Сообщение об успешной регистрации |
| Пустые строки |  | Сообщение об ошибке |
| Ввод существующего имени пользователя | Test | Сообщение об ошибке |

**Чек-лист для формы входа**

|  |  |  |
| --- | --- | --- |
| **Описание** | **Пример** | **Результат** |
| Проверка поля “имя пользователя”/ ”пароль” |  |  |
| Ввод логина/пароля | Test/Test | Успешный вход |
| Пустые строки |  | Сообщение об ошибке |
| Ввод несуществующего имени пользователя | User | Сообщение об ошибке |
| Ввод неверного пароля | User123 | Сообщение об ошибке |

* + - 1. **Выводы**

На основе проведенных тестов сделаны следующие выводы:

* Программа успешно прошла все тесты и работает корректно
* Рекомендации по дальнейшему развитию программы: добавление ограничения времени на ход пользователя, звуковое сопровождение.

**Листинг программы**

**Листинг файла “main.py”**

|  |
| --- |
| import tkinter as tk  from checkers.constants import WIDTH, HEIGHT, SQUARE\_SIZE  from checkers.game import Game  from tkinter import messagebox  import json  import os  def get\_row\_col\_from\_mouse(event):  x, y = event.x, event.y  row = y // SQUARE\_SIZE  col = x // SQUARE\_SIZE  return row, col  class Resize:  def \_\_init\_\_(self, root):  self.root = root  def resize\_and\_center(self, window\_width, window\_height):  screen\_width = self.root.winfo\_screenwidth()  screen\_height = self.root.winfo\_screenheight()  x\_coordinate = int((screen\_width / 2) - (window\_width / 2))  y\_coordinate = int((screen\_height / 2) - (window\_height / 2))  self.root.geometry("{}x{}+{}+{}".format(window\_width, window\_height, x\_coordinate, y\_coordinate))  class LoginRegisterApp(Resize):  def \_\_init\_\_(self, root):  super().\_\_init\_\_(root)  self.root.title("Login or Register")  self.frame = tk.Frame(root)  self.frame.pack(pady=20)  self.title\_label = tk.Label(self.frame, text="Добро пожаловать в Checkers!", font=("Arial", 18))  self.title\_label.pack(pady=10)  self.login\_button = tk.Button(self.frame, text="Вход", command=self.show\_login)  self.login\_button.pack(pady=5)  self.register\_button = tk.Button(self.frame, text="Регистрация", command=self.show\_register)  self.register\_button.pack(pady=5)  self.resize\_and\_center(400, 200)  def show\_login(self):  self.frame.destroy()  LoginForm(self.root, self.start\_game)  def show\_register(self):  self.frame.destroy()  RegisterForm(self.root, self.start\_game)  def start\_game(self):  self.root.destroy()  start\_checkers\_game()  class LoginForm(Resize):  def \_\_init\_\_(self, root, success\_callback):  super().\_\_init\_\_(root)  self.success\_callback = success\_callback  self.frame = tk.Frame(root)  self.frame.pack(pady=20)  self.title\_label = tk.Label(self.frame, text="Вход", font=("Arial", 18))  self.title\_label.pack(pady=10)  self.username\_label = tk.Label(self.frame, text="Имя пользователя")  self.username\_label.pack()  self.username\_entry = tk.Entry(self.frame)  self.username\_entry.pack(pady=5)  self.password\_label = tk.Label(self.frame, text="Пароль")  self.password\_label.pack()  self.password\_entry = tk.Entry(self.frame, show="\*")  self.password\_entry.pack(pady=5)  self.login\_button = tk.Button(self.frame, text="Войти", command=self.login)  self.login\_button.pack(pady=5)  self.back\_button = tk.Button(self.frame, text="Назад", command=self.go\_back)  self.back\_button.pack(pady=5)  self.resize\_and\_center(400, 300)  def login(self):  username = self.username\_entry.get()  password = self.password\_entry.get()  if username and password:  if validate\_user(username, password):  self.success\_callback()  else:  messagebox.showerror("Ошибка", "Неправильное имя пользователя или пароль")  else:  messagebox.showerror("Ошибка", "Заполните все поля")  def go\_back(self):  self.frame.destroy()  LoginRegisterApp(self.root)  class RegisterForm(Resize):  def \_\_init\_\_(self, root, success\_callback):  super().\_\_init\_\_(root)  self.success\_callback = success\_callback  self.frame = tk.Frame(root)  self.frame.pack(pady=20)  self.title\_label = tk.Label(self.frame, text="Регистрация", font=("Arial", 18))  self.title\_label.pack(pady=10)  self.username\_label = tk.Label(self.frame, text="Имя пользователя")  self.username\_label.pack()  self.username\_entry = tk.Entry(self.frame)  self.username\_entry.pack(pady=5)  self.password\_label = tk.Label(self.frame, text="Пароль")  self.password\_label.pack()  self.password\_entry = tk.Entry(self.frame, show="\*")  self.password\_entry.pack(pady=5)  self.register\_button = tk.Button(self.frame, text="Регистрация", command=self.register)  self.register\_button.pack(pady=5)  self.back\_button = tk.Button(self.frame, text="Назад", command=self.go\_back)  self.back\_button.pack(pady=5)  self.resize\_and\_center(400, 300)  def register(self):  username = self.username\_entry.get()  password = self.password\_entry.get()  if username and password:  if add\_user(username, password):  messagebox.showinfo("Успешно", "Регистрация выполнена!")  self.success\_callback()  else:  messagebox.showerror("Ошибка", "Имя пользователя уже занято")  else:  messagebox.showerror("Ошибка", "Заполните все поля для регистрации")  def go\_back(self):  self.frame.destroy()  LoginRegisterApp(self.root)  class CheckersApp(Resize):  def \_\_init\_\_(self, root):  super().\_\_init\_\_(root)  self.root.title("Checkers")  self.button\_frame = tk.Frame(root)  self.button\_frame.pack(pady=10)  self.turn\_label = tk.Label(text="Сейчас ходят: Белые", font=("Arial", 14))  self.turn\_label.pack(pady=2)  self.canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT)  self.canvas.pack()  self.game = Game(self.canvas, self.turn\_label)  self.rules\_button = tk.Button(self.button\_frame, text="Правила", command=self.game.show\_rules)  self.rules\_button.pack(side=tk.LEFT, padx=5)  self.reset\_button = tk.Button(self.button\_frame, text="Начать сначала", command=self.game.reset)  self.reset\_button.pack(side=tk.LEFT, padx=5)  self.reset\_button = tk.Button(self.button\_frame, text="Выход", command=self.quit)  self.reset\_button.pack(side=tk.LEFT, padx=5)  self.resize\_and\_center(1250, 800)  self.canvas.bind("<Button-1>", self.on\_click)  self.update\_game()  def on\_click(self, event):  row, col = get\_row\_col\_from\_mouse(event)  self.game.select(row, col)  def update\_game(self):  if self.game.winner() is not None:  self.game.update()  self.turn\_label.config(text=f"{self.game.winner()} ")  else:  self.game.update()  self.root.after(1000 // 60, self.update\_game)  def quit(self):  self.root.quit()  def validate\_user(username, password):  if not os.path.exists("users.json"):  return False  with open("users.json", "r") as file:  users = json.load(file)  return users.get(username) == password  def add\_user(username, password):  if not os.path.exists("users.json"):  users = {}  else:  with open("users.json", "r") as file:  users = json.load(file)  if username in users:  return False  users[username] = password  with open("users.json", "w") as file:  json.dump(users, file)  return True  def start\_checkers\_game():  root = tk.Tk()  CheckersApp(root)  root.mainloop()  def main():  root = tk.Tk()  LoginRegisterApp(root)  root.mainloop()  main() |

**Листинг файла “board.py”**

|  |
| --- |
| from .constants import BROWN, ROWS, BROWN2, SQUARE\_SIZE, COLS, WHITE, BLACK  from .piece import Piece  class Board:  def \_\_init\_\_(self, canvas):  self.canvas = canvas  self.board = []  self.white\_left = self.black\_left = 16  self.white\_kings = self.black\_kings = 0  self.create\_board()  def draw\_squares(self):  self.canvas.delete("all")  for row in range(ROWS):  for col in range(COLS):  color = BROWN2 if (row + col) % 2 == 1 else BROWN  x0 = col \* SQUARE\_SIZE  y0 = row \* SQUARE\_SIZE  x1 = x0 + SQUARE\_SIZE  y1 = y0 + SQUARE\_SIZE  self.canvas.create\_rectangle(x0, y0, x1, y1, fill=color, outline=color)  def move(self, piece, row, col):  self.board[piece.row][piece.col], self.board[row][col] = self.board[row][col], self.board[piece.row][piece.col]  piece.move(row, col)  if row == ROWS - 1 or row == 0:  piece.make\_king()  if piece.color == BLACK:  self.black\_kings += 1  else:  self.white\_kings += 1  def get\_piece(self, row, col):  return self.board[row][col]  def create\_board(self):  for row in range(ROWS):  self.board.append([])  for col in range(COLS):  if 1<=row<=2:  self.board[row].append(Piece(row, col, BLACK))  elif 5<=row<=6:  self.board[row].append(Piece(row, col, WHITE))  else:  self.board[row].append(0)  def draw(self):  self.draw\_squares()  for row in range(ROWS):  for col in range(COLS):  piece = self.board[row][col]  if piece != 0:  piece.draw(self.canvas)  def remove(self, pieces):  for piece in pieces:  self.board[piece.row][piece.col] = 0  if piece != 0:  if piece.color == WHITE:  self.white\_left -= 1  else:  self.black\_left -= 1  def winner(self):  if self.white\_left <= 1:  return "Побеждают Черные"  elif self.black\_left <= 1:  return "Побеждают Белые"  elif self.white\_left == 1 and self.black\_left == 1:  return "Ничья"  return None  def get\_valid\_moves(self, piece):  moves = {}  row = piece.row  col = piece.col  if piece.color == WHITE and not piece.king:  start = row - 1  stop = max(row - 3, -1)  moves.update(self.\_traverse\_vertical(start, stop, -1, piece.color, col))  moves.update(self.\_traverse\_horizontal(col - 1, -1, -1, piece.color, row))  moves.update(self.\_traverse\_horizontal(col + 1, COLS, 1, piece.color, row))  if piece.color == BLACK and not piece.king:  moves.update(self.\_traverse\_vertical(row + 1, min(row + 3, ROWS), 1, piece.color, col))  moves.update(self.\_traverse\_horizontal(col - 1, -1, -1, piece.color, row))  moves.update(self.\_traverse\_horizontal(col + 1, COLS, 1, piece.color, row))  if piece.king:  moves.update(self.\_traverse\_vertical\_king(row + 1, ROWS, 1, piece.color, col))  moves.update(self.\_traverse\_vertical\_king(row - 1, -1, -1, piece.color, col))  moves.update(self.\_traverse\_horizontal\_king(col - 1, -1, -1, piece.color, row))  moves.update(self.\_traverse\_horizontal\_king(col + 1, COLS, 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())  capture\_moves = {pos: skipped for pos, skipped in capture\_moves.items() if len(skipped) == max\_captures}  return capture\_moves  return moves  def \_traverse\_vertical(self, start, stop, step, color, col, skipped=[]):  moves = {}  last = []  for r in range(start, stop, step):  current = self.board[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:  row\_limit = max(r - 3, -1) if step == -1 else min(r + 3, ROWS)  new\_skipped = skipped + last  moves.update(self.\_traverse\_vertical(r + step, row\_limit, step, color, col, skipped=new\_skipped))  moves.update(self.\_traverse\_horizontal(col - 1, -1, -1, color, r, skipped=new\_skipped))  moves.update(self.\_traverse\_horizontal(col + 1, COLS, 1, color, r, skipped=new\_skipped))  break  elif current.color == color:  break  else:  if last:  break  last = [current]  return moves  def \_traverse\_horizontal(self, start, stop, step, color, row, skipped=[]):  moves = {}  last = []  for c in range(start, stop, step):  current = self.board[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:  col\_limit = max(c - 3, -1) if step == -1 else min(c + 3, COLS)  new\_skipped = skipped + last  moves.update(self.\_traverse\_horizontal(c + step, col\_limit, step, color, row, skipped=new\_skipped))  moves.update(self.\_traverse\_vertical(row - 1, max(row - 3, -1), -1, color, c, skipped=new\_skipped))  moves.update(self.\_traverse\_vertical(row + 1, min(row + 3, ROWS), 1, color, c, skipped=new\_skipped))  break  elif current.color == color:  break  else:  if last:  break  last = [current]  return moves  def \_traverse\_vertical\_king(self, start, stop, step, color, col, skipped=[]):  moves = {}  last = []  for r in range(start, stop, step):  current = self.board[r][col]  if current == 0:  if skipped:  moves[(r, col)] = skipped + last  else:  moves[(r, col)] = last  if last:  row\_limit = -1 if step == -1 else ROWS  new\_skipped = skipped + last  moves.update(self.\_traverse\_vertical(r + step, row\_limit, step, color, col, skipped=new\_skipped))  moves.update(self.\_traverse\_horizontal(col - 1, -1, -1, color, r, skipped=new\_skipped))  moves.update(self.\_traverse\_horizontal(col + 1, COLS, 1, color, r, skipped=new\_skipped))  elif current.color == color:  break  else:  if last:  break  last = [current]  return moves  def \_traverse\_horizontal\_king(self, start, stop, step, color, row, skipped=[]):  moves = {}  last = []  for c in range(start, stop, step):  current = self.board[row][c]  if current == 0:  if skipped:  moves[(row, c)] = skipped + last  else:  moves[(row, c)] = last  if last:  col\_limit = -1 if step == -1 else COLS  new\_skipped = skipped + last  moves.update(self.\_traverse\_horizontal(c + step, col\_limit, step, color, row, skipped=new\_skipped))  moves.update(self.\_traverse\_vertical(row - 1, -1, -1, color, c, skipped=new\_skipped))  moves.update(self.\_traverse\_vertical(row + 1, ROWS, 1, color, c, skipped=new\_skipped))  elif current.color == color:  break  else:  if last:  break  last = [current]  return moves |

**Листинг файла “game.py”**

|  |
| --- |
| from .constants import WHITE, GREEN, SQUARE\_SIZE, BLACK  from checkers.board import Board  from tkinter import messagebox  class Game:  def \_\_init\_\_(self, canvas, turn\_label):  self.canvas = canvas  self.turn\_label = turn\_label  self.\_init()  def show\_rules(self):  messagebox.showinfo("Правила игры", "Правила игры:\n"  "1. Простая шашка ходит на одно поле вперёд, влево, вправо. \n"  "2. Простая шашка бьёт шашку противника, стоящую спереди, справа или слева (бить назад запрещено), перескакивая через неё на следующее поле по вертикали или горизонтали. \n"  "3. 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). \n"  "4. Если есть несколько вариантов боя, игрок обязан выбрать тот, при котором берётся наибольшее количество шашек противника. \n"  "5. Простая шашка, вступившая на восьмую горизонталь, становится дамкой. \n"  "6. Дамка ходит на любое количество пустых полей вперёд, назад, вправо, влево. \n"  "7. Дамка бьёт шашки противника, стоящие от неё через любое количество пустых клеток спереди, сзади, справа и слева, если следующее за шашкой поле свободно. \n"  "8. Выигрывает тот, кто смог уничтожить все шашки противника, либо тот, кто остался с несколькими своими простыми шашками против одной простой шашки противника. \n"  "9. Если на доске осталось по одной шашке — объявляется ничья.")  def update(self):  self.board.draw()  self.draw\_valid\_moves(self.valid\_moves)  def \_init(self):  self.selected = None  self.board = Board(self.canvas)  self.turn = WHITE  self.valid\_moves = {}  def winner(self):  return self.board.winner()  def reset(self):  self.\_init()  self.turn\_label.config(text="Сейчас ходят: Белые")  self.update()  def select(self, row, col):  if self.selected:  result = self.\_move(row, col)  if not result:  self.selected = None  self.select(row, col)  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)  return True  if self.selected != piece and self.turn == WHITE:  self.turn\_label.config(text="Выберите шашку БЕЛОГО цвета")  if self.selected != piece and self.turn == BLACK:  self.turn\_label.config(text="Выберите шашку ЧЕРНОГО цвета")  return False  def \_move(self, row, col):  piece = self.board.get\_piece(row, col)  if self.selected and piece == 0 and (row, col) in self.valid\_moves:  self.board.move(self.selected, row, col)  skipped = self.valid\_moves[(row, col)]  if skipped:  self.board.remove(skipped)  self.change\_turn()  else:  return False  return True  def draw\_valid\_moves(self, moves):  for move in moves:  row, col = move  x = col \* SQUARE\_SIZE + SQUARE\_SIZE // 2  y = row \* SQUARE\_SIZE + SQUARE\_SIZE // 2  self.canvas.create\_oval(x - 15, y - 15, x + 15, y + 15, fill=GREEN, outline=GREEN)  def change\_turn(self):  self.valid\_moves = {}  if self.turn == WHITE:  self.turn = BLACK  self.turn\_label.config(text="Сейчас ходят: Черные")  else:  self.turn = WHITE  self.turn\_label.config(text="Сейчас ходят: Белые") |

**Листинг файла “piece.py”**

|  |
| --- |
| from .constants import SQUARE\_SIZE, GREY  from PIL import Image, ImageTk  import tkinter as tk  class Piece:  PADDING = 15  OUTLINE = 2  def \_\_init\_\_(self, row, col, color):  self.row = row  self.col = col  self.color = color  self.king = False  self.x = 0  self.y = 0  self.calc\_pos()  crown\_image = Image.open('crown.png')  crown\_image = crown\_image.resize((40, 25), Image.Resampling.LANCZOS)  self.crown\_image = ImageTk.PhotoImage(crown\_image)  def calc\_pos(self):  self.x = SQUARE\_SIZE \* self.col + SQUARE\_SIZE // 2  self.y = SQUARE\_SIZE \* self.row + SQUARE\_SIZE // 2  def make\_king(self):  self.king = True  def draw(self, canvas):  radius = SQUARE\_SIZE // 2 - self.PADDING  canvas.create\_oval(  self.x - radius - self.OUTLINE,  self.y - radius - self.OUTLINE,  self.x + radius + self.OUTLINE,  self.y + radius + self.OUTLINE,  fill=GREY,  outline=""  )  canvas.create\_oval(  self.x - radius,  self.y - radius,  self.x + radius,  self.y + radius,  fill=self.color,  outline=""  )  if self.king:  canvas.create\_image(self.x, self.y, image=self.crown\_image, anchor=tk.CENTER)  def move(self, row, col):  self.row = row  self.col = col  self.calc\_pos() |