**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 «Мак-Йек – Поддавки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2**413**-24

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-23

*Карась Илья Юрьевич*

« » 2023 г.

**2023**

**Содержание**

Аннотация.......................................................................................1

Техническое задание.....................................................................3

Пояснительная записка ................................................................8

Руководство программиста ..........................................................14

Текст программы............................................................................26

**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 «Мак-Йек – Поддавки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2413-24 ТЗ-02

**Исполнитель**:

студент гр. ИСТбд-23

*Карась И. Ю.*

« » 2023 г.

**2023**

**Введение**

Приложение «Мак-Йек-Поддавки» реализовано для развлечения пользователей. Приложение реализовано по следующим правилам:

**Поле и игроки.** Играет один игрок против комьютера. Игра ведется на шахматном поле. Шашечная доска состоит из 64 (8х8) одинаковых клеток, чередующиеся цветами: белые (светлого цвета), черные (темного цвета). Игра ведется только по черным клеткам.

**Шашки.** Шашки подразделяются на простые и дамки. Дамки маркируются знаком короны. Перед началом игры игрокам предоставляется по 8 простых шашек: одному — белых, другому — черных.

**Стартовая позиция.** Шашки расставляются на темных клетках первых двух горизонтальных рядов с каждой стороны.

**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 только вперед на соседнее поле по диагонали. В случае, когда простая шашка достигает одной из клеток последнего, восьмого (считая от себя) горизонтального ряда, она превращается в дамку и получает новые права. Дамка обозначаетсясимволом короны. Дамка, в отличие от простой шашки, ходит на любое из свободных (не занятых другими шашками) полей по диагонали в любом направлении (как вперед, так и назад), 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. Шашка другого игрока в этом случае снимается с поля. Если дамка находится на одной диагонали рядом или на расстоянии от шашки другого игрока, за которой имеется одно или несколько свободных клеток, она должна быть перенесена через эту шашку на первую свободную клетку. Шашка другого игрока в этом случае снимается с поля. Взятие шашки другого игрока является обязательным и производится как вперед, так и назад. Взятие считается одним сыгранным ходом. Взятие своих шашек запрещается. Взятие должно быть четко обозначено и проводится в определенном порядке. Оно считается законченным после снятия шашки соперника с поля. Если в процессе взятия шашкой она вновь оказывается на одной диагонали рядом с другой шашкой игрока, за которой имеется свободная клетка, она должна быть перенесена через вторую, третью и т.д. шашку. Если в процессе взятия дамкой она снова оказывается на одной диагонали рядом или на расстоянии от шашки другого игрока, за которой находится одна или несколько свободных клеток, дамка должна быть обязательно перенесена через вторую, третью и т.д. шашку и занять любую свободную клетку на той же диагонали за последней взятой шашкой. После завершения взятия взятые шашки другого игрока 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 достигает последнего (восьмого от себя) горизонтального ряда, ход игрока(компьютера) заканчивается, даже если у него есть возможность продолжить/начать взятие.

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

### **1. Основания для разработки**

Учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету «О закреплении тем курсовых работ (проектов) за студентами 2 курса ФИСТ направления 09.03.02 «Информационные системы и технологии» (профиль Информационные системы и технологии) по дисциплине «Алгоритмы и структуры данных» от 20.10.2023

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

**2.1. Функциональное назначение**

Требуется разработать десктопное приложение для одного пользователя 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;

- калькуляция хода противника;

- проверка окончания игры;

- 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. 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 “enum”, “ hashlib”, “json”.

И сторонние библиотеки “tkinter 8.6.”.

Разработка ведётся в “PyCharm community edition 2022.3” на версии языка программирования Python 3.10.

**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**

Определяется заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяется заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяется 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 «Мак-Йек – Поддавки»

Пояснительная записка

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2413-24 ПЗ-01

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2023 г.

**Исполнитель**:

студент гр. ИСТбд-23

*Карась Илья Юрьевич*

« » 2023 г.

**2023**

**Введение**

Приложение «Мак-Йек-Поддавки» реализует функционал логической игры «Мак-Йек – Поддавки»

В качестве подхода для разработки была выбрана каскадная модель разработки («Водопад»). Каскадная модель была выбрана из-за своей простоты, она позволяет наглядно представить объём работы и сроки выполнения, а также эффективно разбить проект на несколько подзадач.

Приложение «Мак-Йек-Поддавки» представляет собой игру Поддавки на 64 клеточном поле (8 клеток в ширину и 8 в длину) для игры одного игрока против компьютера.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу

**1.2 Математические методы**

В качестве математической модели для представления и отрисовки поля был выбран список, он позволяет легко записать положение всех шашек, выполнять все необходимые проверки и отрисовать шашки, а также может быть легко изменен, что упрощает вывод хода пользователя. В списке пустые клетки представлены значение none, белые шашки white\_regular, белые дамки white\_queens, черные шашки black\_regular, черные дамки black\_ queens.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура



1. Регистрация/Авторизация.

Интерфейс, созданный с помощью библиотеки Tkinter позволяющий пользователю зарегистрировать новый аккаунт и сыграть с помощью уже существующего.

2. Отрисовка основных виджетов.

Вывод на экран игрового поля с расставленными шашками

3. Обработчик событий.

После хода происходит калькуляция ходов компьютера с помощью алгоритма MiniMax.

4. Отрисовка ходов.

Вывод на экран ходов пользователя и компьютера.

5. Определение конца игры.

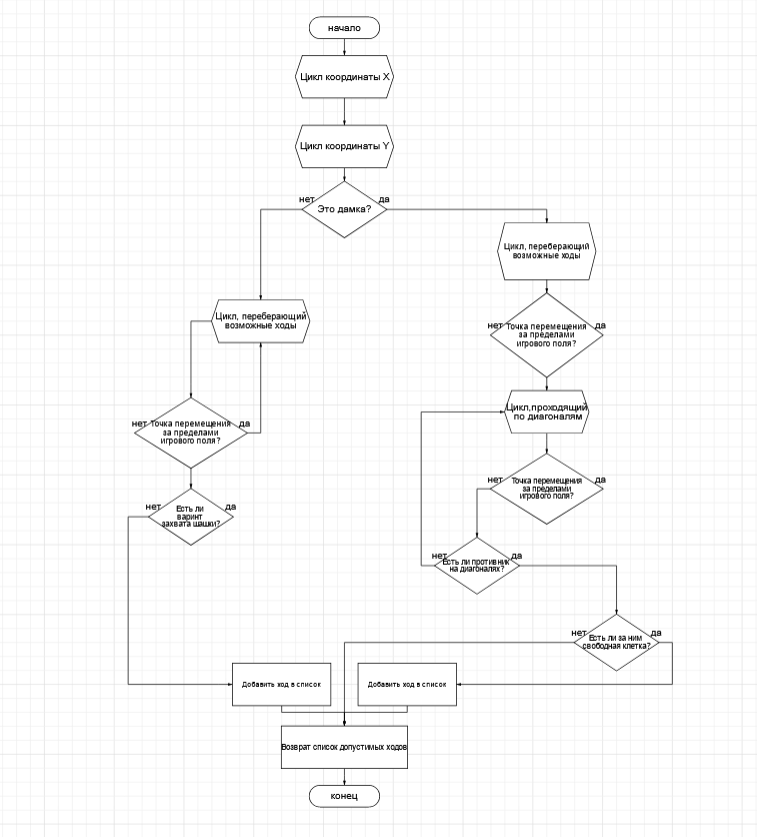
Алгоритм проверки окончания игры, проверяет после хода пользователя закончена ли игра, в случае конца игры выводит на экран победителя.

1.3.2.1. Алгоритм проверки регистрации

Данный алгоритм необходим для проверки регистрации пользователя. Алгоритм принимает логин и пароль, шифрует пароль, сверяет полученные данные с данными о зарегистрированных пользователях. Если пользователь с таким логином найден и пароль соответствует записанному в файл с зарегистрированными, то алгоритм позволяет начать игру, иначе выводит предупреждение о том, что данные некорректны.

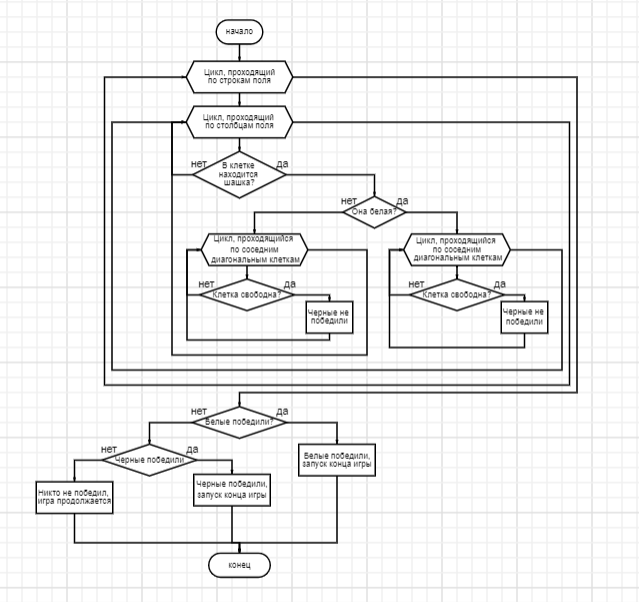
1.3.2.3. Алгоритм проверки доступности хода

Данный алгоритм необходим для контроля за соблюдением пользователем правил игры. Алгоритм получает на вход два переменные. Первая переменная указывает на координату по оси X, а вторая на координату по Оси Y. Алгоритм проверяет все возможные ходы и, если он возможен добавляет его в список, иначе ничего не делает и проверяет до тех пор, пока не будут известны все возможные ходы.



1.3.2.4. Алгоритм проверки конца игры.

Данный алгоритм необходим для того, чтобы определить победителя и корректно закончить игру. Алгоритм проверяет наличие шашек и ходов и в случае их отсутствия запускает сценарий конца игры, иначе ничего не делает и игра продолжается.



**1.4 Тестирование**

Весь процесс тестирования проходил вручную, без привлечения специального ПО. На протяжении всего хода разработки, использовался метод белого ящика, так как в любом время имелся доступ ко всем компонентам программы. Всё тестирование выполнялось интуитивным методом, без подготовки специальных тестов.

На протяжении всего хода разработки, по мере добавления новых функций программы, использовалось системное тестирование новых функций, для устранения возникших в ходе написания ошибок. После положительных результатов тестирования функция считалась внедренной.

**2. Источники, использованные при разработке**

Разработка логических компьютерных игр с графическим интерфейсом в среде Питон: учебное пособие для студентов направления 09.03.02 «Информационные системы и технологии» / В.В. Шишкин, Д.С. Афонин. – Ульяновск : УлГТУ, 2023. – 88 с.

**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 «Мак-Йек – Поддавки»

Руководство программиста

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.22/2413-24 РП-01

**Исполнитель**:

студент гр. ИСТбд-23

*Карась Илья Юрьевич*

« » 2023 г.

**2023**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение «Мак-Йек-Поддавки» реализовано для развлечения пользователей. Приложение реализовано по следующим правилам:

**Поле и игроки**. Играет один игрок против комьютера. Игра ведется на шахматном поле. Шашечная доска состоит из 64 (8х8) одинаковых клеток, чередующиеся цветами: белые (светлого цвета), черные (темного цвета). Игра ведется только по черным клеткам.

**Шашки**. Шашки подразделяются на простые и дамки. Дамки маркируются знаком короны. Перед началом игры игрокам предоставляется по 8 простых шашек: одному — белых, другому — черных.

**Стартовая позиция**. Шашки расставляются на темных клетках первых двух горизонтальных рядов с каждой стороны.

**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 только вперед на соседнее поле по диагонали. В случае, когда простая шашка достигает одной из клеток последнего, восьмого (считая от себя) горизонтального ряда, она превращается в дамку и получает новые права. Дамка обозначается символом короны. Дамка, в отличие от простой шашки, ходит на любое из свободных (не занятых другими шашками) полей по диагонали в любом направлении (как вперед, так и назад), 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. Шашка другого игрока в этом случае снимается с поля. Если дамка находится на одной диагонали рядом или на расстоянии от шашки другого игрока, за которой имеется одно или несколько свободных клеток, она должна быть перенесена через эту шашку на первую свободную клетку. Шашка другого игрока в этом случае снимается с поля. Взятие шашки другого игрока является обязательным и производится как вперед, так и назад. Взятие считается одним сыгранным ходом. Взятие своих шашек запрещается. Взятие должно быть четко обозначено и проводится в определенном порядке. Оно считается законченным после снятия шашки соперника с поля. Если в процессе взятия шашкой она вновь оказывается на одной диагонали рядом с другой шашкой игрока, за которой имеется свободная клетка, она должна быть перенесена через вторую, третью и т.д. шашку. Если в процессе взятия дамкой она снова оказывается на одной диагонали рядом или на расстоянии от шашки другого игрока, за которой находится одна или несколько свободных клеток, дамка должна быть обязательно перенесена через вторую, третью и т.д. шашку и занять любую свободную клетку на той же диагонали за последней взятой шашкой. После завершения взятия взятые шашки другого игрока 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 достигает последнего (восьмого от себя) горизонтального ряда, ход игрока(компьютера) заканчивается, даже если у него есть возможность продолжить/начать взятие.

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

В приложении предоставляется возможность зарегистрироваться чтобы играть со своим аккаунтом, сыграть в логическую игру Мак-Йек Поддавки, в приложении реализованы все правила этой игры: порядок хода игроков, ходы шашек, взятие шашки, превращения шашки в дамку, обязательное взятие шашки, последовательное взятие, проверка конца игры и вывод победителя.

**1.2 Условия, необходимые для использования приложения**

Рекомендуется к использованию на Windows 7.

При создании программы использовались встроенная библиотека “enum” и сторонние библиотеки “tkinter 3.6.”, “Pillow 9.5.0”.

Разработка велась в “PyCharm community edition 2023.2.1” на версии языка программирования Python 3.10.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Значимых строк кода 692

Структура данных одна – массив.

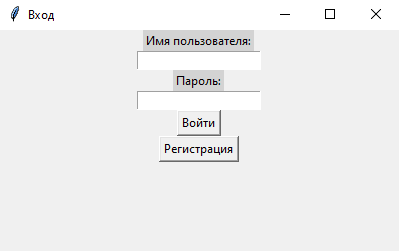
Использованные библиотеки:

“tkinter 3.6” - библиотека для разработки графического интерфейса на языке Python. Методы, включенные в неё, позволяют создавать окна, размещать на них виджеты, настраивать параметры окна и виджетов.

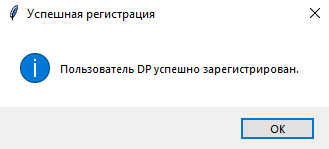
“Pillow 9.5.0” - библиотека языка Python, предоставляет функции для работы с изображениями, такие как изменение размера, поворот, наложение фильтров и многое другое .

“enum” – библиотека, предоставляет поддержку для создания и использования перечислений. Перечисления представляют собой набор уникальных и именованных значений.

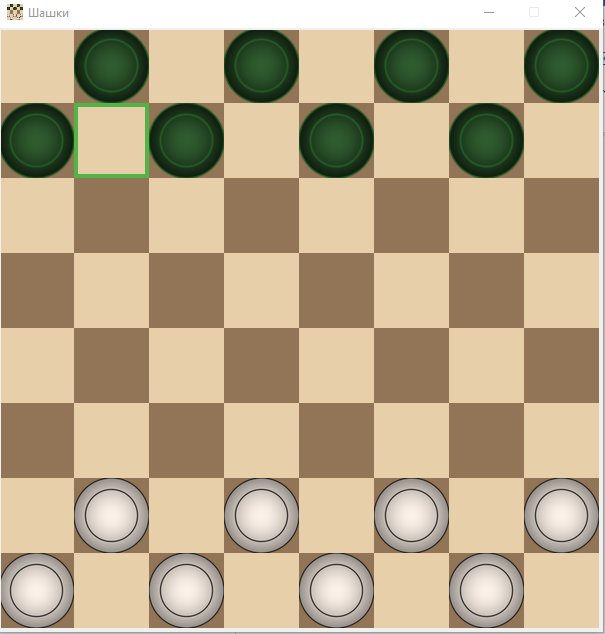
Работа приложения начинается с окна авторизации и регистрации



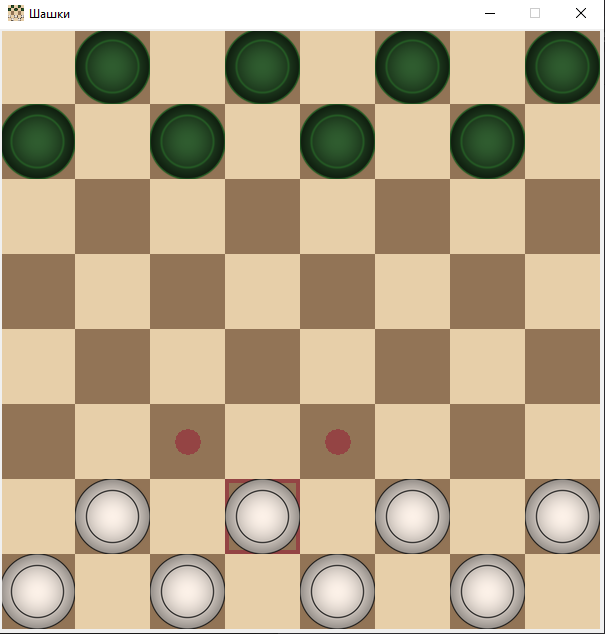
Успешная регистрация



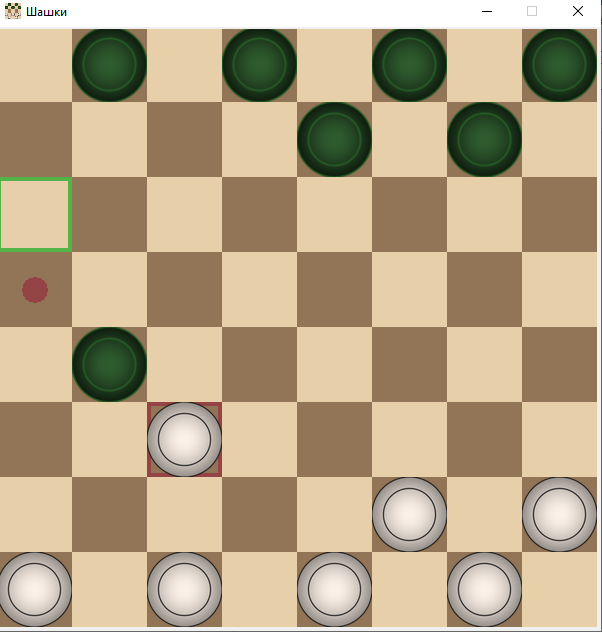
Отрисовка поля

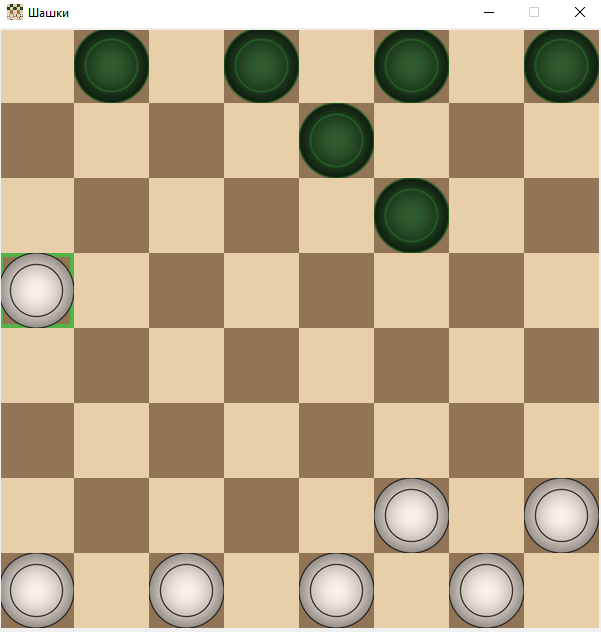


Выбор шашки для хода

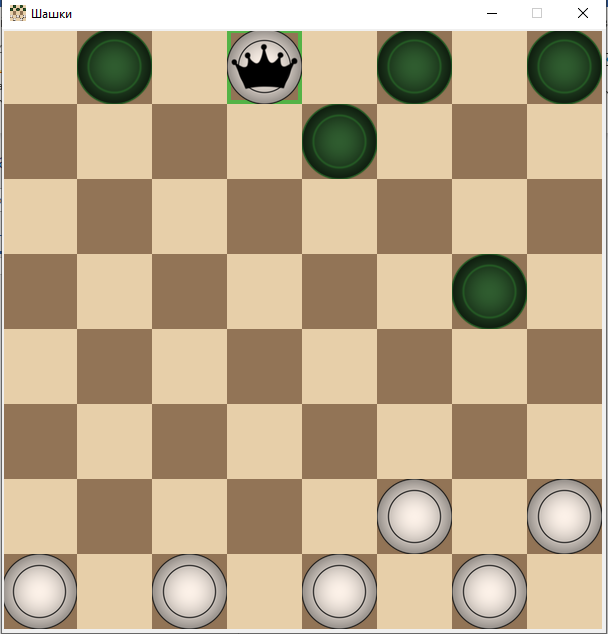


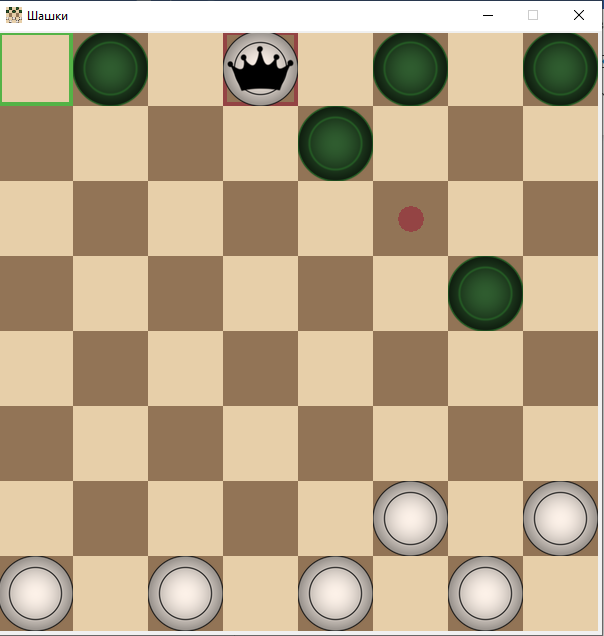
Взятие шашки

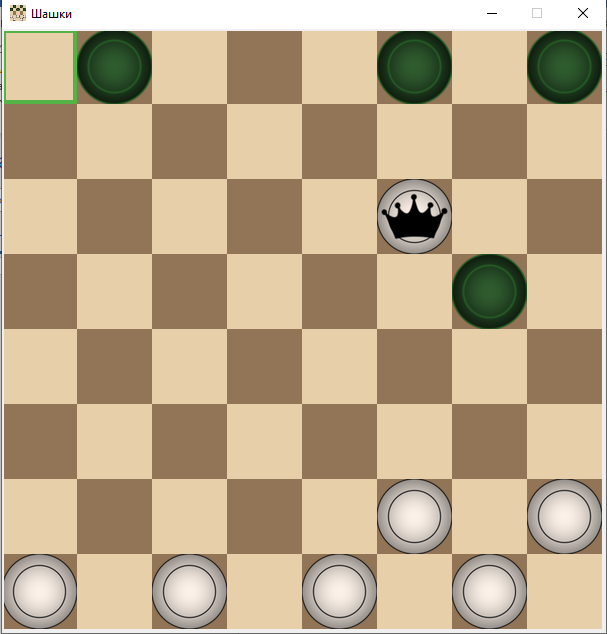


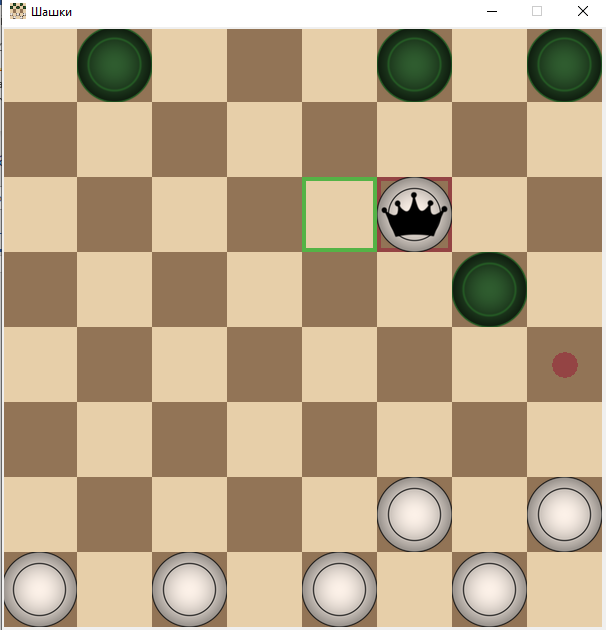


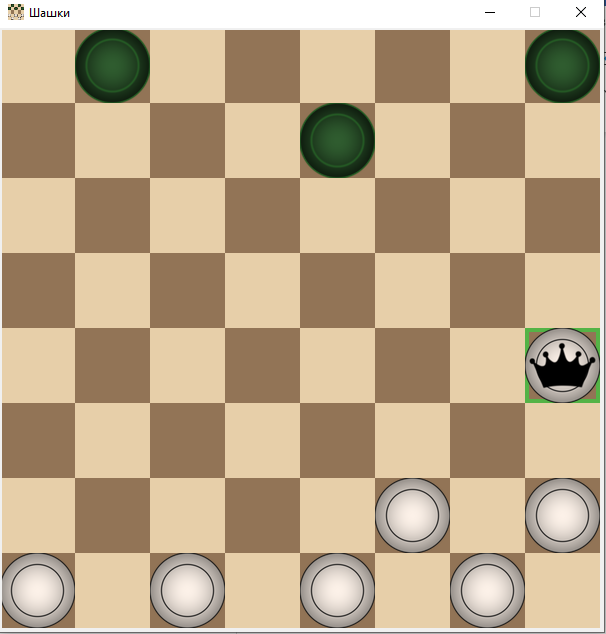
Последовательное взятие



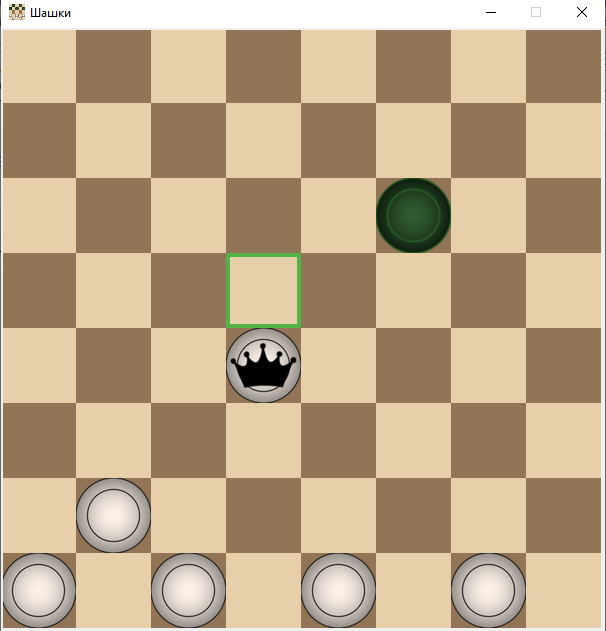


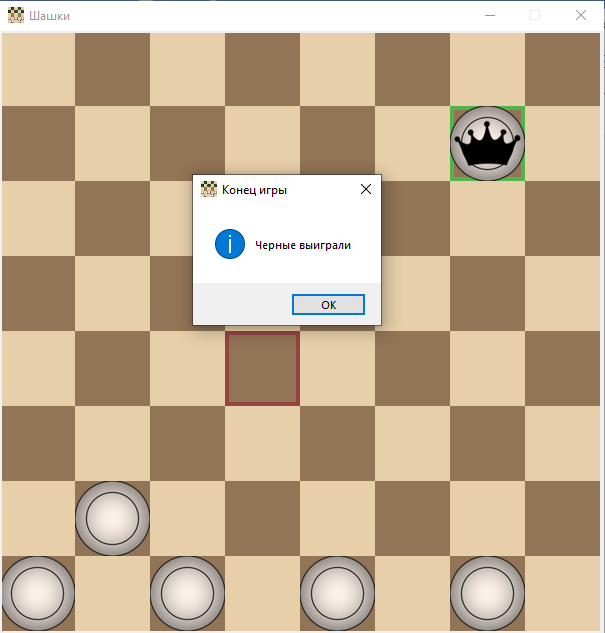


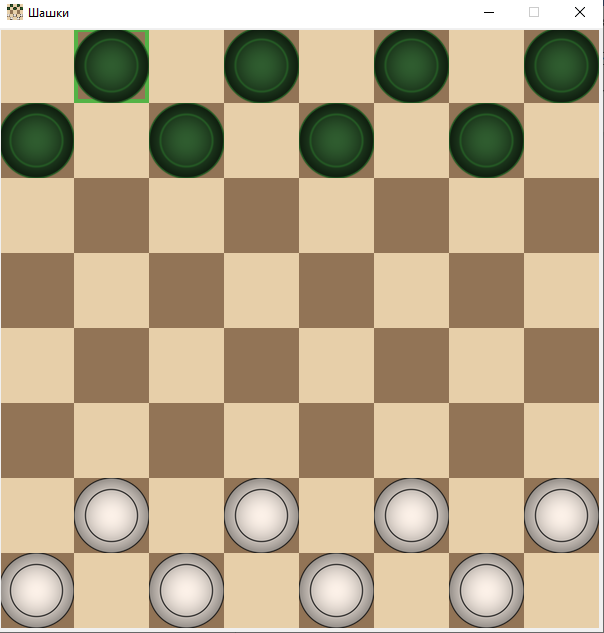




Окончание игры и возможность перезапуска







**2.2 Особенности реализации приложения**

В программе используется структура данных список, эта структура была выбрана ввиду удобной навигации для отрисовки и проверок хода, обязательного взятия, превращения шашки в дамку и последовательного боя.

**3. Обращение к программе**

Класс UserManager содержит в себе следующие методы:

def register\_user – Метод регистрация пользователя, принимает в себя username, password. Проверяет на наличие пользователя в пользователях, иначе добавляет нового пользователя с помощью метода “save\_users” .

def login\_user – Метод входа пользователя, принимает в себя username, password. Проверяет подлинность логина и пароля, иначе выводит ошибку.

def save\_users – метод сохраняющий в файл .json новых пользователей.

def load\_users – метод достающий из файла .json логин и пароль пользователя.

def \_hash\_password – метод, который шифрует кодом sha256 пароль.

def \_verify\_password – метод проверяющий подлинность пароля.

Класс Checker содержит в себе следующие методы:

def type – возвращает тип шашки.

def change\_type – возвращает измененный тип шашки. (смена шашки на дамку)

Класс Point содержит в себе следующие методы:

def x и def y – возвращают координаты точки

Класс Move содержит в себе следующие методы:

def from\_x и def from\_y – возвращает координаты местоположения выбранной шашки.

def to\_x и def to\_y – метод возвращающий координаты возможного хода

def show\_game\_window – метод показывающий окно игры после входа пользователя.

def main – метод, отвечающий за отрисовку окна

def register\_window – метод отрисовки окна регистрации.

Класс Game содержит в себе следующие методы:

def \_\_init\_images – метод инициализирующий файлы .png, отвечающие за шашки.

def \_\_draw – метод, отвечающий за вызов метод отрисовки поля и шашек.

def \_draw\_field\_grid – метод отрисовки сетки поля. Создается холст на котором рисуется поле. Если выбрана какая-то шашка, на нем рисуется точки перемещения.

def \_draw\_checkers – метод отрисовки шашек

def mouse\_down – метод, который обрабатывает нажатия мыши. При нажатии на шашку игрока, выбирает её. При нажатии на ячейку доступную для хода, сделает ход.

def \_\_handle\_move – метод, отвечающий за изменение типа шашки, если она дошла до края; изменение позиции шашки; удаление шашки;

def \_\_handle\_player\_turn – метод, обрабатываещий ход игрока, включая захват и проверку на дополнительные ходы.

def \_\_handle\_enemy\_turn – метод симулирует ход противника, делая оптимальные ходы.

def \_\_check\_for\_game\_over – метод, проверяющий завершена ли игра, и выводит сообщение об этом.

def \_\_predict\_optimal\_moves – метод, подсчитывающий оптимальные ходы для указанной стороны.

def \_\_get\_predicted\_moves\_list – метод, который рекурсивно генерирует список предсказанных ходов.

def \_\_get\_moves\_list – метод, который возвращает список ходов для какой-либо стороны.

def \_\_get\_required\_moves\_list - метод, который возвращает список обязательных ходов для какой-либо стороны.

Обязательные ходы возможны, когда шашка имеет возможность съесть вражескую шашку.

def \_get\_optional\_moves\_list – метод возвращающий список дополнительных ходов для какой-либо стороны. Дополнительные ходы возможны, когда шашка может сделать ход без съедания вражеской шашки.

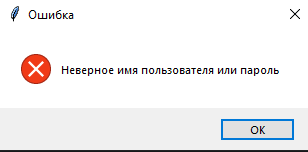
Библиотеки, используемые во время разработки приложения:

Библиотека “tkinter 8.6” – предоставляет возможность создания графических интерфейсов с помощью оконо, различных виджетов и команд. Основные виджеты библиотеки: Label, Button, Entry, Listbox, Canvas, Listbox.

Модуль hashlib предоставляет набор функций для работы с криптографическими хеш-функциями, такими как MD5, SHA-1, SHA-256 и другими.

**4. Сообщения**

Пользователь не найден или введен неверный пароль



**Текст программы:**

**Main.py**

from enum import Enum, auto  
class SideType(Enum):  
 WHITE = auto()  
 BLACK = auto()  
  
 def opposite(side):  
 if (side == SideType.WHITE):  
 return SideType.BLACK  
 elif (side == SideType.BLACK):  
 return SideType.WHITE  
 else: raise ValueError()  
  
class CheckerType(Enum):  
 NONE = auto()  
 WHITE\_REGULAR = auto()  
 BLACK\_REGULAR = auto()  
 WHITE\_QUEEN = auto()  
 BLACK\_QUEEN = auto()  
  
class Checker:  
 def \_\_init\_\_(self, type: CheckerType = CheckerType.NONE):  
 self.\_\_type = type  
   
 def type(self):  
 return self.\_\_type  
  
 def change\_type(self, type: CheckerType):  
 *'''Изменение типа шашки'''* self.\_\_type = type  
  
class Point:  
 def \_\_init\_\_(self, x: int = -1, y: int = -1):  
 self.\_\_x = x  
 self.\_\_y = y  
  
 def x(self):  
 return self.\_\_x  
  
 def y(self):  
 return self.\_\_y  
  
 def \_\_eq\_\_(self, other):  
 if isinstance(other, Point):  
 return (  
 self.x == other.x and  
 self.y == other.y  
 )  
  
 return NotImplemented

**Checkers.py**

from enum import Enum, auto  
class SideType(Enum):  
 WHITE = auto()  
 BLACK = auto()  
  
 def opposite(side):  
 if (side == SideType.WHITE):  
 return SideType.BLACK  
 elif (side == SideType.BLACK):  
 return SideType.WHITE  
 else: raise ValueError()  
  
class CheckerType(Enum):  
 NONE = auto()  
 WHITE\_REGULAR = auto()  
 BLACK\_REGULAR = auto()  
 WHITE\_QUEEN = auto()  
 BLACK\_QUEEN = auto()  
  
class Checker:  
 def \_\_init\_\_(self, type: CheckerType = CheckerType.NONE):  
 self.\_\_type = type  
  
 def type(self):  
 return self.\_\_type  
  
 def change\_type(self, type: CheckerType):  
 *'''Изменение типа шашки'''* self.\_\_type = type  
  
class Point:  
 def \_\_init\_\_(self, x: int = -1, y: int = -1):  
 self.\_\_x = x  
 self.\_\_y = y  
  
 def x(self):  
 return self.\_\_x  
  
 def y(self):  
 return self.\_\_y  
  
 def \_\_eq\_\_(self, other):  
 if isinstance(other, Point):  
 return (  
 self.x == other.x and  
 self.y == other.y  
 )  
  
 return NotImplemented

**Field.py**

from checkers.checker import Checker, CheckerType  
from checkers.constants import WHITE\_CHECKERS, BLACK\_CHECKERS  
  
from functools import reduce  
  
class Field:  
 def \_\_init\_\_(self, x\_size: int, y\_size: int):  
 self.\_\_x\_size = x\_size  
 self.\_\_y\_size = y\_size  
 self.\_\_generate()  
  
 def x\_size(self) -> int:  
 return self.\_\_x\_size  
  
 def y\_size(self) -> int:  
 return self.\_\_y\_size  
  
 def size(self) -> int:  
 return max(self.x\_size, self.y\_size)  
  
 def copy(cls, field\_instance):  
 *'''Создаёт копию поля из образца'''* field\_copy = cls(field\_instance.x\_size, field\_instance.y\_size)  
  
 for y in range(field\_instance.y\_size):  
 for x in range(field\_instance.x\_size):  
 field\_copy.at(x, y).change\_type(field\_instance.type\_at(x, y))  
  
 return field\_copy  
  
 def \_\_generate(self):  
 *'''Генерация поля с шашками'''* self.\_\_checkers = [[Checker() for x in range(self.x\_size)] for y in range(self.y\_size)]  
   
 for y in range(self.y\_size):  
 for x in range(self.x\_size):  
 if ((y + x) % 2):  
 if (y < 2):  
 self.\_\_checkers[y][x].change\_type(CheckerType.BLACK\_REGULAR)  
 elif (y >= self.y\_size - 2):  
 self.\_\_checkers[y][x].change\_type(CheckerType.WHITE\_REGULAR)  
  
 def type\_at(self, x: int, y: int) -> CheckerType:  
 *'''Получение типа шашки на поле по координатам'''* return self.\_\_checkers[y][x].type  
  
 def at(self, x: int, y: int) -> Checker:  
 *'''Получение шашки на поле по координатам'''* return self.\_\_checkers[y][x]  
  
 def is\_within(self, x: int, y: int) -> bool:  
 *'''Определяет лежит ли точка в пределах поля'''* return (0 <= x < self.x\_size and 0 <= y < self.y\_size)  
  
 def white\_checkers\_count(self) -> int:  
 *'''Количество белых шашек на поле'''* return sum(reduce(lambda acc, checker: acc + (checker.type in WHITE\_CHECKERS), checkers, 0) for checkers in self.\_\_checkers)  
  
 def black\_checkers\_count(self) -> int:  
 *'''Количество чёрных шашек на поле'''* return sum(reduce(lambda acc, checker: acc + (checker.type in BLACK\_CHECKERS), checkers, 0) for checkers in self.\_\_checkers)  
  
  
 def white\_score(self) -> int:  
 *'''Счёт белых'''* return sum(reduce(lambda acc, checker: acc + (checker.type == CheckerType.WHITE\_REGULAR) + (checker.type == CheckerType.WHITE\_QUEEN) \* 3, checkers, 0) for checkers in self.\_\_checkers)  
  
 def black\_score(self) -> int:  
 *'''Счёт чёрных'''* return sum(reduce(lambda acc, checker: acc + (checker.type == CheckerType.BLACK\_REGULAR) + (checker.type == CheckerType.BLACK\_QUEEN) \* 3, checkers, 0) for checkers in self.\_\_checkers)

**Game.py**from tkinter import Canvas, Event, messagebox  
from PIL import Image, ImageTk  
from random import choice  
from pathlib import Path  
from math import inf  
  
from checkers.field import Field  
from checkers.constants import \*  
from checkers.checker import CheckerType, SideType  
  
class Move:  
 def \_\_init\_\_(self, from\_x: int = -1, from\_y: int = -1, to\_x: int = -1, to\_y: int = -1):  
 self.\_\_from\_x = from\_x  
 self.\_\_from\_y = from\_y  
 self.\_\_to\_x = to\_x  
 self.\_\_to\_y = to\_y  
  
 def from\_x(self):  
 return self.\_\_from\_x  
   
 def from\_y(self):  
 return self.\_\_from\_y  
  
 def to\_x(self):  
 return self.\_\_to\_x  
  
 def to\_y(self):  
 return self.\_\_to\_y  
  
 def \_\_str\_\_(self):  
 return f'{self.from\_x}-{self.from\_y} -> {self.to\_x}-{self.to\_y}'  
  
 def \_\_repr\_\_(self):  
 return f'{self.from\_x}-{self.from\_y} -> {self.to\_x}-{self.to\_y}'  
  
 def \_\_eq\_\_(self, other):  
 if isinstance(other, Move):  
 return (  
 self.from\_x == other.from\_x and  
 self.from\_y == other.from\_y and  
 self.to\_x == other.to\_x and  
 self.to\_y == other.to\_y  
 )  
  
 return NotImplemented  
  
class Game:  
 def \_\_init\_\_(self, canvas: Canvas, x\_field\_size: int, y\_field\_size: int):  
 self.\_canvas = canvas  
 self.\_field = Field(x\_field\_size, y\_field\_size)  
  
 self.\_\_player\_turn = True  
  
 self.\_\_hovered\_cell = Point()  
 self.\_selected\_cell = Point()  
 self.\_\_animated\_cell = Point()  
  
 self.\_\_init\_images()  
  
 self.\_\_draw()  
  
 def \_\_init\_images(self):  
 *#Инициализация изображений* self.\_\_images = {  
 CheckerType.WHITE\_REGULAR: ImageTk.PhotoImage(Image.open(Path('assets', 'white-regular.png')).resize((cell\_size, cell\_size), Image.ANTIALIAS)),  
 CheckerType.BLACK\_REGULAR: ImageTk.PhotoImage(Image.open(Path('assets', 'black-regular.png')).resize((cell\_size, cell\_size), Image.ANTIALIAS)),  
 CheckerType.WHITE\_QUEEN: ImageTk.PhotoImage(Image.open(Path('assets', 'white-queen.png')).resize((cell\_size, cell\_size), Image.ANTIALIAS)),  
 CheckerType.BLACK\_QUEEN: ImageTk.PhotoImage(Image.open(Path('assets', 'black-queen.png')).resize((cell\_size, cell\_size), Image.ANTIALIAS)),  
 }

def \_\_draw(self):  
 *#Отрисовка сетки поля и шашек* self.\_canvas.delete('all')  
 self.\_draw\_field\_grid()  
 self.\_draw\_checkers()  
  
 def \_draw\_field\_grid(self):  
 *#Отрисовка сетки поля* for y in range(self.\_field.y\_size):  
 for x in range(self.\_field.x\_size):  
 self.\_canvas.create\_rectangle(x \* cell\_size, y \* cell\_size, x \* cell\_size + cell\_size, y \* cell\_size + cell\_size, fill=f\_colors[(y + x) % 2], width=0, tag='boards')  
  
 *# Отрисовка рамок у необхдимых клетоко* if x == self.\_selected\_cell.x and y == self.\_selected\_cell.y:  
 self.\_canvas.create\_rectangle(x \* cell\_size + border\_width // 2, y \* cell\_size + border\_width // 2, x \* cell\_size + cell\_size - border\_width // 2, y \* cell\_size + cell\_size - border\_width // 2, outline=SELECT\_BORDER\_COLOR, width=border\_width, tag='border')  
  
 elif x == self.\_\_hovered\_cell.x and y == self.\_\_hovered\_cell.y:  
 self.\_canvas.create\_rectangle(x \* cell\_size + border\_width // 2, y \* cell\_size + border\_width // 2, x \* cell\_size + cell\_size - border\_width // 2, y \* cell\_size + cell\_size - border\_width // 2, outline=HOVER\_BORDER\_COLOR, width=border\_width, tag='border')  
  
 *# Отрисовка возможных точек перемещения, если есть выбранная ячейка* if self.\_selected\_cell:  
 player\_moves\_list = self.\_\_get\_moves\_list(player\_side)  
 for move in player\_moves\_list:  
 if self.\_selected\_cell.x == move.from\_x and self.\_selected\_cell.y == move.from\_y:  
 self.\_canvas.create\_oval(move.to\_x \* cell\_size + cell\_size / 3, move.to\_y \* cell\_size + cell\_size / 3, move.to\_x \* cell\_size + (cell\_size - cell\_size / 3), move.to\_y \* cell\_size + (cell\_size - cell\_size / 3), fill=POSIBLE\_MOVE\_CIRCLE\_COLOR, width=0, tag='posible\_move\_circle')  
  
 def \_draw\_checkers(self):  
 *#Отрисовка шашек* for y in range(self.\_field.y\_size):  
 for x in range(self.\_field.x\_size):  
 *# Не отрисовывать пустые ячейки* if self.\_field.type\_at(x, y) != CheckerType.NONE:  
 self.\_canvas.create\_image(x \* cell\_size, y \* cell\_size, image=self.\_\_images.get(self.\_field.type\_at(x, y)), anchor='nw', tag='checkers')  
  
 def mouse\_move(self, event: Event):  
 *#Событие перемещения мышки* x, y = event.x // cell\_size, event.y // cell\_size  
 if x != self.\_\_hovered\_cell.x or y != self.\_\_hovered\_cell.y:  
 self.\_\_hovered\_cell = Point(x, y)  
  
 *# Если ход игрока, то перерисовать* if self.\_\_player\_turn:  
 self.\_\_draw()  
  
 def mouse\_down(self, event: Event):  
 *# Событие нажатия мышки* if not self.\_\_player\_turn:  
 return  
 x, y = event.x // cell\_size, event.y // cell\_size  
  
 *# Если точка не внутри поля* if not self.\_field.is\_within(x, y):  
 return  
  
 if player\_side == SideType.WHITE:  
 player\_checkers = WHITE\_CHECKERS  
 elif player\_side == SideType.BLACK:  
 player\_checkers = BLACK\_CHECKERS  
 else:  
 return  
  
 *# Если нажатие по шашке игрока, то выбрать её* if self.\_field.type\_at(x, y) in player\_checkers:  
 self.\_selected\_cell = Point(x, y)  
 self.\_\_draw()  
 elif self.\_\_player\_turn:  
 move = Move(self.\_selected\_cell.x, self.\_selected\_cell.y, x, y)  
 *# Если нажатие по ячейке, на которую можно походить* if move in self.\_\_get\_moves\_list(player\_side):  
 self.\_\_handle\_player\_turn(move)  
 *# Если не ход игрока, то ход противника* if not self.\_\_player\_turn:  
 self.\_\_handle\_enemy\_turn()  
  
 def \_\_handle\_move(self, move: Move, draw: bool = True) -> bool:  
 *# Изменение типа шашки, если она дошла до края* if move.to\_y == 0 and self.\_field.type\_at(move.from\_x, move.from\_y) == CheckerType.WHITE\_REGULAR:  
 self.\_field.at(move.from\_x, move.from\_y).change\_type(CheckerType.WHITE\_QUEEN)  
 elif move.to\_y == self.\_field.y\_size - 1 and self.\_field.type\_at(move.from\_x, move.from\_y) == CheckerType.BLACK\_REGULAR:  
 self.\_field.at(move.from\_x, move.from\_y).change\_type(CheckerType.BLACK\_QUEEN)  
  
 *# Изменение позиции шашки* self.\_field.at(move.to\_x, move.to\_y).change\_type(self.\_field.type\_at(move.from\_x, move.from\_y))  
 self.\_field.at(move.from\_x, move.from\_y).change\_type(CheckerType.NONE)  
 *# Вектора движения* dx = -1 if move.from\_x < move.to\_x else 1  
 dy = -1 if move.from\_y < move.to\_y else 1  
  
 *# Удаление съеденных шашек* has\_captured\_checked = False  
 x, y = move.to\_x, move.to\_y  
 while x != move.from\_x or y != move.from\_y:  
 x += dx  
 y += dy  
 if self.\_field.type\_at(x, y) != CheckerType.NONE:  
 self.\_field.at(x, y).change\_type(CheckerType.NONE)  
 has\_captured\_checked = True  
 if draw: self.\_\_draw()  
  
 return has\_captured\_checked  
  
 def \_\_handle\_player\_turn(self, move: Move):  
 *#Обработка хода игрока* self.\_\_player\_turn = False  
  
 has\_killed\_checker = self.\_\_handle\_move(move)  
 required\_moves\_list = list(filter(lambda required\_move: move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y, self.\_\_get\_required\_moves\_list(player\_side))) *#Запросить лист доступных ходов* if has\_killed\_checker and required\_moves\_list:  
 self.\_\_player\_turn = True  
  
 self.\_selected\_cell = Point()  
  
 def \_\_handle\_enemy\_turn(self): *#* self.\_\_player\_turn = False  
  
 optimal\_moves\_list = self.\_\_predict\_optimal\_moves(SideType.opposite(player\_side))  
  
 for move in optimal\_moves\_list:  
 self.\_\_handle\_move(move)  
  
 self.\_\_player\_turn = True  
 self.\_\_check\_for\_game\_over()  
  
 def \_\_check\_for\_game\_over(self):  
 *# Проверка на конец игры* game\_over = False  
  
 white\_moves\_list = self.\_\_get\_moves\_list(SideType.WHITE)  
 if not white\_moves\_list:  
 answer = messagebox.showinfo('Конец игры', 'Белые выиграли')  
 game\_over = True  
  
 black\_moves\_list = self.\_\_get\_moves\_list(SideType.BLACK)  
 if not black\_moves\_list:  
 answer = messagebox.showinfo('Конец игры', 'Черные выиграли')  
 game\_over = True  
  
 if game\_over:  
 *# Новая игра* self.\_\_init\_\_(self.\_canvas, self.\_field.x\_size, self.\_field.y\_size)  
  
 def \_\_predict\_optimal\_moves(self, side: SideType) -> list[Move]:  
 *'''Предсказать оптимальный ход'''* best\_result = 0  
 optimal\_moves = []  
 predicted\_moves\_list = self.\_\_get\_predicted\_moves\_list(side)  
  
 if predicted\_moves\_list:  
 field\_copy = Field.copy(self.\_field)  
 for moves in predicted\_moves\_list:  
 for move in moves:  
 self.\_\_handle\_move(move, draw=False)  
  
 try:  
 if side == SideType.WHITE:  
 result = self.\_field.black\_score / self.\_field.black\_score  
 elif side == SideType.BLACK:  
 result = self.\_field.white\_score / self.\_field.black\_score  
 except ZeroDivisionError:  
 result = inf  
  
 if result > best\_result:  
 best\_result = result  
 optimal\_moves.clear()  
 optimal\_moves.append(moves)  
 elif result == best\_result:  
 optimal\_moves.append(moves)  
  
 self.\_field = Field.copy(field\_copy)  
  
 optimal\_move = []  
 if optimal\_moves:  
 *# Фильтрация хода* for move in choice(optimal\_moves):  
 if side == SideType.WHITE and self.\_field.type\_at(move.from\_x, move.from\_y) in BLACK\_CHECKERS:  
 break  
 elif side == SideType.BLACK and self.\_field.type\_at(move.from\_x, move.from\_y) in WHITE\_CHECKERS:  
 break  
  
 optimal\_move.append(move)  
  
 return optimal\_move  
  
 def \_\_get\_predicted\_moves\_list(self, side: SideType, current\_prediction\_depth: int = 0, all\_moves\_list: list[Move] = [], current\_moves\_list: list[Move] = [], required\_moves\_list: list[Move] = []) -> list[Move]:  
 *# Получить предсказанные ходы* if current\_moves\_list:  
 all\_moves\_list.append(current\_moves\_list)  
 else:  
 all\_moves\_list.clear()  
  
 if required\_moves\_list:  
 moves\_list = required\_moves\_list  
 else:  
 moves\_list = self.\_\_get\_moves\_list(side)  
  
 if moves\_list and current\_prediction\_depth < MAX\_PREDICTION\_DEPTH:  
 field\_copy = Field.copy(self.\_field)  
 for move in moves\_list:  
 has\_killed\_checker = self.\_\_handle\_move(move, draw=False)  
  
 required\_moves\_list = list(filter(lambda required\_move: move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y, self.\_\_get\_required\_moves\_list(side)))  
  
 *# Если есть ещё ход этой же шашкой* if has\_killed\_checker and required\_moves\_list:  
 self.\_\_get\_predicted\_moves\_list(side, current\_prediction\_depth, all\_moves\_list, current\_moves\_list + [move], required\_moves\_list)  
 else:  
 self.\_\_get\_predicted\_moves\_list(SideType.opposite(side), current\_prediction\_depth + 1, all\_moves\_list, current\_moves\_list + [move])  
  
 self.\_field = Field.copy(field\_copy)  
  
 return all\_moves\_list  
  
 def \_\_get\_moves\_list(self, side: SideType) -> list[Move]:  
 *# Получение списка ходов* moves\_list = self.\_\_get\_required\_moves\_list(side)  
 if not moves\_list:  
 moves\_list = self.\_get\_optional\_moves\_list(side)  
 return moves\_list  
  
 def \_\_get\_required\_moves\_list(self, side: SideType) -> list[Move]:  
 *# Получение списка обязательных ходов* moves\_list = []  
  
 *# Определение типов шашек* if side == SideType.WHITE:  
 friendly\_checkers = WHITE\_CHECKERS  
 enemy\_checkers = BLACK\_CHECKERS  
 elif side == SideType.BLACK:  
 friendly\_checkers = BLACK\_CHECKERS  
 enemy\_checkers = WHITE\_CHECKERS  
 else: return moves\_list  
  
 for y in range(self.\_field.y\_size):  
 for x in range(self.\_field.x\_size):  
  
 *# Для обычной шашки* if self.\_field.type\_at(x, y) == friendly\_checkers[0]:  
 for offset in MOVE\_OFFSETS:  
 if not (self.\_field.is\_within(x + offset.x \* 2, y + offset.y \* 2)): continue  
  
 if self.\_field.type\_at(x + offset.x, y + offset.y) in enemy\_checkers and self.\_field.type\_at(x + offset.x \* 2, y + offset.y \* 2) == CheckerType.NONE:  
 moves\_list.append(Move(x, y, x + offset.x \* 2, y + offset.y \* 2))  
  
 *# Для дамки* elif self.\_field.type\_at(x, y) == friendly\_checkers[1]:  
 for offset in MOVE\_OFFSETS:  
 if not (self.\_field.is\_within(x + offset.x \* 2, y + offset.y \* 2)): continue  
  
 has\_enemy\_checker\_on\_way = False  
  
 for shift in range(1, self.\_field.size):  
 if not (self.\_field.is\_within(x + offset.x \* shift, y + offset.y \* shift)): continue  
  
 if not has\_enemy\_checker\_on\_way:  
 if self.\_field.type\_at(x + offset.x \* shift, y + offset.y \* shift) in enemy\_checkers:  
 has\_enemy\_checker\_on\_way = True  
 continue  
  
 elif (self.\_field.type\_at(x + offset.x \* shift,  
 y + offset.y \* shift) in friendly\_checkers):  
 break  
  
 if has\_enemy\_checker\_on\_way:  
 if (self.\_field.type\_at(x + offset.x \* shift,  
 y + offset.y \* shift) == CheckerType.NONE):  
 moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))  
 else:  
 break  
  
 break  
  
 return moves\_list  
  
 def \_get\_optional\_moves\_list(self, side: SideType) -> list[Move]:  
 *# Определение наилучшего хода* moves\_list = []  
  
 if side == SideType.WHITE:  
 friendly\_checkers = WHITE\_CHECKERS  
 elif side == SideType.BLACK:  
 friendly\_checkers = BLACK\_CHECKERS  
 else: return moves\_list  
 for y in range(self.\_field.y\_size):  
 for x in range(self.\_field.x\_size):  
  
 if self.\_field.type\_at(x, y) == friendly\_checkers[0]:  
 for offset in MOVE\_OFFSETS[:2] if side == SideType.WHITE else MOVE\_OFFSETS[2:]:  
 if not (self.\_field.is\_within(x + offset.x, y + offset.y)): continue  
  
 if self.\_field.type\_at(x + offset.x, y + offset.y) == CheckerType.NONE:  
 moves\_list.append(Move(x, y, x + offset.x, y + offset.y))  
  
 elif self.\_field.type\_at(x, y) == friendly\_checkers[1]:  
 for offset in MOVE\_OFFSETS:  
 if not (self.\_field.is\_within(x + offset.x, y + offset.y)): continue  
  
 for shift in range(1, self.\_field.size):  
 if not (self.\_field.is\_within(x + offset.x \* shift, y + offset.y \* shift)): continue  
  
 if self.\_field.type\_at(x + offset.x \* shift, y + offset.y \* shift) == CheckerType.NONE:  
 moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))  
 else:  
 break  
 return moves\_list