МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема Компьютерная логическая игра Турецкие шашки-поддавки

Р.02069337.22/2411-12 ТЗ-01

Листов 41

Руководитель разработки:
кандидат технических наук,
доцент кафедры
«Измерительно-
вычислительные
комплексы»,
Шишкин Вадим
Викторинович
«____» _____ 2024
г.

Исполнитель:
студент гр. ИСТбд-21
Прокофьев Сергей
Александрович
«____» _____ 2024
г.

2024

Полп. и

Инв

Вза

Полп. и

Инв.

Содержание

Аннотация

Тема курсового проекта: Компьютерная логическая игра «Турецкие шашки-поддавки»

Исполнитель: студент гр. ИСТбд-21 Прокофьев Сергей Александрович

Руководитель разработки: Шишкин Вадим Викторинович

Работа состоит из технического задания, пояснительной записки,

руководства программиста и текста программы.

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

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

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

Текст программы представляет собой полный код программы,

реализующий компьютерную логическую игру «Турецкие шашки-поддавки».

Ключевые слова: компьютерная игра, Турецкие шашки, поддавки, python.

Ключевые функции приложения:

1. Авторизация/регистрация пользователя.
2. Отображение игрового поля.
3. Проверка ходов игрока.
4. Объявление результатов игры.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема Компьютерная логическая игра Турецкие шашки-поддавки

Р.02069337.22/2411-12 ТЗ-01

Листов 5

Исполнитель:
студент гр. ИСТбд-21
Прокофьев Сергей Александрович
«____» _____ 2024 г.

2024

Полп. и
Инв
Вза
Полп. и
Инв.

Введение

Компьютерная логическая игра «Турецкие шашки – Поддавки» Приложение должно соответствовать правилам игры, приведенным ниже.

Поле и игроки. Играют двое игроков. Игра ведется на шашечном поле. Шашечная доска состоит из 64 (8x8) одинаковых клеток. Традиционно используется разлинованная одноцветная доска.

Шашки. Шашки подразделяются на простые и дамки. Дамки маркируются знаком короны. Перед началом игры игрокам предоставляется по 16 простых шашек: одному — белых, другому — чёрных.

Стартовая позиция. Шашки расставляются на второй и третьей от игрока горизонталях, по 8 шашек в ряд, при этом первая от игрока горизонталь остается свободной.

Ходы. Ходом в партии считается передвижение шашки с одной клетки поля на другое. Первый ход всегда делает игрок, играющий светлыми. Игроки поочередно делают по одному ходу до тех пор, пока игра не закончится. Шашки разделяются на простые шашки и дамки. Простая шашка ходит на одно поле вперёд, влево, вправо. В случае, когда простая шашка, вступившая на восьмую горизонталь (считая от себя), превращается в дамку и получает новые права. 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. Дамка обозначается символом короны. Дамка, в отличие от простой шашки, ходит на любое количество пустых полей вперёд, назад, вправо, влево, но становиться может, как и простая шашка, лишь на не занятые другими шашками клетки, причем через свои шашки она перескакивать не может.

Взятие. Если простая шашка находится на одной диагонали рядом с шашкой другого игрока, за которой имеется свободная клетка, она должна быть перенесена через эту шашку на свободную клетку. Если с новой позиции

шашки, побившей шашку противника, можно бить дальше, взятие 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. Шашки другого игрока в этом случае снимаются с поля. 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. Шашка другого игрока в этом случае снимается с поля. Взятие шашки другого игрока является обязательным и производится как вперед, так и назад. 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°. Взятие своих шашек. После завершения взятия взятые шашки другого игрока поочерёдно снимаются с доски в порядке их взятия. Это называется последовательным взятием. В процессе последовательного взятия запрещается переносить шашки через свои собственные. В процессе последовательного взятия разрешается проходить несколько раз через одну и ту же клетку, но запрещается переносить свою шашку через одну и ту же шашку другого игрока более одного раза. При возможности взятия по двум и более направлениям дамкой или шашкой выбор, вне зависимости от количества или качества снимаемых шашек (дамки или простой), предоставляется берущему. Если простая шашка при взятии достигает последнего (восьмого от себя) горизонтального ряда и если ей предоставляется возможность дальнейшего взятия шашек, то она обязана тем же ходом продолжать взятие, но уже на правах дамки. Если же простая шашка достигает последнего горизонтального ряда без взятия и ей после этого предоставляется возможность взятия, то она должна брать (если эта возможность сохранится) лишь следующим ходом на правах дамки.

Выигрыш партии. Выигравшим партию признается тот, кто первым достигнет положения, при котором: у игрока не осталось ни одной шашки.

1. Основания для разработки

В качестве оснований для разработки указывается учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету.

2. Требования к программе или программному изделию

2.1. Функциональное назначение

Требуется разработать однопользовательское десктопное приложение по игре в шашки с графическим интерфейсом в среде Windows.

2.2 Требования к функциональным характеристикам

2.2.1 Требования к структуре приложения

Приложение должно быть разработано в виде нескольких модулей, взаимодействующих между собой с использованием дополнительных информационных файлов.

2.2.2 Требования к составу функций приложения

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

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

2.2.2 Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Изображения шашек могут храниться в

отдельных графических файлах. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде. Пояснительные информационные сообщения для пользователя должны выводиться внизу игрового поля по ходу игры.

2.3 Требования к надёжности

Приложение должно чётко выполнять команды игрока и в случае ошибки сообщить игроку о ней и попытается исправить.

2.4 Требования к информационной и программной совместимости

- ОС Windows
- Версия Python: 3.11
- Среда разработки: PyCharm Community Edition

2.5 Требования к маркировке и упаковке

Определяются заданием на курсовую работу.

2.6 Требования к транспортированию и хранению

2.6.1 Условия транспортирования

Требования к условиям транспортирования не предъявляются.

2.6 2 Условия хранения

Требования к условиям хранения не предъявляются

2.6 3 Сроки хранения

Срок хранения – до июля 2024 года.

3. Требования к программной документации

Определяются заданием на курсовую работу.

4. Стадии и этапы разработки

Определяются заданием на курсовую работу.

5. Порядок контроля и приёмки

Определяются заданием на курсовую работу.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема Компьютерная логическая игра Турецкие шашки-поддавки

Пояснительная записка

Р.02069337.22/2411-12 ПЗ-01

Листов 7

Полп. и

Инв

Взз

Полп. и

Инв.

Исполнитель:
студент гр. ИСТбд-21
Прокофьев Сергей Александрович
«____» _____ 2024 г.

2024

Введение

Название разрабатываемого приложения — "Турецкие шашки-поддавки". Это приложение представляет собой электронную версию игры "Турецкие шашки", доступную на персональных компьютерах.

При разработке "Турецкие шашки-поддавки" был выбран подход, основанный на математической модели игры "Турецкие шашки". Математическая модель позволяет точно определить логику ходов, правила игры и алгоритмы принятия решений компьютерного противника. Такой подход обеспечивает точность и надежность реализации игры, а также позволяет создать интеллектуального компьютерного противника, способного представить вызов даже опытным игрокам.

Приложение обладает удобным и интуитивно понятным интерфейсом, позволяющим игрокам легко размещать свои фишки на игровом поле и применять стратегические решения.

1. Проектная часть

1.1 Постановка задачи на разработку приложения

Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании (приложение 1).

1.2 Математические методы

Модель игрового поля компьютерной логической игры
«Турецкие шашки-поддавки»

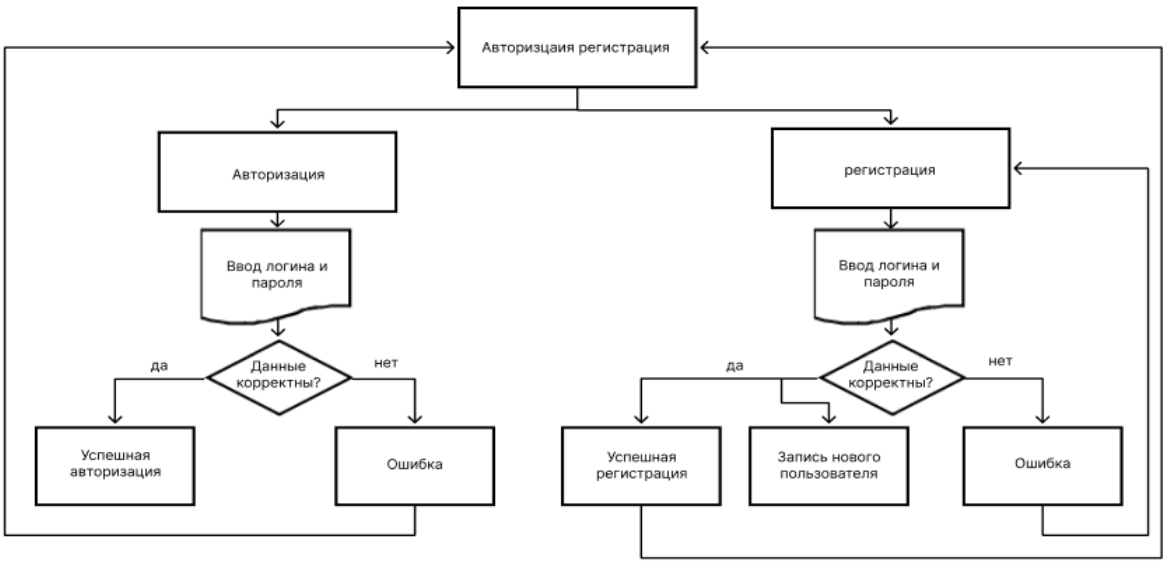
Матрица расположения шашек **8**x**8**

	a	b	c	d	e	f	g	h
8								
7	2	2	2	2	2	2	2	2
6	2	2	2	2	2	2	2	2
5								
4								
3	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
1								

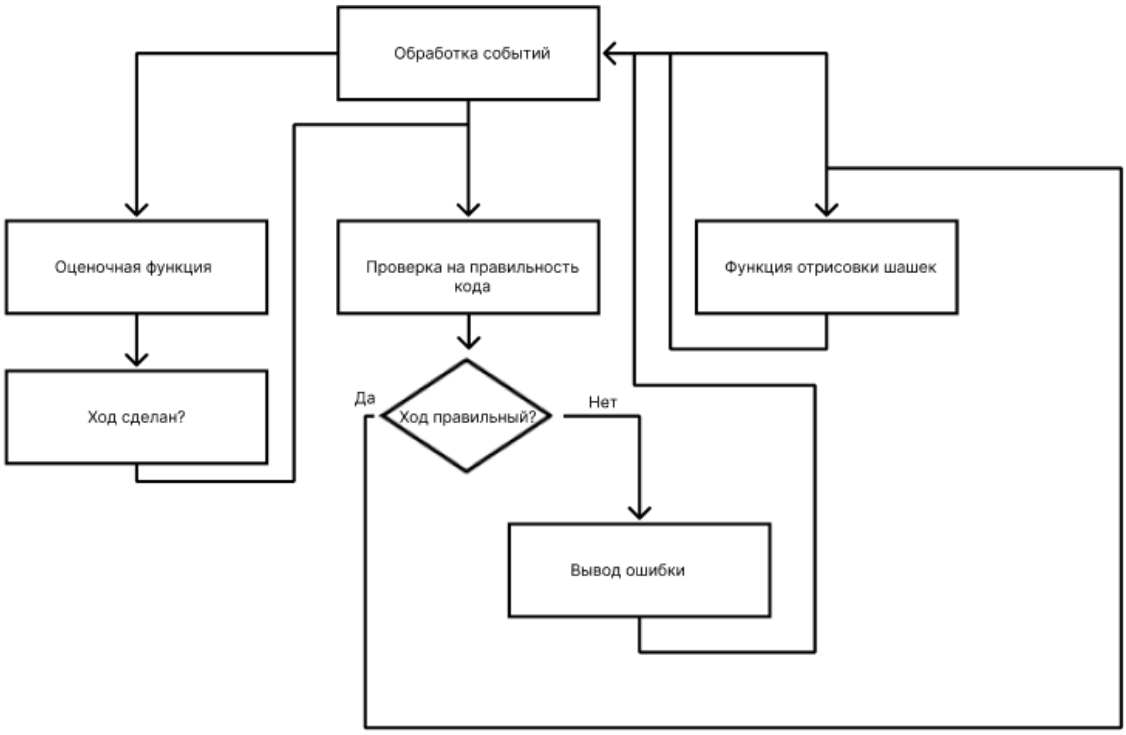
1.3 Архитектура и алгоритмы

1.3.1. Архитектура

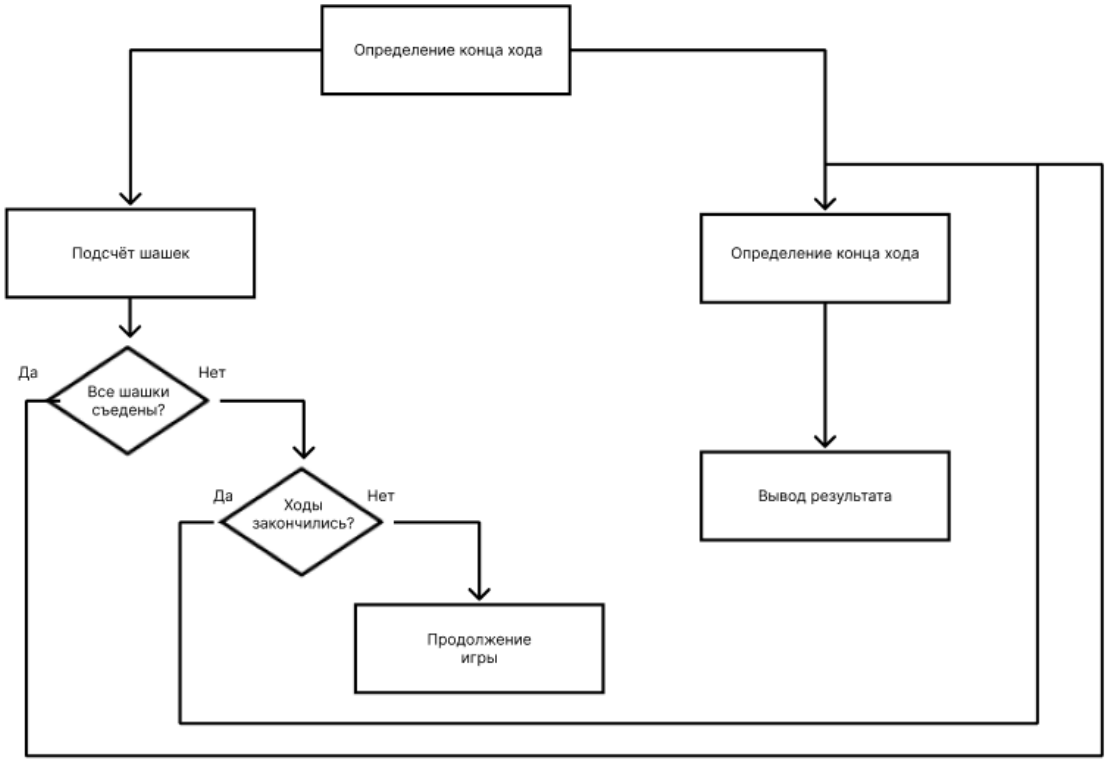
1.3.2. Алгоритм авторизации/регистрации



1.3.3 Алгоритм обработки событий



1.3.4 Алгоритм определение конца хода



1.4 Тестирование

1.4.1 Описание отчета о тестировании

Отчет о тестировании игры "Турецкие шашки-поддавки" предоставляет обзор текущего состояния игры и качества ее реализации. Он включает информацию о проведенных функциональных тестах, тестировании пользовательского интерфейса, проверке граничных значений, совместимости, производительности и безопасности. Каждый из этих аспектов тестирования подробно описывается в отчете.

Основная цель отчета о тестировании игры "Турецкие шашки-поддавки" - предоставить полную картину о ее качестве и выявленных проблемах. Отчет содержит рекомендации по исправлению проблем и улучшению функциональности игры.

1.4.2 Цель тестирования

Проверить функциональность, корректность ввода/вывода и общую стабильность приложения "Турецкие шашки поддавки".

1.4.3 Методика тестирования

Модульное тестирование: в этом методе можно проводится тестирование отдельных модулей игры, таких как логика ходов, правила игры, алгоритмы принятия решений компьютерного противника и другие важные компоненты. Это позволит убедиться в правильной работе каждого модуля и их соответствии требованиям.

Интеграционное тестирование: здесь проверяется взаимодействие различных модулей игры между собой. Например, как правильно обновляется игровое поле при ходе игрока или компьютера, как корректно обновляется счет игры и т.д. Целью является выявление возможных ошибок во взаимодействии компонентов игры.

Функциональное тестирование: этот метод позволяет проверить функциональность игры в соответствии с требованиями. Проверить правильность подсчета очков, корректность применения правил игры и другие функциональные аспекты.

Графическое тестирование: этот метод предполагает проверку корректности отображения графического интерфейса игры. Это включает проверку расположения элементов на экране, соответствие цветов и шрифтов, и других визуальных аспектов игры.

1.4.4 Проведенные тесты

Тестирование функциональности:

- Цель: проверить основную функциональность игры "Турецкие шашки-поддавки".
- Шаги выполнения:

1. Открыть приложение "Турецкие шашки-поддавки".
2. Зарегистрировать нового пользователя.
3. Начать игру.
4. Выполнить несколько ходов.
5. Завершить игру.

Ожидаемые результаты:

- Пользователь успешно зарегистрирован.
- Игра успешно запускается.
- Игровое поле отображается корректно.
- Игрок может совершать ходы в соответствии с правилами игры.
- Результаты игры отображаются правильно.
- Фактические результаты: Все ожидаемые результаты подтверждены.

Тестирование корректности ввода/вывода:

Цель: проверить корректность обработки пользовательского ввода и вывода информации.

- Шаги выполнения:

1. Оставить окно регистрации пустым.
2. Выбрать клетку во время хода, не соответствующую правилам игры.

Ожидаемые результаты:

- Если оставить поля в окне регистрации пустыми, то выводится сообщение об ошибке.
- При выборе клетки во время хода, не соответствующей правилам игры, ход не производится.

Фактические результаты: все ожидаемые результаты подтверждены.

1.3.5 Выводы

Приложение "Турецкие шашки-поддавки" демонстрирует стабильную работу и правильную функциональность в большинстве случаев.

2. Источники, использованные при разработке

1. Правила игры в шашки: классические, русские, английские [Электронный ресурс] URL: https://minigames.mail.ru/info/article/shashki_pravila?ysclid=lrrxps0qgp880540204

2. Комбинация в турецких шашках [Электронный ресурс] URL: http://wmsg.ru/2015/07/28/turkish-draughts-beauty/

3. Шишкин, В.В. Разработка логических компьютерных игр с графическим интерфейсом в среде питон / В.В. Шишкин, Д.С. Афонин. - Ульяновск: УлГТУ, 2023. – 89 с. – URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://lib.ulstu.ru/venec/disk/2023/112.pdf. – Текст: электронный

4. Руководство по Tkinter . – URL: https://metanit.com/python/tkinter/

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема Компьютерная логическая игра Турецкие шашки-поддавки

Руководство программиста

Р.02069337.22/2411-12 РП-01

Листов 24

Исполнитель:
студент гр. ИСТбд-21
Прокофьев Сергей Александрович
«____» _____ 2024 г.

2024

16

1. Назначение и условия применения программы

1.1 Назначение и функции, выполняемые приложением

Функциональное назначение: разработка приложения для игры "Турецкие шашки-поддавки" с целью предоставления пользователю возможности играть в эту стратегическую настольную игру на своем устройстве.

Свод правил игры "Турецкие шашки-поддавки":

- Игра проводится на игровом поле размером 8x8 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.
- Простая шашка ходит на одно поле вперёд, влево, вправо.
- Игроки ходят по очереди, ставя фишки на свободные клетки поля.
- 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. Побив шашку противника дамка обязана встать на поле за битой шашкой.
- 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°
- 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.
- Выигрывает тот, кто смог «отдать» все шашки противнику.

Общая характеристика функциональных возможностей приложения:

1. Игра с другими игроками: Приложение позволяет играть в многопользовательском режиме, где пользователи могут играть против друг друга на одном устройстве.

1.2 Условия, необходимые для использования приложения

Требования к операционной системе: Windows

Требования к платформе: настольный компьютер

2. Характеристики программы

2.1 Характеристики приложения

Количество значимых строк программного кода: 896.

Библиотеки, используемые при написании кода:

os – для записи данных о зарегистрированных пользователях;

tkinter – для отображения окон, игрового поля и других графических интерфейсов в программе.

После запуска приложения у пользователя появляется окно авторизации/регистрации (Рис.1).

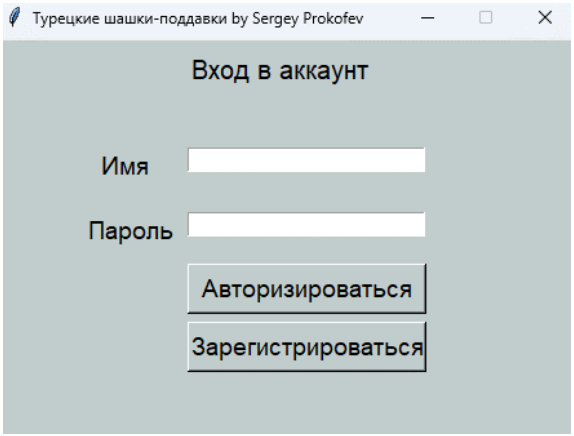


Рис. 1

В появившемся окне необходим ввести своё имя пользователя. Если пользователь не зарегистрирован, ему необходимо придумать своё имя. Во втором поле необходимо ввести пароль, указанный при регистрации. Если аккаунт пользователя ещё не создан, то необходимо придумать пароль.

Если никаких данных введено не было, возникнет сообщение об ошибке (Рис.2):

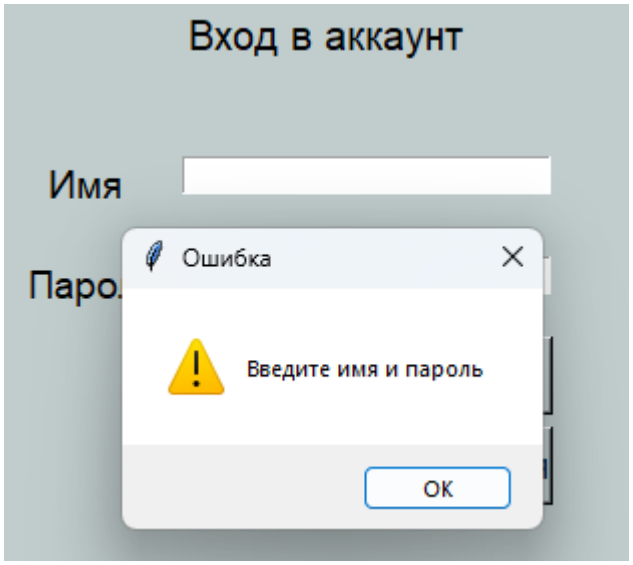


Рис. 2

При вводе неправильного логина или пароля возникнет сообщение об ошибке (Рис. 3):

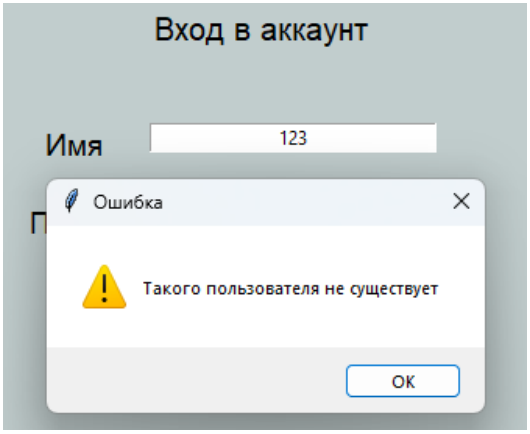


Рис. 3

При вводе имеющихся данных выведет сообщение (Рис.4):

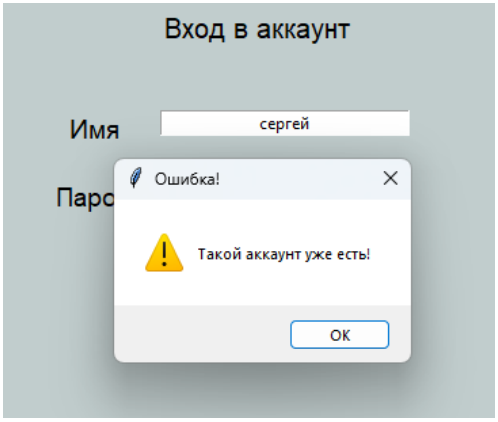


Рис. 4

После успешной авторизации/регистрации появляется окно (рис.5), где предлагается начать игру.

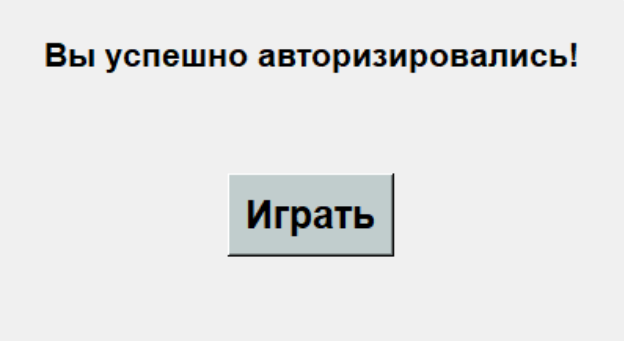


Рис. 5

После нажатия кнопки «Играть» появляется окно (рис. 6): на котором расположено игровое поле размеров 8x8 клеток. В самом низу окна имеется отображение хода белых/черных.

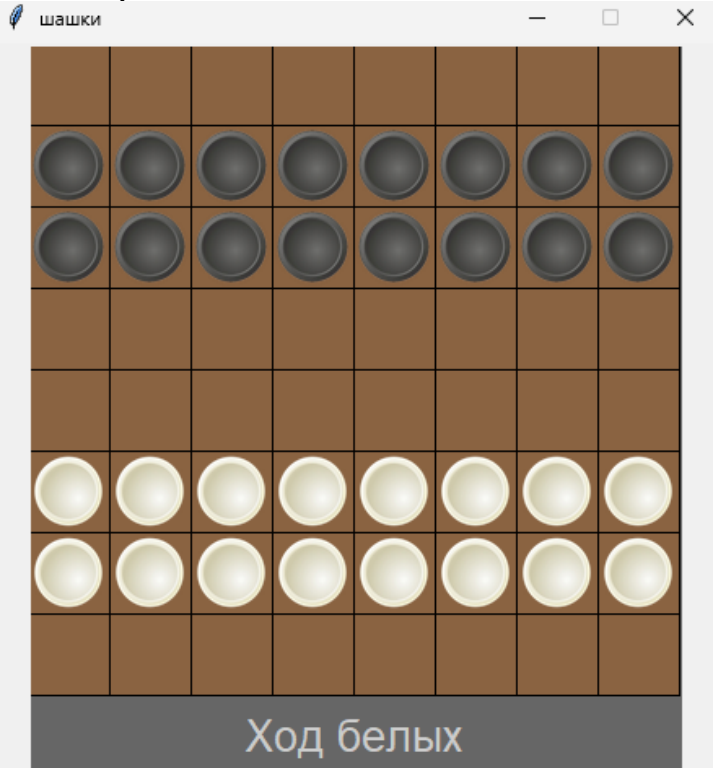


Рис. 6

Чтобы сделать ход, необходимо нажать на соответствующую клетку игрового поля. После хода пользователя, ход делает другой пользователь.

Игра считается завершённой, когда у пользователя не остается ни одной из шашек. (Рис. 7)

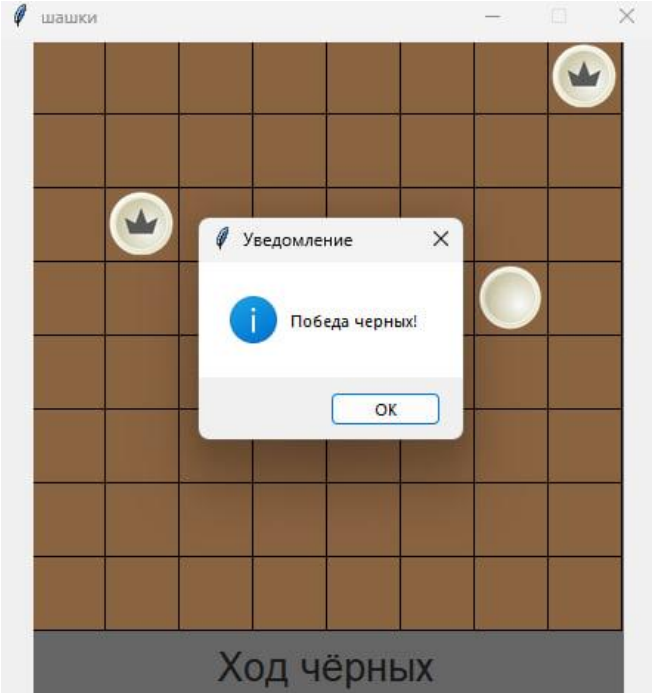


Рис. 7

2.2 Особенности реализации приложения

Структура данных.

Игровое поле: для представления игрового поля размером 8x8 клеток, используется двумерный массив или матрицу. Каждая клетка представлена значением, указывающим на цвет фишки (0 – пустая клетка, 1 – белая шашка, 2 – чёрная шашка). Использование массива обеспечивает эффективный доступ к элементам и удобство при выполнении операций на игровом поле.

Массив поля:

```
board = [[0, 0, 0, 0, 0, 0, 0, 0],
         [2, 2, 2, 2, 2, 2, 2, 2], черные
         [2, 2, 2, 2, 2, 2, 2, 2], фигуры
         [0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0],
         [1, 1, 1, 1, 1, 1, 1, 1], белые
         [1, 1, 1, 1, 1, 1, 1, 1], фигуры
         [0, 0, 0, 0, 0, 0, 0, 0]]
```

3. Обращение к программе

Библиотека os:

Библиотека используется для записи данных при регистрации пользователя, а также проверки данных при авторизации пользователя.

Библиотека tkinter:

Tkinter – это кросс-платформенная событийно-ориентированная графическая библиотека Python, предназначенная для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя (GUI).

Модуль tkinter.messagebox, предоставляющий доступ к готовым диалоговым окнам.

4. Сообщения

«Ошибка», «Введите имя и пароль» при проверке на пустое поле;

«Информация», «Такой аккаунт уже есть»;

«Информация», «Такого пользователя не существует»;

«Информация», «Регистрация успешно пройдена» при успешной регистрации;

«Информация», «Авторизация успешно пройдена» при успешной авторизации;

«Информация», «Игра окончена: чёрные (белые) победили!».

Текст программы

```
# Импорт необходимых модулей
from tkinter import *
import os
from tkinter import messagebox
from PIL import Image
from PIL import ImageTk


# Класс, представляющий шашечную доску
class CheckersBoard:


  def reset_board(self):
    # Сброс игровой доски и отображение начальной позиции фигур
    self.board = [
      [0, 0, 0, 0, 0, 0, 0, 0],
      [2, 2, 2, 2, 2, 2, 2, 2], # Второй игрок
      [2, 2, 2, 2, 2, 2, 2, 2], # Черные фигуры
      [0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0],
      [1, 1, 1, 1, 1, 1, 1, 1], # Первый игрок
      [1, 1, 1, 1, 1, 1, 1, 1], # Белые фигуры
      [0, 0, 0, 0, 0, 0, 0, 0]
    ]
    self.canvas.delete("all") # Удаление всех элементов на холсте
```

```
    self.draw_board(self.current_player) # Отрисовка начального состояния доски

def check_col(self, start, end, col):
    # Проверка на возможность хода по столбцу
    count = 0  # Счетчик шашек в столбце
    index = 0  # Индекс возможного хода
    correct_ind = end
    if start > end:
        start, end = end - 1, start - 1
    else:
        correct_ind -= 2
    for i in range(start + 1, end + 1):
        if self.board[i][col] != 0:  # Если в клетке есть шашка
            if self.board[i][col] in self.current_player:  # Если шашка принадлежит текущему игроку
                count += 1
            count += 1
            index = i + 1
    if count == 0:
        return [True, False]
    elif count == 1 and correct_ind + 1 == index - 1:
        return [True, index]
    else:
        return [False, False]

def check_row(self, start, end, row):
    # Проверка на возможность хода по строке
    count = 0
    index = 0
    correct_ind = end - 1
    if start > end:
        correct_ind += 2
        start, end = end, start
    for i in range(start + 1, end):
        if self.board[row][i] != 0:  # Если в клетке есть шашка
            if self.board[row][i] in self.current_player:
                count += 1  # Увеличиваем счетчик шашек
            index = i + 1  # Устанавливаем индекс возможного хода
            count += 1  # Увеличиваем счетчик шашек
    if count == 0:
        return [True, False]  # Возможный ход без взятия
    elif count == 1 and correct_ind == index - 1:
        return [True, index]  # Возможный ход с взятием
    else:
        return [False, False]  # Ход невозможен

def attack(self, take, row, col):
```

```
# Если нет взятий
if not take:
    # Если текущий игрок - 1 или 3
    if self.current_player == [1, 3]:
        # Перебираем клетки доски
        for i in range(8):
            for j in range(8):
                try:
                    # Если фишка на доске является дамкой или обычной фишкой игрока
                    if self.board[i][j] in [2, 4]:
                        # Проверяем возможные ходы вверх и вниз
                        if self.board[i - 1][j] == 0:
                            for k in range(i + 1, 8):
                                if self.board[k][j] == 3:
                                    self.need_attack = True
                                    break
                                elif self.board[k][j] in [1, 2, 3, 4]:
                                    break
                        if self.board[i + 1][j] == 0:
                            for k in range(i - 1, -1, -1):
                                if self.board[k][j] == 3:
                                    self.need_attack = True
                                    break
                                elif self.board[k][j] == 1 and abs(i - k) == 1:
                                    self.need_attack = True
                                    break
                                elif self.board[k][j] in [1, 2, 3, 4]:
                                    break
                except IndexError:
                    pass
        # Перебираем клетки доски
        for i in range(8):
            for j in range(8):
                try:
                    # Если фишка на доске является дамкой или обычной фишкой игрока
                    if self.board[i][j] in [2, 4]:
                        # Проверяем возможные ходы влево и вправо
                        if self.board[i][j - 1] == 0:
                            for k in range(j + 1, 8):
                                if self.board[i][k] == 3:
                                    self.need_attack = True
                                    break
                                elif self.board[i][k] == 1 and abs(j - k) == 1:
                                    self.need_attack = True
                                    break
                                elif self.board[i][k] in [1, 2, 3, 4]:
```

```
                        break
                if self.board[i][j + 1] == 0:
                    for k in range(j - 1, -1, -1):
                        if self.board[i][k] == 3:
                            self.need_attack = True
                            break
                        elif self.board[i][k] == 1 and abs(j - k) == 1:
                            self.need_attack = True
                            break
                        elif self.board[i][k] in [1, 2, 3, 4]:
                            break
            except IndexError:
                pass
    # Если текущий игрок - 2 или 4
    if self.current_player == [2, 4]:
        # Перебираем клетки доски
        for i in range(8):
            for j in range(8):
                try:
                    # Если фишка на доске является дамкой или обычной фишкой игрока
                    if self.board[i][j] in [1, 3]:
                        # Проверяем возможные ходы вверх и вниз
                        if self.board[i - 1][j] == 0:
                            for k in range(i + 1, 8):
                                if self.board[k][j] == 4:
                                    self.need_attack = True
                                    break
                                elif self.board[k][j] == 2 and abs(i - k) == 1:
                                    self.need_attack = True
                                    break
                                elif self.board[k][j] in [1, 3]:
                                    break
                        if self.board[i + 1][j] == 0:
                            for k in range(i - 1, -1, -1):
                                if self.board[k][j] == 4:
                                    self.need_attack = True
                                    break
                                elif self.board[k][j] in [1, 2, 3, 4]:
                                    break
                except IndexError:
                    pass
        # Перебираем клетки доски
        for i in range(8):
            for j in range(8):
                try:
                    # Если фишка на доске является дамкой или обычной фишкой игрока
```

```
                    if self.board[i][j] in [1, 3]:
                        # Проверяем возможные ходы влево и вправо
                        if self.board[i][j - 1] == 0:
                            for k in range(j + 1, 8):
                                if self.board[i][k] == 4:
                                    self.need_attack = True
                                    break
                                elif self.board[i][k] == 2 and abs(j - k) == 1:
                                    self.need_attack = True
                                    break
                                elif self.board[i][k] in [1, 3]:
                                    break
                        if self.board[i][j + 1] == 0:
                            for k in range(j - 1, -1, -1):
                                if self.board[i][k] == 4:
                                    self.need_attack = True
                                    break
                                elif self.board[i][k] == 2 and abs(j - k) == 1:
                                    self.need_attack = True
                                    break
                                elif self.board[i][k] in [1, 2, 3, 4]:
                                    break
                except IndexError:
                    pass


        def check_win(self):
            # Проверка на завершение игры и определение победителя или ничьей
            count = 0
            count2 = 0
            for i in range(8):
                count += self.board[i].count(1)
                count += self.board[i].count(3)
                count2 += self.board[i].count(2)
                count2 += self.board[i].count(4)
            if count > 0 >= count2: # Проверяет, выиграл ли игрок 1 (черные), если количество его фишек больше нуля, а у игрока 2 (белых) равно нулю
                messagebox.showinfo("Уведомление", "Победа черных!")
                self.reset_board()
            elif count2 > 0 >= count: # Проверяет, выиграл ли игрок 2 (белые), если количество его фишек больше нуля, а у игрока 1 (черных) равно нулю
                messagebox.showinfo("Уведомление", "Победа белых!")
                self.reset_board()
            elif count == count2 == 1: # Проверяет, наступила ли ничья, если количество фишек у обоих игроков равно 1
                messagebox.showinfo("Уведомление", "Игра закончилась ничьей!")
```

```
            self.reset_board()
        else:
            count = 0 # Сбрасывает переменную count для подсчета возможных ходов у игрока 1 (черных)
            count2 = 0 # Сбрасывает переменную count2 для подсчета возможных ходов у игрока 2 (белых)
            for i in range(8):
                for j in range(8):
                    if self.board[i][j] in [1, 3]:
                        try:
                            if self.board[i + 1][j] == 0 or (
                                    self.board[i + 1][j] in [2, 4] and self.board[i + 2][j] == 0):
                                count += 1
                            elif self.board[i][j + 1] == 0 or (
                                    self.board[i][j + 1] in [2, 4] and self.board[i][j + 2] == 0):
                                count += 1
                            elif self.board[i][j - 1] == 0 or (
                                    self.board[i][j - 1] in [2, 4] and self.board[i][j - 2] == 0):
                                count += 1
                        except IndexError:
                            pass
                    elif self.board[i][j] in [2, 4]:
                        try:
                            if self.board[i - 1][j] == 0 or (
                                    self.board[i - 1][j] in [1, 3] and self.board[i - 2][j] == 0):
                                count2 += 1
                            elif self.board[i][j + 1] == 0 or (
                                    self.board[i][j + 1] in [1, 3] and self.board[i][j + 2] == 0):
                                count2 += 1
                            elif self.board[i][j - 1] == 0 or (
                                    self.board[i][j - 1] in [1, 3] and self.board[i][j - 2] == 0):
                                count2 += 1
                        except IndexError:
                            pass
            print(count, count2)
            if count == 0:
                messagebox.showinfo("Уведомление", "Победа белых!")
                self.reset_board()
            if count2 == 0:
                messagebox.showinfo("Уведомление", "Победа черных!")
                self.reset_board()


    def switch_player(self):
        # Переключение текущего игрока между [1, 3] и [2, 4]
        if self.current_player == [1, 3]:
            self.current_player = [2, 4]
        else:
```

```
        self.current_player = [1, 3]
    def v_moves(self, src_row, src_col, dest_row, dest_col):
        # Если исходная клетка содержит обычную фишку и клетка назначения пуста
        if self.board[src_row][src_col] == 1 and self.board[dest_row][dest_col] == 0:
            # Проверяем возможные ходы для обычной фишки
            if (dest_row - src_row == -1 and abs(dest_col - src_col) == 0) or \
                    (dest_row - src_row == 0 and abs(dest_col - src_col) == 1):
                # Если нет необходимости в атаке
                if not self.need_attack:
                    return [True, False]
            elif (dest_row - src_row == -2 and abs(dest_col - src_col) == 0 and
                  self.board[dest_row + 1][dest_col] in [2, 4]):
                # Если возможно взятие по вертикали
                self.board[dest_row + 1][dest_col] = 0
                if self.need_attack:
                    self.need_attack = False
                try:
                    if self.board[dest_row - 1][dest_col] in [2, 4] and self.board[dest_row - 2][dest_col] == 0:
                        return [True, True]
                    elif self.board[dest_row][dest_col - 1] in [2, 4] and self.board[dest_row][dest_col - 2] == 0:
                        return [True, True]
                    elif self.board[dest_row][dest_col + 1] in [2, 4] and self.board[dest_row][dest_col + 2] == 0:
                        return [True, True]
                    else:
                        return [True, False]
                except IndexError:
                    return [True, False]
            elif dest_col - src_col == 2 and dest_row == src_row and self.board[dest_row][
                dest_col - 1] in [2, 4]:
                # Если возможно взятие по горизонтали
                self.board[dest_row][dest_col - 1] = 0
                if self.need_attack:
                    self.need_attack = False
                try:
                    if self.board[dest_row + 1][dest_col] in [2, 4] and self.board[dest_row + 2][dest_col] == 0:
                        return [True, True]
                    elif self.board[dest_row][dest_col + 1] in [2, 4] and self.board[dest_row][dest_col + 2] == 0:
                        return [True, True]
                    else:
                        return [True, False]
                except IndexError:
                    return [True, False]
            elif dest_col - src_col == -2 and dest_row == src_row and self.board[dest_row][
                dest_col + 1] in [2, 4]:
                # Если возможно взятие по горизонтали
                self.board[dest_row][dest_col + 1] = 0
```

```
            if self.need_attack:
                self.need_attack = False
            try:
                if self.board[dest_row + 1][dest_col] in [2, 4] and self.board[dest_row + 2][dest_col] == 0:
                    return [True, True]
                elif self.board[dest_row][dest_col - 1] in [2, 4] and self.board[dest_row][dest_col - 2] == 0:
                    return [True, True]
                else:
                    return [True, False]
            except IndexError:
                return [True, False]
        # Если исходная клетка содержит дамку и клетка назначения пуста
        elif self.board[src_row][src_col] == 3 and self.board[dest_row][dest_col] == 0:
            # Проверяем возможные ходы для дамки
            if (abs(dest_row - src_row) >= 1 and abs(dest_col - src_col) == 0) or \
                    (abs(dest_row - src_row) == 0 and abs(dest_col - src_col) >= 1):
                if src_col == dest_col:
                    # Проверяем возможные ходы по вертикали
                    correct, attack = self.check_col(src_row, dest_row, src_col)
                    if correct and attack == 0 and not self.need_attack:
                        return [True, False]
                    elif correct and attack >= 1:
                        # Выполняем взятие
                        self.board[attack - 1][src_col] = 0
                        # Проверяем возможные ходы после взятия
                        if dest_row - (attack - 1) > 0:
                            try:
                                for i in range(dest_row + 1, 8):
                                    if self.board[i][dest_col] in [2, 4] and self.board[i + 1][dest_col] == 0:
                                        return [True, True]
                                    elif self.board[i][dest_col] in [1, 2, 3, 4]:
                                        break
                            except IndexError:
                                pass
                            try:
                                for i in range(dest_col + 1, 8):
                                    if self.board[dest_row][i] in [2, 4] and self.board[dest_row][i + 1] == 0:
                                        return [True, True]
                                    elif self.board[dest_row][i] in [1, 2, 3, 4]:
                                        break
                            except IndexError:
                                pass
                            try:
                                for i in range(dest_col - 1, 0, -1):
                                    if self.board[dest_row][i] in [2, 4] and self.board[dest_row][i - 1] == 0:
                                        return [True, True]
```

```
                elif self.board[dest_row][i] in [1, 2, 3, 4]:
                    break
            except IndexError:
                pass
        elif dest_row - (attack - 1) < 0:
            try:
                for i in range(dest_row - 1, 0, -1):
                    if self.board[i][dest_col] in [2, 4] and self.board[i - 1][dest_col] == 0:
                        return [True, True]
                    elif self.board[i][dest_col] in [1, 2, 3, 4]:
                        break
            except IndexError:
                pass
            try:
                for i in range(dest_col + 1, 8):
                    if self.board[dest_row][i] in [2, 4] and self.board[dest_row][i + 1] == 0:
                        return [True, True]
                    elif self.board[dest_row][i] in [1, 2, 3, 4]:
                        break
            except IndexError:
                pass
            try:
                for i in range(dest_col - 1, 0, -1):
                    if self.board[dest_row][i] in [2, 4] and self.board[dest_row][i - 1] == 0:
                        return [True, True]
                    elif self.board[dest_row][i] in [1, 2, 3, 4]:
                        break
            except IndexError:
                pass
        return [True, False]
    else:
        if not take:
            return [False, False]
elif src_row == dest_row:
    # Проверяем возможные ходы по горизонтали
    correct, attack = self.check_row(src_col, dest_col, src_row)
    if correct and attack == 0 and not self.need_attack:
        return [True, False]
    elif correct and attack >= 1:
        # Выполняем взятие
        self.board[src_row][attack - 1] = 0
        # Проверяем возможные ходы после взятия
        if dest_col - (attack - 1) > 0:
            try:
                for i in range(dest_col + 1, 8):
                    if self.board[dest_row][i] in [2, 4] and self.board[dest_row][i + 1] == 0:
```

```
                return [True, True]
            elif self.board[dest_row][i] in [1, 2, 3, 4]:
                break
    except IndexError:
        pass
    try:
        for i in range(dest_row + 1, 8):
            if self.board[i][dest_col] in [2, 4] and self.board[i + 1][dest_col] == 0:
                return [True, True]
            elif self.board[i][dest_col] in [1, 2, 3, 4]:
                break
    except IndexError:
        pass
    try:
        for i in range(dest_row - 1, 0, -1):
            if self.board[i][dest_col] in [2, 4] and self.board[i - 1][dest_col] == 0:
                return [True, True]
            elif self.board[i][dest_col] in [1, 2, 3, 4]:
                break
    except IndexError:
        pass
elif dest_col - (attack - 1) < 0:
    try:
        for i in range(dest_col - 1, 0, -1):
            if self.board[dest_row][i] in [2, 4] and self.board[dest_row][i - 1] == 0:
                return [True, True]
            elif self.board[dest_row][i] in [1, 2, 3, 4]:
                break
    except IndexError:
        pass
    try:
        for i in range(dest_row + 1, 8):
            if self.board[i][dest_col] in [2, 4] and self.board[i + 1][dest_col] == 0:
                return [True, True]
            elif self.board[i][dest_col] in [1, 2, 3, 4]:
                break
    except IndexError:
        pass
    try:
        for i in range(dest_row - 1, 0, -1):
            if self.board[i][dest_col] in [2, 4] and self.board[i - 1][dest_col] == 0:
                return [True, True]
            elif self.board[i][dest_col] in [1, 2, 3, 4]:
                break
    except IndexError:
        pass
```

```
                return [True, False]
            else:
                if not take:
                    return [False, False]
    # Если исходная клетка содержит фишку-дамку и клетка назначения пуста
    elif self.board[src_row][src_col] == 2 and self.board[dest_row][dest_col] == 0:
        # Проверяем возможные ходы для фишки-дамки
        if (dest_row - src_row == 1 and abs(dest_col - src_col) == 0) or \
                (dest_row - src_row == 0 and abs(dest_col - src_col) == 1):
            # Если нет необходимости в атаке
            if not self.need_attack:
                return [True, False]
        elif (dest_row - src_row == 2 and abs(dest_col - src_col) == 0 and
              self.board[dest_row - 1][dest_col] in [1, 3]):
            # Если возможно взятие по вертикали
            self.board[dest_row - 1][dest_col] = 0
            if self.need_attack:
                self.need_attack = False
            try:
                if self.board[dest_row + 1][dest_col] in [1, 3] and self.board[dest_row + 2][dest_col] == 0:
                    return [True, True]
                elif self.board[dest_row][dest_col - 1] in [1, 3] and self.board[dest_row][dest_col - 2] == 0:
                    return [True, True]
                elif self.board[dest_row][dest_col + 1] in [1, 3] and self.board[dest_row][dest_col + 2] == 0:
                    return [True, True]
                else:
                    return [True, False]
            except IndexError:
                return [True, False]
        elif dest_col - src_col == 2 and dest_row == src_row and self.board[dest_row][
            dest_col - 1] in [1, 3]:
            # Если возможно взятие по горизонтали
            self.board[dest_row][dest_col - 1] = 0
            if self.need_attack:
                self.need_attack = False
            try:
                if self.board[dest_row - 1][dest_col] in [1, 3] and self.board[dest_row - 2][dest_col] == 0:
                    return [True, True]
                elif self.board[dest_row][dest_col + 1] in [1, 3] and self.board[dest_row][dest_col + 2] == 0:
                    return [True, True]
                else:
                    return [True, False]
            except IndexError:
                return [True, False]
        elif dest_col - src_col == -2 and dest_row == src_row and self.board[dest_row][
            dest_col + 1] in [1, 3]:
```

```
                    # Если возможно взятие по горизонтали
                    self.board[dest_row][dest_col + 1] = 0
                    if self.need_attack:
                        self.need_attack = False
                    try:
                        if self.board[dest_row - 1][dest_col] in [1, 3] and self.board[dest_row - 2][dest_col] == 0:
                            return [True, True]
                        elif self.board[dest_row][dest_col - 1] in [1, 3] and self.board[dest_row][dest_col - 2] == 0:
                            return [True, True]
                        else:
                            return [True, False]
                    except IndexError:
                        return [True, False]
                elif self.board[src_row][src_col] == 4 and self.board[dest_row][dest_col] == 0: # Если исходная клетка содержит дамку и клетка назначения пуста
                    if (abs(dest_row - src_row) >= 1 and abs(dest_col - src_col) == 0) or \
                        (abs(dest_row - src_row == 0) and abs(dest_col - src_col) >= 1):
                        if src_col == dest_col:
                            correct, attack = self.check_col(src_row, dest_row, src_col)
                            if correct and attack == 0 and not self.need_attack:
                                return [True, False]
                            elif correct and attack >= 1:
                                self.board[attack - 1][src_col] = 0
                                if dest_row - (attack - 1) > 0:
                                    try:
                                        for i in range(dest_row + 1, 8):
                                            if self.board[i][dest_col] in [1, 3] and self.board[i + 1][dest_col] == 0:
                                                return [True, True]
                                            elif self.board[i][dest_col] in [1, 2, 3, 4]:
                                                break
                                    except IndexError:
                                        pass
                                    try:
                                        for i in range(dest_col + 1, 8):
                                            if self.board[dest_row][i] in [1, 3] and self.board[dest_row][i + 1] == 0:
                                                return [True, True]
                                            elif self.board[dest_row][i] in [1, 2, 3, 4]:
                                                break
                                    except IndexError:
                                        pass
                                    try:
                                        for i in range(dest_col - 1, 0, -1):
                                            if self.board[dest_row][i] in [1, 3] and self.board[dest_row][i - 1] == 0:
                                                return [True, True]
                                            elif self.board[dest_row][i] in [1, 2, 3, 4]:
                                                break
```

```
                except IndexError:
                    pass
            elif dest_row - (attack - 1) < 0:
                try:
                    for i in range(dest_row - 1, 0, -1):
                        if self.board[i][dest_col] in [1, 3] and self.board[i - 1][dest_col] == 0:
                            return [True, True]
                        elif self.board[i][dest_col] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                try:
                    for i in range(dest_col + 1, 8):
                        if self.board[dest_row][i] in [1, 3] and self.board[dest_row][i + 1] == 0:
                            return [True, True]
                        elif self.board[dest_row][i] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                try:
                    for i in range(dest_col - 1, 0, -1):
                        if self.board[dest_row][i] in [1, 3] and self.board[dest_row][i - 1] == 0:
                            return [True, True]
                        elif self.board[dest_row][i] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                return [True, False]
            else:
                if not take:
                    return [False, False]
        elif src_row == dest_row:
            correct, attack = self.check_row(src_col, dest_col, src_row)
            if correct and attack == 0 and not self.need_attack:
                return [True, False]
            elif correct and attack >= 1:
                self.board[src_row][attack - 1] = 0
                if dest_col - (attack - 1) > 0:
                    try:
                        for i in range(dest_col + 1, 8):
                            if self.board[dest_row][i] in [1, 3] and self.board[dest_row][i + 1] == 0:
                                return [True, True]
                            elif self.board[dest_row][i] in [1, 2, 3, 4]:
                                break
                    except IndexError:
                        pass
```

```
                try:
                    for i in range(dest_row + 1, 8):
                        if self.board[i][dest_col] in [1, 3] and self.board[i + 1][dest_col] == 0:
                            return [True, True]
                        elif self.board[i][dest_col] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                try:
                    for i in range(dest_row - 1, 0, -1):
                        if self.board[i][dest_col] in [1, 3] and self.board[i - 1][dest_col] == 0:
                            return [True, True]
                        elif self.board[i][dest_col] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
            elif dest_col - (attack - 1) < 0:
                try:
                    for i in range(dest_col - 1, 0, -1):
                        if self.board[dest_row][i] in [1, 3] and self.board[dest_row][i - 1] == 0:
                            return [True, True]
                        elif self.board[dest_row][i] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                try:
                    for i in range(dest_row + 1, 8):
                        if self.board[i][dest_col] in [1, 3] and self.board[i + 1][dest_col] == 0:
                            return [True, True]
                        elif self.board[i][dest_col] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                try:
                    for i in range(dest_row - 1, 0, -1):
                        if self.board[i][dest_col] in [1, 3] and self.board[i - 1][dest_col] == 0:
                            return [True, True]
                        elif self.board[i][dest_col] in [1, 2, 3, 4]:
                            break
                except IndexError:
                    pass
                return [True, False]
            else:
                if not take:
                    return [False, False]
    return [False, False]
```

```
    # Инициализация главного окна
def __init__(self, master):
    self.master = master
    self.master.title("Турецкие шашки-поддавки by Sergey Prokofev")
    self.master.resizable(False, False)
    self.center_window()
    self.canvas = Canvas(self.master, width=540, height=450, bg="azure3")
    self.canvas.pack(side=RIGHT)
    self.users = {}
    self.canvas.pack()
    self.users = {}
    # Загрузка изображений шашек и дамок
    global black_queen, white_queen, black_checker, white_checker
    black_checker = ImageTk.PhotoImage(
        (Image.open('black-regular1.png')).resize((50, 50), Image.Resampling.LANCZOS))
    black_queen = ImageTk.PhotoImage(
        (Image.open('black-queen1.png')).resize((50, 50), Image.Resampling.LANCZOS))
    white_checker = ImageTk.PhotoImage(
        (Image.open('white-regular1.png')).resize((50, 50), Image.Resampling.LANCZOS))
    white_queen = ImageTk.PhotoImage(
        (Image.open('white-queen1.png')).resize((50, 50), Image.Resampling.LANCZOS))
    # Инициализация переменных для игры
    global take, correct
    take = False
    correct = False
    self.board = [[0, 0, 0, 0, 0, 0, 0, 0],
                  [2, 2, 2, 2, 2, 2, 2, 2],
                  [2, 2, 2, 2, 2, 2, 2, 2],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [1, 1, 1, 1, 1, 1, 1, 1],
                  [1, 1, 1, 1, 1, 1, 1, 1],
                  [0, 0, 0, 0, 0, 0, 0, 0]]
    self.current_player = [1, 3]
    self.selected_piece = None
    self.selected_rectangle = None
    self.need_attack = False
    # Создание виджетов для авторизации
    self.label = Label(text="Вход в аккаунт", bg='azure3', font=("Arial", 15))
    self.label_login = Label(text="Имя", bg='azure3', font="Arial 14")
    self.label_password = Label(text="Пароль", bg='azure3', font="Arial 14")
    self.entry_login = Entry(width=30, justify="center")
    self.entry_password = Entry(width=30, justify="center")
    self.button_auth = Button(text="Авторизироваться", bg='azure3', font="Arial 14",
                              command=lambda: self.authorization())
```

```
    self.button_reg = Button(text="Зарегистрироваться", bg='azure3', font="Arial 14",
                             command=lambda: self.registrate())
    self.button_back = Button(text="Выход", command=lambda: self.authorization())
    # Размещение виджетов на экране
    self.label.place(x=150, y=10)
    self.label_login.place(x=80, y=85)
    self.entry_login.place(x=150, y=85)
    self.label_password.place(x=70, y=135)
    self.entry_password.place(x=150, y=135)
    self.button_auth.place(x=150, y=175, width=185)
    self.button_reg.place(x=150, y=220, width=185)

# Метод для центрирования окна на экране
def center_window(self):
    screen_width = self.master.winfo_screenwidth()
    screen_height = self.master.winfo_screenheight()
    window_width = 450
    window_height = 450

    x = int((screen_width / 2) - (window_width / 2))
    y = int((screen_height / 2) - (window_height / 2))

    self.master.geometry(f"{window_width}x{window_height}+{x}+{y}")

# Метод для кодирования пароля
def password_code(self, password):
    key = 2
    coded_password = ""
    for i in password:
        coded_password_temp = chr(ord(i) + key)
        coded_password += coded_password_temp
        key = -key + 1
    return coded_password

# Метод для открытия файла с пользователями
def open_file(self):
    try:
        text = open("users.txt", "r+")
        return text
    except FileNotFoundError:
        try:
            text = open("users.txt", "w")
            text.close()
            text = open("users.txt", "r+")
            return text
        except FileNotFoundError:
```

```
            text = open("users.txt", "r+")
            return text

    # Метод для авторизации пользователя
    def authorization(self):
        login = self.entry_login.get()
        password_raw = self.entry_password.get()
        password = self.password_code(password_raw)

        # Проверка наличия введенных данных
        if len(login) == 0 and len(password) == 0:
            messagebox.showwarning(title="Ошибка!", message="Введите имя и пароль")
            return
        elif len(login) == 0 and len(password) != 0:
            messagebox.showwarning(title="Ошибка!", message="Введите имя")
            return
        elif len(login) != 0 and len(password) == 0:
            messagebox.showwarning(title="Ошибка!", message="Введите пароль")
            return

        # Открытие файла с пользователями
        file = self.open_file()
        a = file.readline()[:-1].split(" ")

        # Заполнение словаря пользователей
        while True:
            if a != [""]:
                self.users[a[0]] = a[1]
                a = file.readline()[:-1].split(" ")
            else:
                break

        # Проверка правильности введенных данных
        flag_reg = False
        for i in self.users.items():
            login_check, password_check = i
            if login == login_check and password == password_check:
                flag_reg = True
                break

        # Обработка результата авторизации
        if flag_reg:
            for widget in self.master.winfo_children():
                widget.destroy()

            Label(self.master, text="Вы успешно авторизировались!", font="Arial 16 bold").place(x=55, y=60)
```

```
            button = Button(text="Играть", command=self.do_game, bg="azure3", font="Arial 19 bold")
            button.place(x=175, y=150)
        else:
            messagebox.showwarning(title="Ошибка", message="Такого пользователя не существует")

    # Метод для регистрации нового пользователя
    def registrate(self):
        login = self.entry_login.get()
        password_raw = self.entry_password.get()
        password = self.password_code(password_raw)

        # Проверка корректности введенных данных
        if len(login) == 0 or len(password) == 0:
            messagebox.showwarning(title="Ошибка", message="Введите имя и пароль")
            return

        # Получение данных о пользователях из файла
        lines = self.open_file()
        self.users = {}
        for line in lines:
            l, p = line.strip().split(" ")
            self.users[l] = p

        # Запись нового пользователя в файл
        if login not in self.users:
            with open("users.txt", "a") as file:
                file.write(f'{login} {password}\n')
            self.show_success_message()
        else:
            messagebox.showwarning(title="Ошибка!", message="Такой аккаунт уже есть!")

    # Метод для начала игры
    def do_game(self):
        self.master.title("шашки")
        self.canvas = Canvas(self.master, width=400, height=450, bg="#666666")
        self.canvas.pack()
        self.canvas.bind("<Button-1>", self.on_click)
        self.draw_board(self.current_player)

    # Метод для отрисовки игровой доски
    def draw_board(self, current_plr):
        # Отображение текущего игрока
        if current_plr == [1, 3]:
            self.canvas.create_text(200, 425, text="Ход белых", font=("Arial", 20), fill="#cccccc")
        else:
            self.canvas.create_text(200, 425, text="Ход чёрных", font=("Arial", 20), fill="#1a1a1a")
```

```
    # Отрисовка клеток доски
    for row in range(8):
        for col in range(8):
            x1, y1 = col * 50, row * 50
            x2, y2 = x1 + 50, y1 + 50
            color = "#8a6341" if (row + col) % 2 == 0 else "#8a6341"
            self.canvas.create_rectangle(x1, y1, x2, y2, fill=color)

            piece = self.board[row][col]
            # Отображение шашек на доске
            if piece == 1:
                self.canvas.create_image(x1, y1, anchor='nw', image=white_checker)
            elif piece == 2:
                self.canvas.create_image(x1, y1, anchor='nw', image=black_checker)
            elif piece == 3:
                self.canvas.create_image(x1, y1, anchor='nw', image=white_queen)
            elif piece == 4:
                self.canvas.create_image(x1, y1, anchor='nw', image=black_queen)

# Метод для обработки щелчка мыши
def on_click(self, event):
    global take, correct
    correct = False
    col = event.x // 50 # Определение строки и столбца, по которым был произведен щелчок мыши
    row = event.y // 50 # Определение строки и столбца, по которым был произведен щелчок мыши
    # Если шашка не была выбрана
    if self.selected_piece is None: # Получение информации о клетке, по которой был произведен щелчок
        piece = self.board[row][col]
        if piece != 0: # Если на клетке есть шашка
            if piece in self.current_player: # Отображение выбранной клетки на доске
                self.selected_rectangle = self.canvas.create_rectangle(event.x // 50 * 50, event.y // 50 * 50,
                                                       event.x // 50 * 50 + 50,
                                                       event.y // 50 * 50 + 50,
                                                       outline="red", width=2)
            self.selected_piece = (row, col) # Запоминание выбранной шашки
    else:
        dest_row, dest_col = row, col
        src_row, src_col = self.selected_piece # Проверка, является ли выбранная шашка шашкой текущего игрока
        if self.board[self.selected_piece[0]][
            self.selected_piece[1]] in self.current_player: # Проверка правильности и возможности хода
            correct, take = self.v_moves(src_row, src_col, dest_row, dest_col)
            if correct:
                # Обработка перемещения шашки и обновление доски
                if self.board[src_row][src_col] == 1 and dest_row == 0:
                    self.board[dest_row][src_col] = 3
```

```
                    self.board[src_row][src_col] = 0
                elif self.board[src_row][src_col] == 2 and dest_row == 7:
                    self.board[dest_row][src_col] = 4
                    self.board[src_row][src_col] = 0
                else:
                    self.board[dest_row][dest_col] = self.board[src_row][src_col]
                    self.board[src_row][src_col] = 0
                if take: # Если было выполнено взятие шашки, обновление игрового поля и отображение выбранной
клетки
                    self.selected_piece = dest_row, dest_col
                    self.canvas.delete("all")
                    self.draw_board(self.current_player)
                    self.selected_rectangle = self.canvas.create_rectangle(dest_col * 50, dest_row * 50,
                                                           dest_col * 50 + 50, dest_row * 50 + 50,
                                                           outline="blue", width=2)
                else: # Если взятия не было, переключение текущего игрока и проверка завершения игры
                    self.switch_player()
                    self.selected_piece = None
                    self.canvas.delete("all")
                    self.draw_board(self.current_player)
                    self.need_attack = False
                    self.check_win()
                    self.attack(take, dest_row, dest_col)
                else: # Если ход недопустим, сброс выбранной шашки и обновление игрового поля
                    self.selected_piece = None
                    self.canvas.delete("all")
                    self.draw_board(self.current_player)
            else: # Если выбранная шашка не принадлежит текущему игроку, сброс выбора и обновление поля
                self.selected_piece = None
                self.canvas.delete("all")
                self.draw_board(self.current_player)

    # Создание экземпляра класса и запуск программы
    if __name__ == "__main__":
        root = Tk()
        app = CheckersBoard(root)
        root.mainloop()

        def main():
            # Основная функция запуска игры
            root = Tk() # Создание корневого окна
            app = CheckersBoard(root) # Создание экземпляра игры
            root.mainloop() # Запуск основного цикла программы

        if __name__ == "__main__":
            main()
```