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 игра «Турецкие Шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2411-12 ТЗ-01

Листов 52

Руководитель разработки:

кандидат технических наук, доцент кафедры «Измерительно-вычислительные комплексы»,  
Шишкин Вадим Викторинович

« » 2023 г.

Исполнитель:

студент гр. ИСТбд-21

*Прокофьев Алексей Александрович*

« » 2023 г.

2023

Содержание

Аннотация……...…………………………………………………………..3

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

Пояснительная записка...…………………………………………….....11

Руководство пользователя……………………………….…………....15

Текст программы…..………………………………………………….....23

**Аннотация**

*Задание на курсовую работу:* разработка компьютерной игры «Турецкие шашки».

*Суть задания:* необходимо разработать приложение, которое соответствует правилам настольной игры «Турецкие шашки». Оно должно представлять собой окно, представляющую шахматную доску с интуитивно понятные интерфейсом и основным функционалом.

*Ключевые функции приложения:*

1. Авторизация/регистрация пользователя.
2. Отображение игрового поля.
3. Проверка ходов игрока.
4. Ходы искусственного интеллекта.
5. 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 игра «Турецкие шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2411-12 ТЗ-01

Листов 6

Исполнитель:

студент гр. ИСТбд-21

*Прокофьев Алексей Александрович*

« » 2023 г.

2023

# Введение

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

**Поле и игроки.** Играют двое игроков. Игра ведется на шашечном поле. Шашечная доска состоит из 64 (8х8) 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 обозначаетсясимволом 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**. 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. Возможна ничья по соглашению игроков.

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.

## **2.2.3 Требования к организации информационного обеспечения, входных и выходных данных**

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Отдельно выделены папки под графические файлы, шрифт, заготовку объектов и карты, аудио эффектов, а также для самого кода. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде.

### 2.3 Требования к надёжности

Поддержка непрерывной и стабильной работы компьютера.

**2.4 Требования к информационной и программной совместимости**

Рекомендуется к использованию на Windows 7.

При создании программы используются встроенные библиотеки “random”, “os”. И сторонние библиотеки “tkinter 8.6.”. Разработка ведётся в “PyCharm community edition 2022.3” на версии языка программирования Python 3.9.

**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**

Диск CD-R должен храниться при комнатной температуре, в диапазоне от 20°C до 25°C. Рекомендуется хранить диск в условиях с относительной влажностью воздуха от 20% до 50%. Диск CD-R должен храниться в темном месте, защищенном от прямых солнечных лучей и других источников яркого света. Для предотвращения повреждения диска CD-R рекомендуется хранить его в специальных пластиковых коробках или футлярах, предназначенных для хранения CD-дисков.

**2.6 3 Сроки хранения**

Срок хранения – до июля 2024 года**.**

# 3. Требования к программной документации

1. «Техническое задание» на реализуемое приложение должно соответствовать ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению»;
2. «Пояснительная записка» должна соответствовать ГОСТ 19.404-79 «Пояснительная записка. Требования к содержанию и оформлению»;
3. «Руководство программиста» должна соответствовать ГОСТ 19.504-79

«Руководство программиста. Требования к содержанию и оформлению»;

1. Оформление программного кода приложения должно быть в соответствии с ГОСТ 19.401-79 «Текст программы. Требования к содержанию и оформлению».

# 4. Стадии и этапы разработки

1. Написание технического задания;
2. Написание пояснительной записки;
3. Разработка и отладка прототипа приложения;
4. Разработка и отладка окончательного варианта приложения;
5. Написание руководства программиста;
6. Оформление пояснительной записки курсовой работы;

7) Подготовка презентации.

8) Защита курсовой работы.

# 5. Порядок контроля и приёмки

Созданный программный продукт предоставляется студентом руководителю с помощью репозитория GitHub, как в исходном виде, так и преобразованный в исполняемый файл «exe». В случае наличия каких-либо замечаний, студент может доработать свой программный продукт, записав новую версию по той же ссылке. Программные документы также размещаются в репозитории GitHub. Окончательная (принятая) версия документации предоставляется в печатном виде.

График защиты курсовых работ составляется руководителем и доводится до сведения студентов. Студент обязан убедиться в корректности функционирования приложения до защиты. Проблемы, обнаруженные непосредственно при защите, являются основанием для её переноса (в соответствии с графиком).

В начале защиты студент сообщает название приложения, кратко формулирует его назначение и указывает основные особенности. В процессе защиты – демонстрирует используемую(мые) структуры данных и алгоритмы, акцентируя внимание на наиболее важных и интересных, демонстрирует работоспособность самого приложения.

Оценка работы осуществляется руководителем с учётом качества её выполнения, включая качество документации, полноты учёта предъявляемых требований, выступления с презентацией и ответов на вопросы в ходе защиты, а также соблюдения сроков выполнения и защиты курсовой работы, включая работу на практических занятиях.

Основаниями для получения неудовлетворительной оценки могут являться серьёзное несоответствие программного продукта предъявляемым требованиям, неработоспособность приложения, наличие существенных элементов заимствования из чужих работ как в программном коде или интерфейсе приложения, так и в документации, а также слабая ориентация студента в представляемой работе. При неудовлетворительной оценке руководитель определяет направления и объём доработки программного продукта.

**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 игра «Турецкие шашки»

**Пояснительная записка**

Р.02069337.22/2411-12 ПЗ-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 3

**Исполнитель**:

студент гр. ИСТбд-21

*Прокофьев Алексей Александрович*

« » 2023 г.

2023

**Введение**

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

При разработке "Турецкие шашки" был выбран подход, основанный на математической модели игры "Реверси". 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).

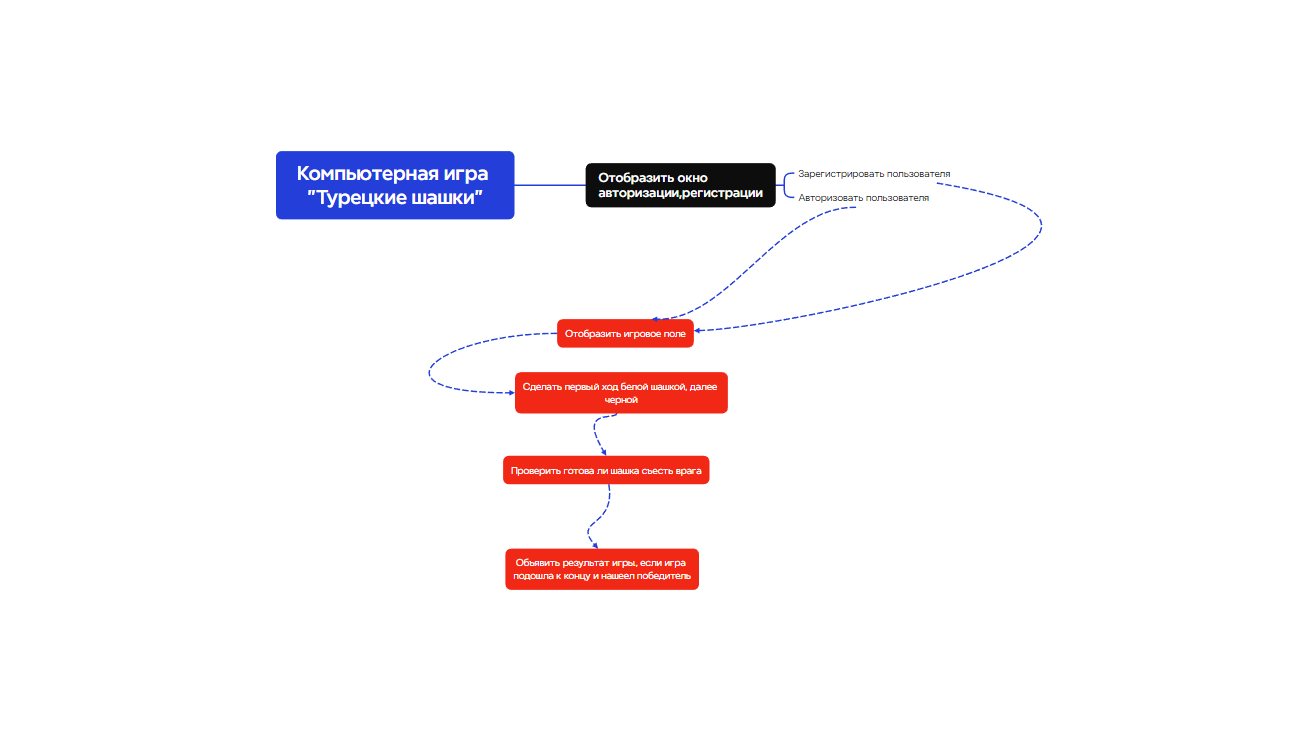
**1.2 Архитектура и алгоритмы**

1.2.1. Архитектура

Классы и структура данных:

CheckersBoard: этот класс представляет окно авторизации/регистрации пользователя. Он содержит методы для отображения окна, проверки данных о зарегистрированных пользователях и последующей авторизации, а также метод регистрации пользователя. В этом же классе представлена сама игра, логика игры, ходы и полная реализация игры «Турецкие шашки».

**1.3 Тестирование**



1.3.1 Описание отчета о тестировании

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

1.3.2 Цель тестирования

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

1.3.3 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.

1.3.4 Проведенные тесты

Тестирование функциональности:

- Цель: проверить основную функциональность игры "Турецкие шашки".

- Шаги выполнения:

1. Открыть приложение "Турецкие шашки".

2. Зарегистрировать нового пользователя.

3. Начать игру.

4. Выполнить ходы и найти победителя.

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

**2. Источники, использованные при разработке**

1. Википедия - <https://ru.wikipedia.org/wiki/Турецкие_шашки>

2. Турецкие шашки правила игры - http://wmsg.ru/checkers1/turkish-draughts/

**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 игра «Турецкие шашки»

**Руководство программиста**

Р.02069337.22/2411-12 РП-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 36

Исполнитель:

студент гр. ИСТбд-21

*Прокофьев Алексей Александрович*

« » 2023 г.

2023

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

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

Свод правил игры "Реверси":

* Игра проводится на игровом поле размером 8x8 клеток.
* У каждого игрока есть фишки одного цвета, обычно черного и белого.
* Игроки ходят по очереди, ставя фишки на свободные клетки поля.
* Ход делается путем размещения фишки на клетке таким образом, чтобы между новой фишкой и уже установленными фишками противника находились фишки этого же цвета.
* Фишки противника, попавшие между новой фишкой и другими фишками текущего игрока, меняют цвет на цвет текущего игрока.
* Цель игры - иметь на игровом поле больше фишек своего цвета, чем у противника, когда все клетки заполнены.

Общая характеристика функциональных возможностей приложения:

1. Игра с компьютерным противником: приложение предоставляет возможность играть против компьютерного противника. Компьютерный противник может использовать различные алгоритмы и стратегии для принятия решений и выполнения ходов.
2. Игра с другими игроками: Приложение позволяет играть в многопользовательском режиме, где пользователи могут играть против друг друга на одном устройстве или по сети.

**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: 271.

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).

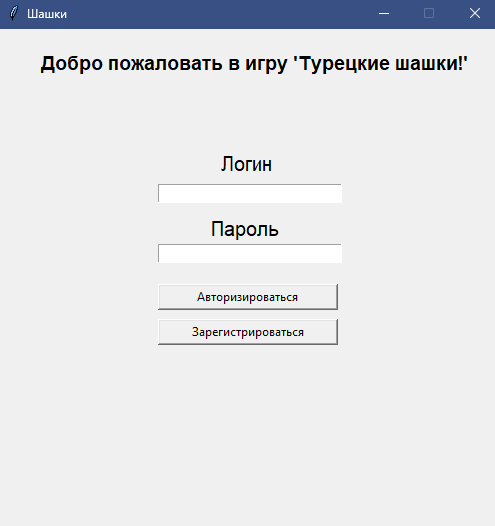


Рис. 1

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.

После успешной авторизации/регистрации появляется окно (рис.2), где предлагается начать игру.

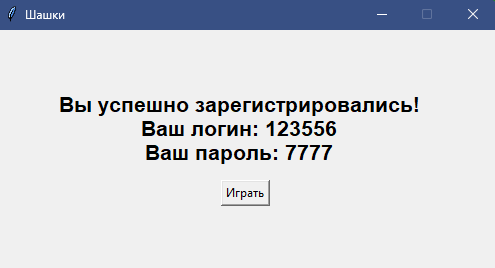


Рис. 2

После нажатия кнопки «Играть» появляется окно (рис. 3): на котором расположено игровое поле размеров 8x8 клеток, а также блок с информацией об игре, где указано, кто сейчас делает ход.

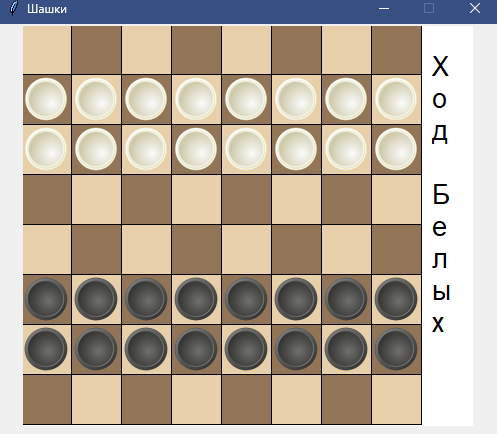


Рис. 3

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

Когда пользователь попадает на своей игровой шашкой на поле дамки, то шашка меняет свой внешний вид, чтобы игроки могли отличить ее от обычной шашки (рис. 4).

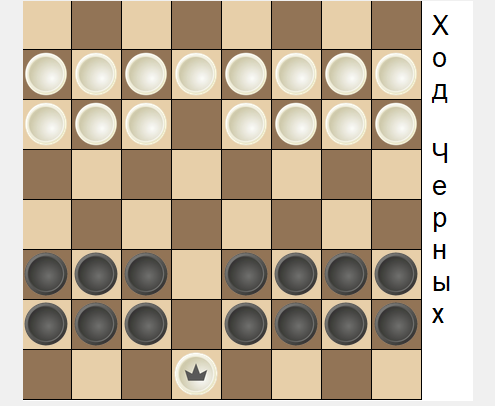
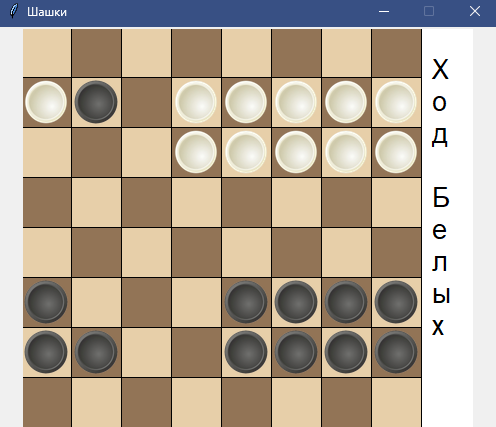
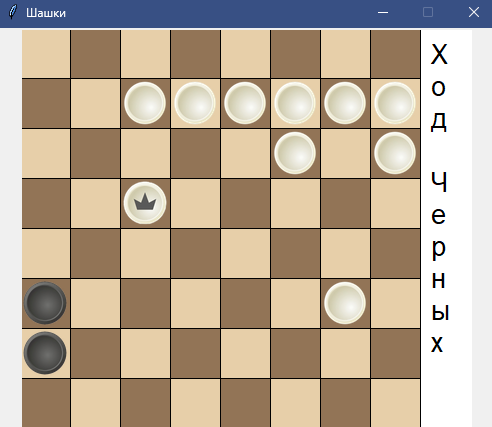
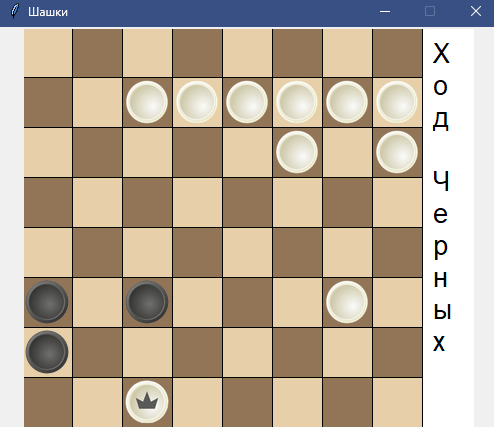


Рис. 4

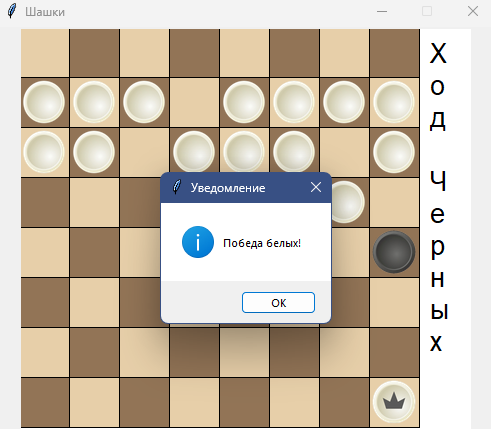
Если шашка стоит рядом с шашкой противника и она может «съесть» соперника, то пользователь не может сходить другой шашкой по правилам игры, а «съесть» появившуюся шашку, то есть «жертву» игрока(рис. 5).

*Рис.5*

Дамка может ходить на любую клетку по горизонтали или вертикали по соответствующим правилам игры. «Есть» соперника она может также на любую клетку, но только не назад от направления предыдущего хода(рис. 6).

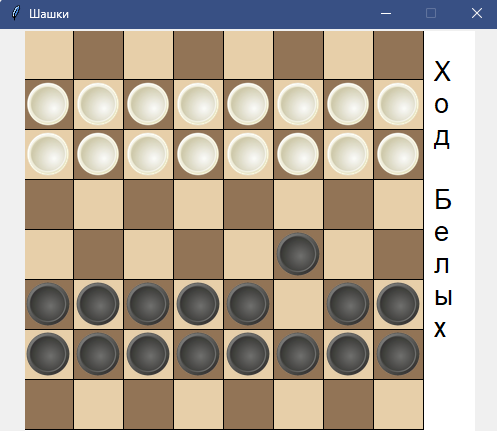
*Рис. 6*

Игра считается завершённой, когда у соперника остается либо одна шашка, либо когда пользователь сумел «съесть» все шашки соперника.(рис. 7).



*Рис. 7*

После нажатия на кнопку «ОК» игра запускается заново и ход начинают черные шашки



Если на доске осталось по одной шашке — объявляется ничья.

**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, 2 – чёрная шашка, 1 – белая шашка). Использование массива обеспечивает эффективный доступ к элементам и удобство при выполнении операций на игровом поле.

Альтернативный вариант: вместо двумерного массива для игрового поля можно использовать одномерный массив с преобразованием индекса клетки в двумерные координаты. Это может упростить некоторые операции, но может быть менее интуитивно понятным при работе с матрицей.

**3. Обращение к программе**

1. Класс «CheckersBoard»:

**-** методы:

-\_\_init\_\_(): создание окна авторизации/регистрации пользователя, шахматной доски.

- center\_window(): вывод окна по центру экрана.

- password\_code(): используется для кодирования (шифрования) строки пароля перед сохранением или сравнением с предварительно закодированным паролем.

- open\_file(): создание файл для записи и считывания логина и пароля.

- authorization(): проверка на ошибки при вводе данных пользователя, если их нет, то пользователь успешно авторизовался.

- registrate(): произвести запись данных зарегистрированного пользователя.

-do\_game(): настраивает окно и холст для игры шашек, назначает обработчик событий и вызывает метод отрисовки доски для начала игры.

- draw\_board(): используется для отрисовки игровой доски на холсте (`Canvas`) приложения. Он также отображает информацию о текущем игроке в правом верхнем углу доски.

-on\_click(): используется для обработки события щелчка мыши на игровой доске. Он определяет логику выбора и перемещения фигур, обновление состояния игры и проверку на победу.

- fix\_attack(): используется для проверки возможности атаки после выполнения хода и обновления значений флага `self.need\_attack`.

- is\_valid\_move(): используется для ходов остальных шашек по правилам игры.

- switch\_player(): используется для переключения текущего игрока между двумя игроками в игре шашек.

- check\_win(): проверка победителя.

- check\_col(): используется для проверки колонок на наличие фигур и возможность хода в игре шашек. Возвращает список с двумя значениями: True/False (можно ли сделать ход) и индекс следующего доступного поля (если есть).

- check\_row(): используется для проверки рядов на наличие фигур и возможность хода в игре шашек. Возвращает список с двумя значениями: True/False (можно ли сделать ход) и индекс следующего доступного поля (если есть).

Библиотека os:

Библиотека os предоставляет функции для взаимодействия с операционной системой. В данном коде она используется для выполнения операций, связанных с обработкой файлов. Конкретно, функция os.path.exists используется для проверки существования файла. Это полезно для проверки наличия файлов при регистрации пользователей и процессе аутентификации.

Библиотека tkinter:

Библиотека tkinter (сокращение от "Tk interface") является стандартной библиотекой Python для создания графического интерфейса пользователя (ГИП). Она предоставляет различные виджеты и методы для создания окон, кнопок, текстовых полей, элементов меню и т.д. в приложениях. В данном коде библиотека tkinter используется для создания окон и элементов графического интерфейса игры в шашки. В частности, библиотека tkinter используется для создания следующих элементов:

- Tk: класс окна, представляющий главное окно приложения.

- Button: виджет для создания кнопок в окне.

- Canvas: виджет, позволяющий отображать и рисовать графические объекты, такие как прямоугольники и круги.

- Image и ImageTk из библиотеки PIL: используются для работы с изображениями.

Библиотека messagebox:

Модуль messagebox является частью библиотеки tkinter и предоставляет функции для отображения диалоговых окон или сообщений пользователю. В данном коде он используется для отображения предупреждающих сообщений с помощью функции showwarning. Эти сообщения могут быть полезными для уведомления пользователя об ошибках или проблемах при работе приложения.

Библиотеки PIL (Python Imaging Library) и ImageTk:

Библиотека PIL используется для выполнения задач по обработке изображений, в то время как ImageTk является модулем в библиотеке PIL, который предоставляет поддержку работы с изображениями в tkinter. В данном коде эти библиотеки импортируются для использования изображений в графическом интерфейсе. Модуль ImageTk позволяет отображать изображения в окнах tkinter с помощью класса ImageTk.PhotoImage.

**4. Сообщения**

«Победили белый/черные!».

«Игра окончена в ничью!».

**Текст программы**

from tkinter import \*

import os

from tkinter import messagebox

from PIL import Image, \

ImageTk

class CheckersBoard:

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

self.master = master

self.master.title("Шашки")

self.master.resizable(False, False) # Запрет изменения размеров окна

self.center\_window() # Центрирование окна

self.canvas = Canvas(self.master, width=800, height=600) # Инициализация canvas

self.canvas.pack()

self.users = {}

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))

global vzyat, correct

vzyat = False

correct = False

self.board = [[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],

[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]]

self.current\_player = [1, 3]

self.selected\_piece = None

self.selected\_rectangle = None

self.need\_attack = False

self.label = Label(text="Добро пожаловать в игру 'Турецкие шашки!'", font="Arial 14 bold")

self.label\_login = Label(text="Логин", font="Arial 15")

self.label\_password = Label(text="Пароль", font="Arial 15")

self.entry\_login = Entry(width=30, justify="center")

self.entry\_password = Entry(width=30, justify="center")

self.button\_auth = Button(text="Авторизироваться", command=lambda: self.authorization())

self.button\_reg = Button(text="Зарегистрироваться", command=lambda: self.registrate())

self.button\_back = Button(text="Выход", command=lambda: self.authorization())

self.label.place(x=40, y=20)

self.label\_login.place(x=220, y=120)

self.entry\_login.place(x=160, y=155)

self.label\_password.place(x=210, y=185)

self.entry\_password.place(x=160, y=215)

self.button\_auth.place(x=160, y=255, width=180)

self.button\_reg.place(x=160, y=290, width=180)

def center\_window(self):

screen\_width = self.master.winfo\_screenwidth()

screen\_height = self.master.winfo\_screenheight()

window\_width = 500

window\_height = 500

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}")

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

def open\_file(self):

try:

text = open("users\_checker\_file.txt", "r+")

return text

except FileNotFoundError:

try:

text = open("users\_checker\_file.txt", "w")

text.close()

text = open("users\_checker\_file.txt", "r+")

return text

except FileNotFoundError:

text = open("users\_checker\_file.txt", "r+")

return text

def authorization(self):

login = self.entry\_login.get()

password\_raw = self.entry\_password.get()

password = self.password\_code(password\_raw)

if len(login) == 0 and len(password) == 0:

messagebox.showwarning(title="Ошибка", message="Введите логин и пароль")

return

elif len(login) == 0 and len(password) != 0:

messagebox.showwarning(title="Ошибка", message="Введите логин")

return

elif len(login) != 0 and len(password) == 0:

messagebox.showwarning(title="Ошибка", message="Введите пароль")

return

file = self.open\_file()

a = file.readline()[:-1].split(" ")

while True:

if a != [""]:

self.users[a[0]] = a[1]

a = file.readline()[:-1].split(" ")

else:

break

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

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=90, y=60)

button = Button(text="Играть", command=self.do\_game)

button.place(x=225, y=150)

else:

messagebox.showwarning(title="Ошибка", message="Такого пользователя не существует")

def registrate(self):

login = self.entry\_login.get()

password\_raw = self.entry\_password.get()

password = self.password\_code(password\_raw)

if len(login) == 0 and len(password) == 0:

messagebox.showwarning(title="Ошибка", message="Введите желаемые логин и пароль")

elif len(login) == 0 and len(password) != 0:

messagebox.showwarning(title="Ошибка", message="Введите логин")

elif len(login) != 0 and len(password) == 0:

messagebox.showwarning(title="Ошибка", message="Введите пароль")

else:

file = self.open\_file()

temp = file.readline()[:-1].split(' ')

while True:

if temp != [""]:

self.users[temp[0]] = temp[1]

temp = file.readline()[:-1].split(' ')

else:

break

flag\_reg = False

for i in self.users.items():

l, p = i

if login == l:

flag\_reg = True

if not flag\_reg:

file = self.open\_file()

file.seek(0, os.SEEK\_END)

file.write(f'{login} {password}\n')

file.close()

self.canvas.delete("all")

for widget in self.master.winfo\_children():

widget.destroy()

Label(text=f"Вы успешно зарегистрировались!\nВаш логин: {login}\nВаш пароль: {password\_raw}",

font="Arial 16 bold").place(x=60, y=60)

button = Button(text="Играть", command=self.do\_game)

button.place(x=225, y=150)

else:

messagebox.showwarning(title="Ошибка", message="Такой аккаунт уже существует")

def do\_game(self):

self.master.title("Шашки")

self.canvas = Canvas(self.master, width=450, height=400, bg="white")

self.canvas.pack()

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

self.draw\_board(self.current\_player)

def draw\_board(self, current\_plr):

if current\_plr == [1, 3]:

self.canvas.create\_text(420, 170, text="Х\nо\nд\n\nБ\nе\nл\nы\nх", font=("Arial", 20), fill="black")

else:

self.canvas.create\_text(420, 170, text="Х\nо\nд\n\nЧ\nе\nр\nн\nы\nх", font=("Arial", 20), fill="black")

for row in range(8):

for col in range(8):

x1, y1 = col \* 50, row \* 50

x2, y2 = x1 + 50, y1 + 50

color = "#E7CFA9" if (row + col) % 2 == 0 else "#927456"

self.canvas.create\_rectangle(x1, y1, x2, y2, fill=color)

piece = self.board[row][col]

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)

def on\_click(self, event):

global vzyat, correct

correct = False

col = event.x // 50

row = event.y // 50

print(self.selected\_piece)

print(self.current\_player, self.need\_attack)

if self.selected\_piece is None:

piece = self.board[row][col]

if piece != 0:

# создание обводки

if piece in self.current\_player:

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="yellow", width=2)

self.selected\_piece = (row, col)

else:

dest\_row, dest\_col = row, col

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

if self.board[self.selected\_piece[0]][

self.selected\_piece[1]] in self.current\_player: # проверка выбора своей шашки игрока

correct, vzyat = self.is\_valid\_move(src\_row, src\_col, dest\_row, dest\_col)

print(correct, vzyat)

if correct:

print('AAAAAAAAAAAAAAAAAAAAA')

if self.board[src\_row][src\_col] == 1 and dest\_row == 7: # становление дамкой у красных

self.board[dest\_row][src\_col] = 3

self.board[src\_row][src\_col] = 0

print('Белые стали дамкой')

elif self.board[src\_row][src\_col] == 2 and dest\_row == 0: # становление дамкой у черных

self.board[dest\_row][src\_col] = 4

self.board[src\_row][src\_col] = 0

for i in range(len(self.board)):

print(self.board[i])

print('Черные стали дамкой')

else:

self.board[dest\_row][dest\_col] = self.board[src\_row][src\_col]

self.board[src\_row][src\_col] = 0

if vzyat:

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="yellow", width=2)

else:

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.fix\_attack(vzyat, dest\_row, dest\_col)

for i in range(len(self.board)):

print(self.board[i])

else:

self.selected\_piece = None

self.canvas.delete("all")

self.draw\_board(self.current\_player)

else:

self.selected\_piece = None

self.canvas.delete("all")

self.draw\_board(self.current\_player)

def fix\_attack(self, vzyat, row, col):

if not vzyat:

if self.current\_player == [1, 3]: # проверка обязаности белых

for i in range(8):

for j in range(8):

try:

if self.board[i][j] in [2, 4]:

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

for i in range(8):

for j in range(8):

try:

if self.board[i][j] in [2, 4]:

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

if self.current\_player == [2, 4]: # проверка обязанности черных

print('ЧЕКАЮ')

for i in range(8):

for j in range(8):

try:

if self.board[i][j] in [1, 3]:

if self.board[i - 1][j] == 0: # проверка атаки вперед

for k in range(i + 1, 8):

if self.board[k][j] == 4: # проверка для дамки

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: # проверка атаки назад дамкой

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

for i in range(8):

for j in range(8):

try:

if self.board[i][j] in [1, 3]:

if self.board[i][j - 1] == 0: # проверка атаки влево

for k in range(j + 1, 8):

if self.board[i][k] == 4: # проверка для дамки

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: # проверка атаки вправо

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 is\_valid\_move(self, src\_row, src\_col, dest\_row, dest\_col):

print('тута')

# Allow vertical (up or down) and horizontal (left or right) moves

if self.board[src\_row][src\_col] == 1 and self.board[dest\_row][dest\_col] == 0: # проверки для белых пешек

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): # ход на пустую клетку

if not self.need\_attack:

return [True, False]

elif (dest\_row - src\_row == 2 and abs(dest\_col - src\_col) == 0 and # взятие по прямой

self.board[dest\_row - 1][dest\_col] in [2, 4]):

self.board[dest\_row - 1][dest\_col] = 0

if self.need\_attack:

self.need\_attack = False

try: # проверка есть ли дальше возможность атаковать

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 ход вправо

self.board[dest\_row][dest\_col - 1] = 0

if self.need\_attack:

self.need\_attack = False

try: # проверка есть ли дальше возможность атаковать

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 ход влево

self.board[dest\_row][dest\_col + 1] = 0

if self.need\_attack:

self.need\_attack = False

try: # проверка есть ли дальше возможность атаковать

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 self.board[src\_row][src\_col] == 3 and self.board[dest\_row][dest\_col] == 0: # првоерка для белой дамки

print('проверка')

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): # ход на пустую клетку

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: # если ход был вниз

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): # вправо

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): # влево

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: # если ход был вверх

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): # вправо

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): # влево

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 vzyat:

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: # если ход был вправо

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): # вниз

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): # вверх

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: # если ход был влево

print('МИУ')

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): # вниз

if self.board[i][dest\_col] in [2, 4] and self.board[i + 1][dest\_col] == 0:

print('МИУ2')

print(self.board[i][dest\_col])

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): # вверх

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 vzyat:

return [False, False]

elif self.board[src\_row][src\_col] == 2 and self.board[dest\_row][dest\_col] == 0: # проверки для черных пешек

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): # ход на пустую клетку

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]): # взятие по прямой

self.board[dest\_row + 1][dest\_col] = 0

if self.need\_attack:

self.need\_attack = False

try: # проверка есть ли дальше возможность атаковать

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 ход вправо

self.board[dest\_row][dest\_col - 1] = 0

if self.need\_attack:

self.need\_attack = False

try: # проверка есть ли дальше возможность атаковать

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 ход влево

self.board[dest\_row][dest\_col + 1] = 0

if self.need\_attack:

self.need\_attack = False

try: # проверка есть ли дальше возможность атаковать

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: # првоерка для черной 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): # ход

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: # если ход был вниз

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): # вправо

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): # влево

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: # если ход был вверх

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): # вправо

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): # влево

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 vzyat:

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: # если ход был вправо

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): # вниз

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): # вверх

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: # если ход был влево

print('МИУ')

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): # вниз

if self.board[i][dest\_col] in [1, 3] and self.board[i + 1][dest\_col] == 0:

print('МИУ2')

print(self.board[i][dest\_col])

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): # вверх

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 vzyat:

return [False, False]

else:

if not vzyat:

return [False, False]

def switch\_player(self):

if self.current\_player == [1, 3]:

self.current\_player = [2, 4]

else:

self.current\_player = [1, 3]

def check\_win(self):

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 > 1 >= count2:

messagebox.showinfo("Уведомление", "Победа белых!")

self.board = [[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],

[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]]

self.canvas.delete("all")

self.draw\_board(self.current\_player)

elif count2 > 1 >= count:

messagebox.showinfo("Уведомление", "Победа черных!")

self.board = [[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],

[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]]

self.canvas.delete("all")

self.draw\_board(self.current\_player)

elif count == count2 == 1:

messagebox.showinfo("Уведомление", "Играла закончилась ничьей!")

self.board = [[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],

[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]]

self.canvas.delete("all")

self.draw\_board(self.current\_player)

else:

count = 0

count2 = 0

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 черных!")

self.board = [[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],

[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]]

self.canvas.delete("all")

self.draw\_board(self.current\_player)

if count2 == 0:

messagebox.showinfo("Уведомление", "Победа белых!")

self.board = [[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],

[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]]

self.canvas.delete("all")

self.draw\_board(self.current\_player)

def check\_col(self, start, end, col):

count = 0

index = 0

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:

if self.board[i][col] in self.current\_player:

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):

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:

if self.board[row][i] in self.current\_player:

count += 1

index = i + 1

count += 1

if count == 0:

return [True, False]

elif count == 1 and correct\_ind == index - 1:

return [True, index]

else:

return [False, False]

def main():

root = Tk()

app = CheckersBoard(root)

root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main()