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

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

по дисциплине «Алгоритмы и структуры данных»

# Тема Компьютерная логическая игра «Киммерийские шашки - Поддавки»

# Р.02069337. 22/2366-8

Листов 33

Руководитель разработки:

<должность и ФИО преподавателя>

**« » 2023 г.**

Исполнитель:

студент гр. ИСТбд-21

*Тарпанов Кирилл Вадимович*

« » 2023 г.

2023

Содержание

Аннотация……...…………………………………………………………3

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

Пояснительная записка...……………………………………………...9

Руководство пользователя……………………………….…………...14

Текст программы…..…………………………………………………...24

**Аннотация**

Данный документ представляет собой пояснительную записку на курсовую работу на тему «Компьютерная логическая игра, Киммерийские шашки – Поддавки». Документ содержит следующие разделы: техническое задание, пояснительная записка и руководство программиста, код программы; в нём излагается постановка задачи и описание реализуемой программы, её назначение. Документ может быть использован в качестве инструкции для применения рассматриваемого программного средства.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

**на курсовую работу**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема** «Киммерийские шашки – поддавки»

Р.02069337. 22/2374 22 ТЗ-2

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 5

**Исполнитель**:

студент гр. ИСТбд-22

*Тарпанов Кирилл Вадимович*

« » 2023 г.

**2023**

**Введение**

Курсовая работа представляет собой однопользовательское десктопное приложение, реализующее игру в киммерийские шашки - поддавки.

Краткие правила игры

В киммерийских шашках работают правила русских шашек за следующими исключениями:

* Положение фигур полностью симметрично, за счёт того, что по вертикали на доске 7 линий, а не 8.
* У обоих игроков есть две дамки.
* Остальные шашки, попадая на последнюю противоположную горизонталь, дамками не становятся.

Функциональные возможности

* Графический интерфейс взаимодействия с пользователем
* Регистрация/авторизация пользователя
* Проверка правильности и отрисовка ходов пользователя и компьютера

1. **Основания для разработки**

Основанием для разработки является учебный план направления 09.03.02 «Информационные системы и технологии».

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

**2.1. Функциональное назначение**

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

**2.2 Требования к функциональным характеристикам**

Приложение должно соответствовать следующим правилам игры.

Игра ведётся между двумя соперниками (пользователь-компьютер) шашками разного цвета на прямоугольном поле размером 8х7 клеток. В начальной позиции у каждого игрока 12 шашек, расставленных на белых клетках доски симметрично шашкам противника, причём у каждого игрока по десять простых и две дамки. Дамки размещаются на полях d1, f1 (белые) и d7, f7 (чёрные).

Игроки совершают ходы поочерёдно. Первый ход делает пользователь. В процессе хода игрок может переместить только одну свою шашку. Простая шашка ходит по диагонали вперёд на одну клетку. Дамка ходит по диагонали на любое свободное поле как вперёд, так и назад. Взятие обязательно. Побитые шашки и дамки снимаются только после завершения хода. Простая шашка, находящаяся рядом с шашкой соперника, за которой имеется свободное поле, переносится через эту шашку на это свободное поле. Если есть возможность продолжить взятие других шашек соперника, то это взятие продолжается, пока бьющая шашка не достигнет положения, из которого бой невозможен. Взятие простой шашкой производится как вперёд, так и назад. Дамка бьёт по диагонали, как вперёд, так и назад, и становится на любое свободное поле после побитой шашки. Аналогично, дамка может бить несколько фигур соперника и должна бить до тех пор, пока это возможно. При бое через дамочное поле простая шашка не превращается в дамку. При взятии применяется правило турецкого удара — за один ход шашку противника можно побить только один раз. То есть, если при бое нескольких шашек противника шашка или дамка повторно выходит на уже побитую шашку, то ход останавливается. При нескольких вариантах взятия, например, одну шашку или две, игрок выбирает вариант взятия по своему усмотрению.

Партия считается закончившейся вничью в следующих случаях:

* Если один из участников предлагает ничью, а другой её принимает;
* При невозможности выигрыша ни одного из соперников;
* Если три раза повторяется одна и та же позиция;
* Если участник, имеющий три дамки (и более) против одной дамки противника, за 15 ходов не возьмёт дамку противника;
* Если в позиции, в которой оба соперника имеют дамки, не изменилось соотношение сил (то есть не было взятия, и ни одна простая шашка не стала дамкой) на протяжении: в 2-х и 3-х фигурных окончаниях — 5 ходов, в 4-х и 5-и фигурных окончаниях — 30 ходов, в 6-и и 7-и фигурных окончаниях — 60 ходов;
* Если участник, имея в окончании партии три дамки, две дамки и простую, дамку и две простые, три простые против одинокой дамки, находящейся на большой дороге, своим 5-м ходом не сможет добиться выигранной позиции;
* Если в течение 15 ходов игроки делали ходы только дамками, не передвигая простых шашек и не производя взятия.

Цель игры – лишить себя возможности хода. Выигрывает не тот, кто заберёт или запрёт все шашки противника, а тот, кто сам отдаст все свои шашки или позволит их запереть противнику.

2.2.1 Требования к структуре приложения

Приложение должно быть разработано в виде одного модуля с дополнительными информационными файлами при необходимости.

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 приём, проверка правильности и отрисовка хода пользователя;

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода компьютера;

- информирование пользователя об окончании игры и победителе.

2.2.2 Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Изображения шашек могут храниться в отдельных графических файлах. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде. Пояснительные информационные сообщения для пользователя должны выводиться внизу игрового поля по ходу игры.

**2.3 Требования к надёжности**

Программа должна нормально функционировать при бесперебойной работе ЭВМ. При возникновении сбоя в работе аппаратуры, восстановление нормальной работы программы должно производиться после: перезагрузки операционной системы; запуска исполняемого файла программы; повторного выполнения действий, потерянных до последнего сохранения информации в файл на диске. Уровень надёжности программы должен соответствовать технологии программирования, предусматривающей: инспекцию исходных текстов программы; автономное тестирование модулей (методов) программы; тестирование сопряжении модулей (методов) программы; комплексное тестирование программы.

**2.4 Требования к информационной и программной совместимости**

Операционная система: Windows 10

Используемые библиотеки: tkinter, numpy, os, random

Язык: Python 3.9.1

Среда разработки: Visual Studio Code 1.83

**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

Все файлы проекта должны хранится в специально отведённом репозитории онлайн-сервиса GitHub.

2.6 3 Сроки хранения

Срок хранения – до июля 2026 года.

**3. Требования к программной документации**

Определяются заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

**По дисциплине «Алгоритмы и структуры данных»**

**Тема** Компьютерная логическая игра «Киммерийские шашки - Поддавки»

**Пояснительная записка**

# Р.02069337. 22/2366-8

Листов 5

**Исполнитель**:

студент гр. ИСТбд-22

*Тарпанов Кирилл Вадимович*

« » 2023 г.

2023

**Введение**

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализованной игры. Приводится описание и обоснование выбранного подхода, краткое описание реализованного приложения.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу.

* 1. **Математические методы**

1. Описание состояния игры:

* + Положение шашек на шахматной доске. Необходимо использование двумерного массива размером 8x7, где каждый элемент представляет ячейку доски и содержит информацию о наличии или отсутствии шашки, стоимость шашки.
  + Состояние игры: текущий игрок, наличие взятия на проходе, возможность хода выбранной шашки, отслеживание таких ситуаций, как: ничья.

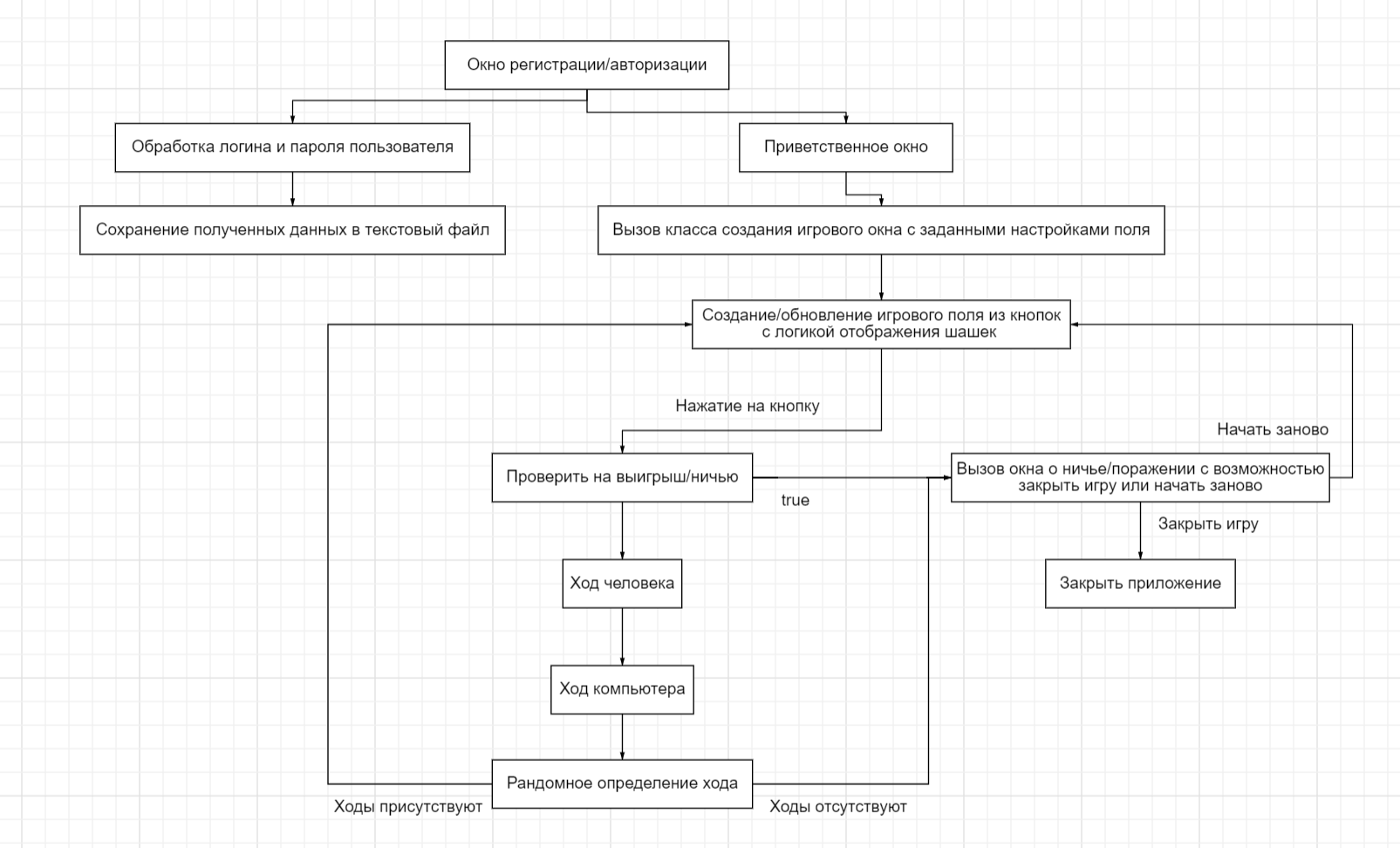
1. Правила игры:
   * Правила перемещения каждой фигуры, включая дамки.
   * Ограничения на действия: нельзя ходить на занятую клетку, нельзя ходить за пределы доски.
   * Обязательные условия окончания игры, такие как выигрыш или ничья.
2. Алгоритмы:
   * Создание списка возможных ходов для каждой фигуры.
   * Проверка возможности хода с учётом правил и ограничений.
   * Определение возможных ходов для противника, оценка выгодности позиции, определение наилучшего хода с помощью алгоритма минимакс с альфа-бета отсечением.

Выбор такой модели обоснован тем, что шашки являются стратегической настольной игрой, в которой игроки принимают решения на основе анализа возможных ходов и прогнозирования дальнейших ходов соперника. Математическая модель позволяет представить все состояния игры, правила и алгоритмы в виде логических и математических выражений, что упрощает анализ и программную реализацию игры.

**1.3 Архитектура и алгоритмы**

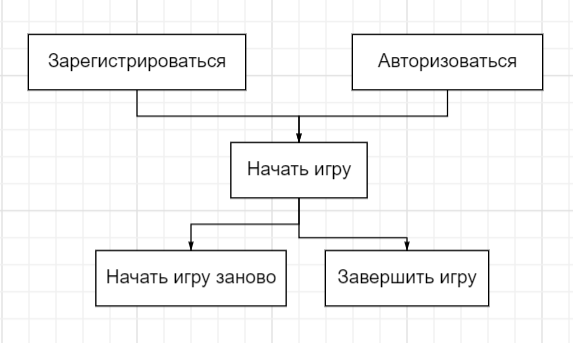
1.3.1. Архитектура

Основные структуры данных (классы и основные методы в классе):



**1.4 Тестирование**

Интеллектуальная карта приложения



1.4.1 Описание отчета о тестировании

Отчёт служит для: определения соответствия исходного продукта техническому заданию, обнаружения дефектов в исходном приложении, исторической фиксации информации о разработке приложения, анализа для дальнейшего улучшения приложения.

1.4.2 Цель тестирования

Целью тестирования является проверка соответствия ПО предъявляемым требованиям, а также выявление возможных багов. По результатам тестирования следует исправление выявленных багов.

1.4.3 Методика тестирования

Тестирование проводилось с использованием следующих методов:

– Статическое тестирование: анализ и проверка кода без его запуска, выявление ошибок в технической документации;

– Ручное тестирование: запуск пользовательских сценариев программы с различными входными данными и проверка корректности полученных результатов;

1.4.4 Проведенные тесты

4.1) Статическое тестирование

Количество обнаруженных и исправленных ошибок в документации: 3.

Количество обнаруженных и исправленных ошибок в программном

коде: 2.

4.2) Ручное тестирование

ТК1 Регистрация/вход с верным вводом данных

Шаги:

1. Запустить главное меню
2. Нажать на кнопку «Регистрация»
3. Ввести верные логин и пароль

Ожидание:

Переход в меню выбора режима игры

Фактический результат:

Переход в меню выбора режима игры

ТК2 Регистрация/вход с неверным вводом данных

Шаги:

1. Запустить главное меню
2. Нажать на кнопку «Регистрация»
3. Ввести неверные логин и пароль

Ожидание:

Вывод сообщения об ошибке

Фактический результат:

Вывод сообщения об ошибке

ТК3 Запуск игры

Шаги:

1. Совершил ход по правилам в игровом окне
2. Совершил ход не по правилам в игровом окне
3. Совершил ход не дождавшись хода соперника(искусственного интеллекта

Ожидание:

Игра продолжается в нормальном режиме. В случае шага 3 искусственный интеллект реагирует на ход оппонента

Фактический результат:

Игра продолжается в нормальном режиме. В случае шага 3 искусственный интеллект реагирует на ход оппонента

1.4.5 Выводы

На основе проведенных тестов сделаны следующие выводы:

– Программа успешно прошла все тесты и работает корректно.

– Обнаружены и исправлены следующие дефекты: Отсутствие хода соперника при «быстром» ходе игрока, отображение всевозможных ходов при выборе игроком шашки.

**2. Источники, использованные при разработке**

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ [Текст] - 3-е изд. - М.: Вильямс, 2004. - 1345 с.

2. "Metanit.com" [Электронный ресурс]. - Режим доступа: https://metanit.com. - Дата доступа: 10.11.2023

3. "Chess Programming Wiki" [Электронный ресурс]. - Режим доступа: https://chessprogramming.org. - Дата доступа: 21.11.2023

4. Dzen.ru [Электронный ресурс]. - Режим доступа: https://dzen.ru/a/YwU5ybOuaELgpTxb. - Дата доступа: 20.11.2023

**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 «Киммерийские шашки - Поддавки»

**Руководство программиста**

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 22/2366-8

Листов 10

Исполнитель:

студент гр. ИСТбд-22

*Тарпанов Кирилл Вадимович*

« » 2023 г.

2023

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение предназначено для развлечения, развития интеллекта, совершенствования способности игры в шашки.

Правила игры:

* Положение фигур полностью симметрично, за счёт того, что по вертикали на доске 7 линий, а не 8.
* У обоих игроков есть две дамки.
* Остальные шашки, попадая на последнюю противоположную горизонталь, дамками не становятся.

Функциональные возможности

* Графический интерфейс взаимодействия с пользователем
* Регистрация/авторизация пользователя
* Проверка правильности и отрисовка ходов пользователя и компьютера

В приложении реализованы: главное меню, окно регистрации, выбор режима игры, модуль, который отрисовывает игру и проверяет на выигрыш/ничью, логика искусственного интеллекта, меню для создания новой партии или выхода из приложения

**1.2 Условия, необходимые для использования приложения**

1. Операционная система: Приложение должно быть доступно для Windows, macOS и Linux.

2. Платформа: Приложение должно быть написано на Python и совместимо с версиями Python 3 и выше.

3. Инструментальная среда: Для использования приложения необходимо установить Python и настроить его на компьютере пользователя.

4. Библиотеки: Приложение должно использовать библиотеки, такие как tkinter для создания графического интерфейса пользователя, а также стандартные библиотеки языка Python версии 3. и выше

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк: 256

Количество структур данных: 3(структура для хранения состояния шахматной доски и соответствующими методами поиска допустимых ходов, проверки доски на выигрыш/ничью, добавления хода и другие, структура для хранения состояния шахматной доски, которая может прочитаться алгоритмом отрисовки интерфейса, структура данных для хранения логина и пароля, а также шифрования пароля)

Количество алгоритмов: 4(алгоритм для отрисовки игрового поля и возможных ходов, алгоритм поиска наилучшего хода для ИИ, алгоритм проверки на мат/пат/блокаду, алгоритм обработки хода игрока)

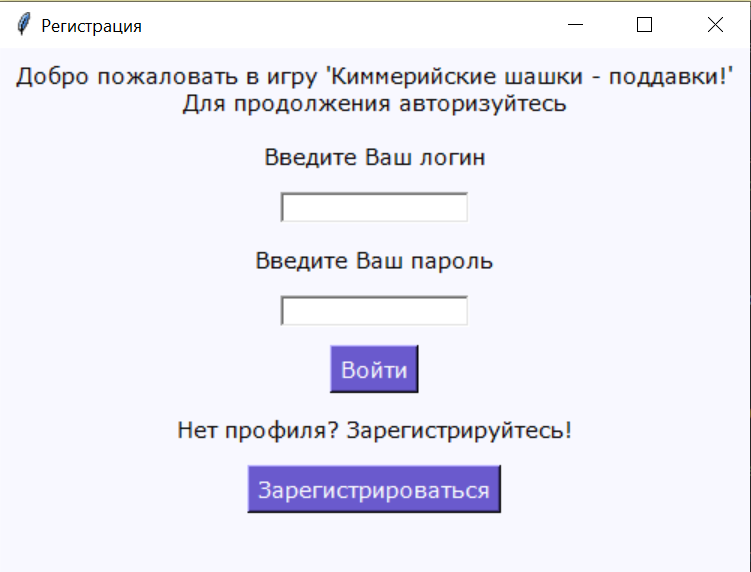
Описание используемых библиотек.

1. Библиотека tkinter: это стандартная библиотека для создания графического интерфейса пользователя (GUI) в Python. Она предоставляет различные виджеты и методы для создания окон, кнопок, полей ввода, меню и других элементов интерфейса. Tkinter обеспечивает простой способ взаимодействия с пользователем и создания приятного пользовательского опыта.

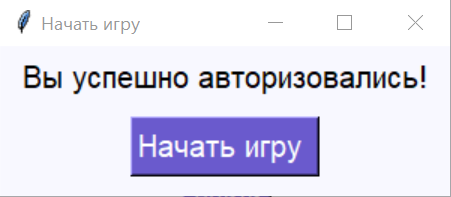
2. Библиотека random: это стандартная библиотека Python, которая предоставляет функции для генерации случайных чисел и выбора случайных элементов из последовательностей (списков, кортежей и т. д.). Она может быть использована для создания случайных данных, случайной генерации игровых элементов, тестирования и других случайных операций. Библиотека random предоставляет различные методы для генерации случайных чисел с разными распределениями и диапазонами значений.

Описание работы приложения:

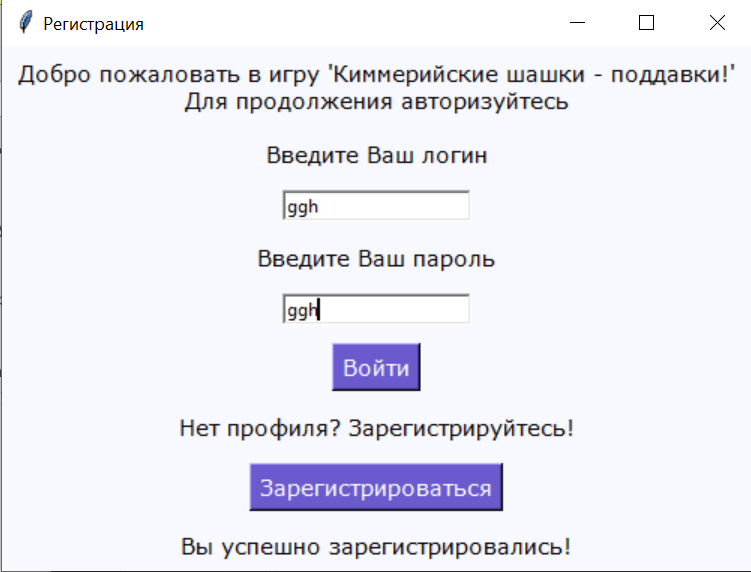
1. Изначально пользователь видит окно авторизации и регистрации, в котором можно зарегистрироваться или войти в существующий профиль.

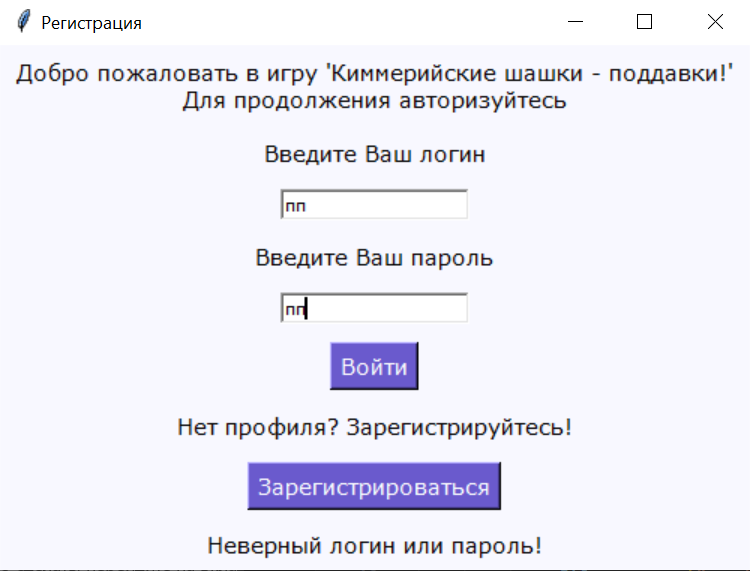


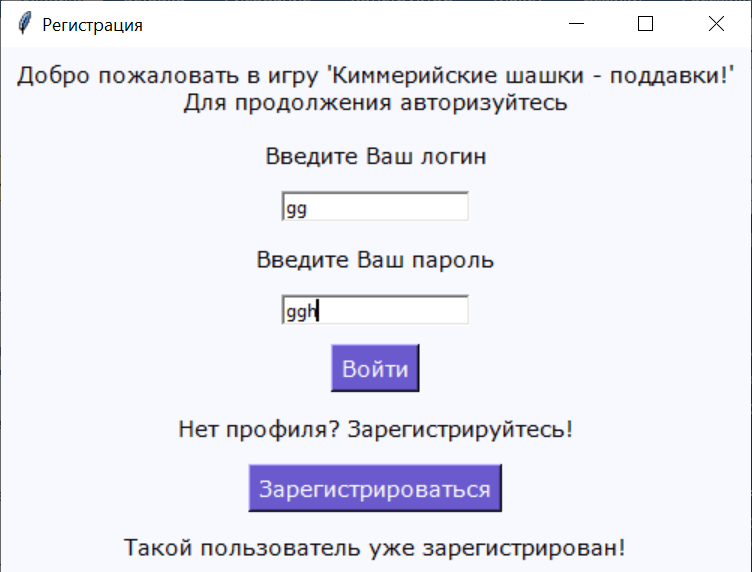
1. При нажатии на кнопку «Зарегистрироваться» происходит регистрация пользователя, при нажатии на кнопку «Войти» открывается приветственное окно.



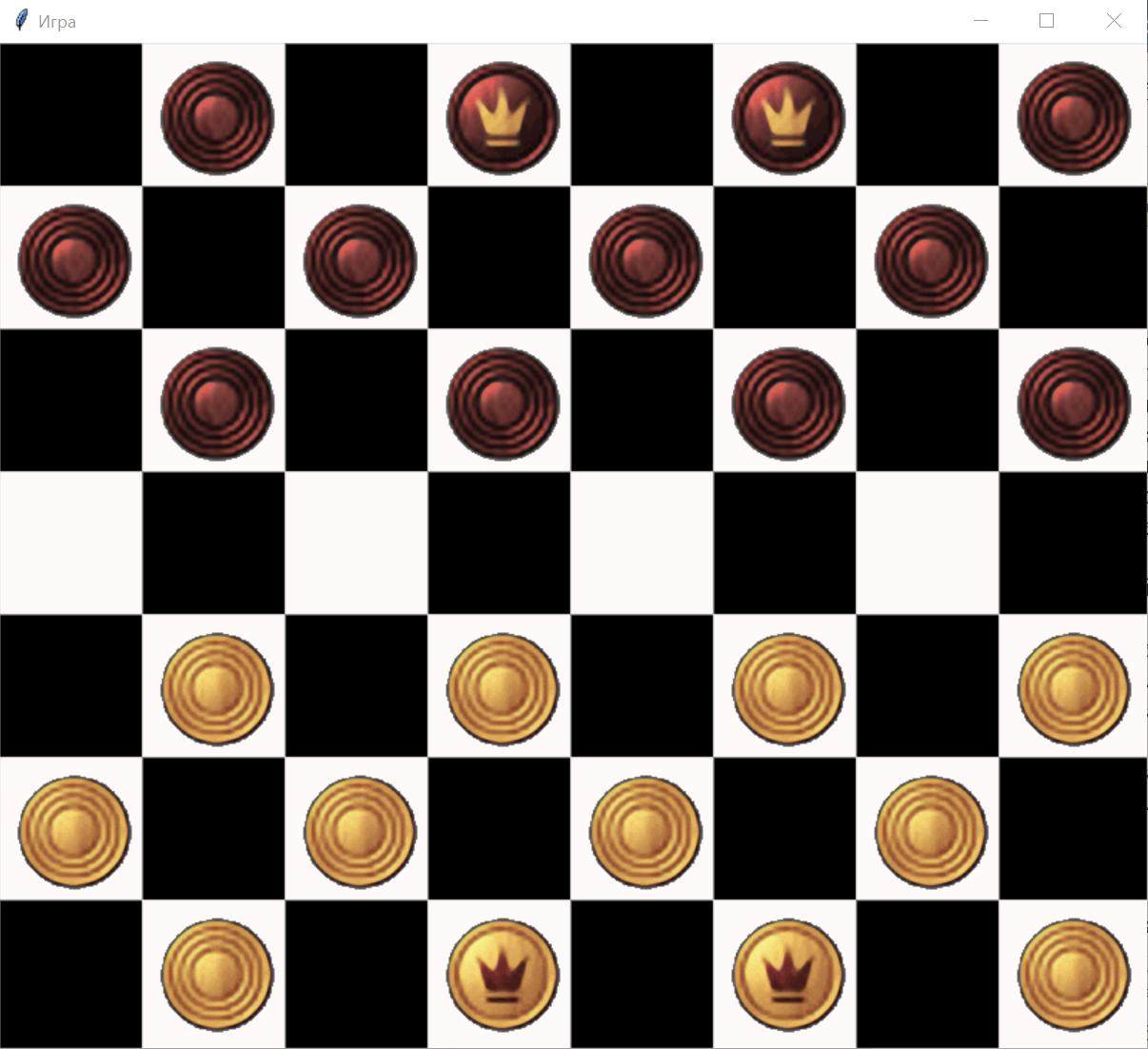
1. В окне регистрации обрабатываются случаи правильного и неправильного ввода данных



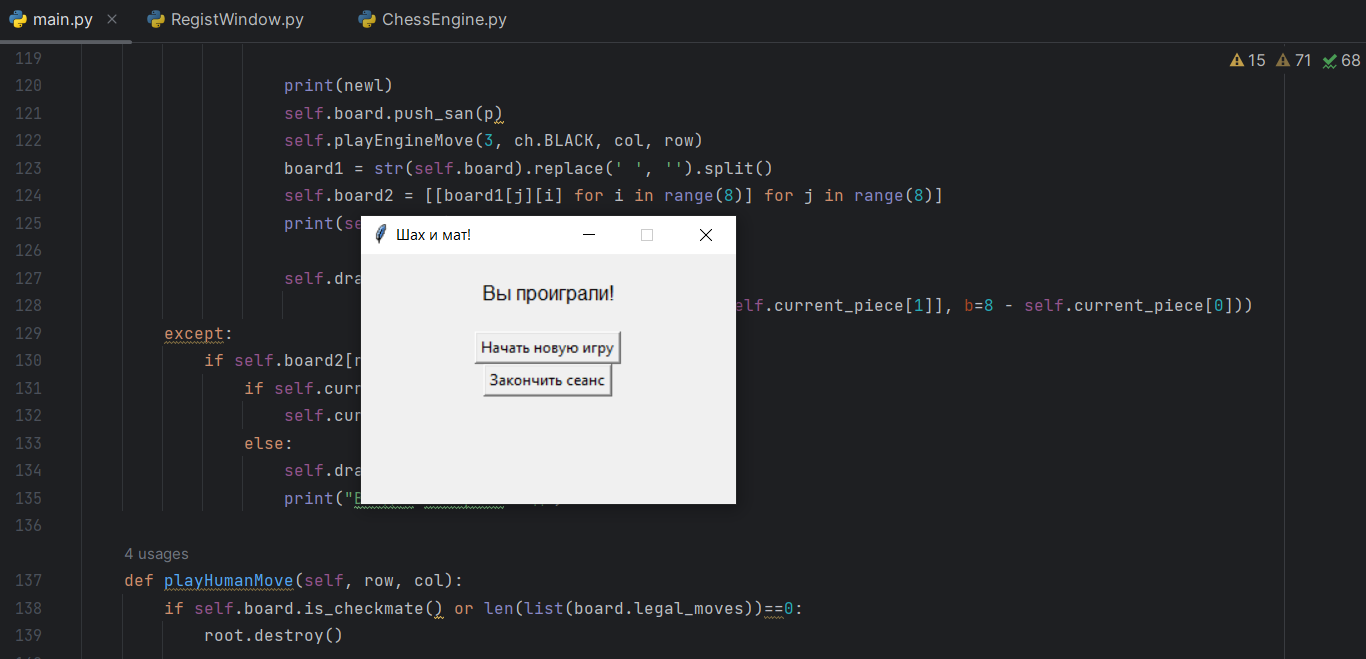




1. Открывается игровое окно



1. После окончания игры (три случая) открывается соответствующее меню, в котором можно закрыть приложение или начать игру заново

**2.2 Особенности реализации приложения**

Использована структура данных из библиотеки chess - Board в ней реализованы все основные методы проверки состояния игрового поля (проверка на мат/пат/блокаду, поиск возможных ходов в зависимости от состояния поля, отслеживание текущего игрока, проверка всевозможных ходов, таких как рокировка, и смена цены пешки при достижении той края доски соперника)

Использован двумерный массив для хранения состояния доски, который может читаться алгоритмом создания графического игрового поля. Реализовано стандартными методами языка Python

Использована структура данных для хранения логина и пароля пользователя. Реализовано с помощью текстового документа. Реализованы методы проверки содержимого документа и добавление новых логинов и паролей.

Возможна ручная реализация структуры из библиотеки chess для меньших затрат ресурсов компьютера. Выбор в сторону библиотечной структуры был сделан в силу меньших затрат времени и более простой реализации. Иначе потребовалось бы более 100 часов на реализацию всех возможных проверок с тем же качеством и скоростью работоспособности.

**3. Обращение к программе**

init(self, board=ch.Board, depth\_lvl=1, root=tk.Tk): Это метод конструктора класса. Он инициализирует различные атрибуты, такие как шахматная доска, уровень глубины и корневой объект Tkinter.

draw\_board(self, root, c\_pc="", a1="normal"): Этот метод используется для рисования шахматной доски в графическом интерфейсе. Он принимает корневой объект Tkinter в качестве параметра и необязательно принимает текущую фигуру и состояние a1. Он создает кнопки для каждого квадрата на шахматной доске и присваивает им соответствующий текст и цвет фона в зависимости от текущего состояния игры.

move(self, row, col): Этот метод вызывается, когда на шахматной доске делается ход. Он обновляет текущее состояние игры, такое как позиции фигур и допустимые ходы, совершает проверку на мат/пат/блокаду.

playEngineMove(self, depth, turn, col, row): Этот метод вызывается шахматным движком для выполнения хода. Он использует алгоритм минимакса с альфа-бета-обрезкой для поиска наилучшего хода. Параметр depth определяет глубину дерева поиска, turn указывает сторону, чья очередь играть, а col и row указывают столбец и строку фигуры для перемещения.

playHumanMove(self, row, col): Этот метод вызывается, когда игрок-человек делает ход. Он обновляет текущую фигуру и вызывает метод draw\_board для обновления графического интерфейса. Он также вызывает метод playEngineMove, чтобы сделать ход с помощью шахматного движка.

Функция `play(pieces, rut)`:

- Принимает два аргумента: `pieces` - строка, содержащая информацию о начальном расположении фигур на шахматной доске в формате FEN, `rut` - объект окна, который нужно закрыть.

- Создает глобальные переменные `root` и `board`.

- Уничтожает окно `rut`.

- Создает новое окно `root` с фиксированным размером и без возможности изменения размера.

- Создает экземпляр класса `Board` из модуля `chess` и устанавливает начальную позицию фигур на доске.

- Создает экземпляр класса `Main` и вызывает его метод `draw\_board` для отрисовки шахматной доски на окне `root`.

Функция `hard()`:

- Создает новое окно `root12` с заголовком "Меню" и размером 300x200.

- Запрещает изменение размера окна `root12`.

- Задает фон окна `root12` черным.

- Создает метку с текстом "Выберите режим игры" и устанавливает цвет шрифта и фона.

- Создает кнопки "Полноценные шахматы" и "Король,ладья,пешка - король,ферзь", каждая из которых вызывает функцию `play` с соответствующими аргументами (FEN-строкой начального расположения фигур на доске и окном `root12`).

- Размещает метку и кнопки на окне `root12`.

Функция `register\_login()`:

- Уничтожает главное окно `root1`.

- Проверяет, существует ли файл "usersааа1.txt". Если да, создает экземпляр класса `RegistrationWindow` с этим файлом в качестве аргумента. Если нет, создает файл "usersааа1.txt" и затем создает экземпляр класса `RegistrationWindow` с этим файлом в качестве аргумента.

- Запускает метод `run()` экземпляра класса `RegistrationWindow` для отображения окна регистрации.

- Выводит в консоль результат метода `accesf()` экземпляра класса `RegistrationWindow`.

- Если результат метода `accesf()` равен `True`, вызывает функцию `hard()`.

Функция `play\_without\_reg()`:

- Уничтожает главное окно `root1`.

- Вызывает функцию `hard()`.

Методы, описанные в коде:

Методы из окна регистрации.

1. Метод `shifr(self, message1)`: Этот метод выполняет шифрование сообщения, принимая на вход строку `message1`. Внутри метода определены переменные `alfavit\_EU` (содержит алфавит в верхнем регистре), `smeshenie` (сдвиг для шифрования), `itog` (переменная для сохранения зашифрованного сообщения). Метод преобразует входную строку в верхний регистр, затем проходит по каждому символу и находит его индекс в `alfavit\_EU`. Затем происходит сдвиг индекса на значение `smeshenie` и добавление соответствующего символа в `itog`. Если символ не найден в алфавите, он остается без изменений. Метод выводит зашифрованное сообщение и возвращает его.

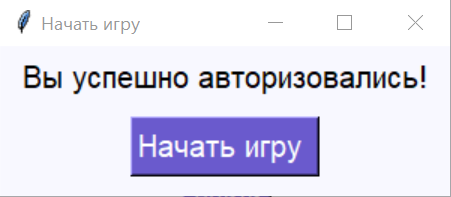
2. Метод `register(self)`: Этот метод выполняет регистрацию пользователя. Он получает логин (`username`) и пароль (`password1`) из соответствующих полей, а затем вызывает метод `shifr` для шифрования пароля. Если логин или пароль не указаны, метод выводит ошибку. Затем метод проверяет, есть ли уже такой пользователь в файле. Если пользователь уже существует, метод выводит ошибку. В противном случае, метод добавляет нового пользователя в файл и выводит сообщение об успешной регистрации.

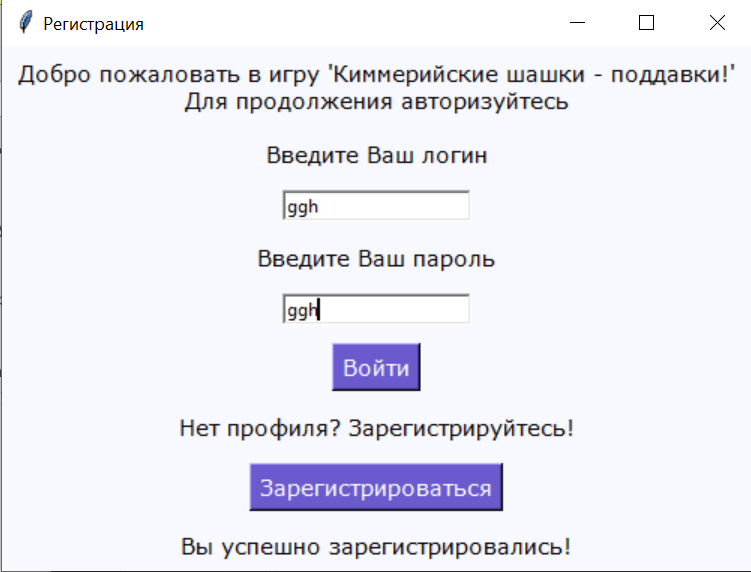
3. Метод `run(self)`: Этот метод запускает основной цикл графического интерфейса пользователя (GUI).

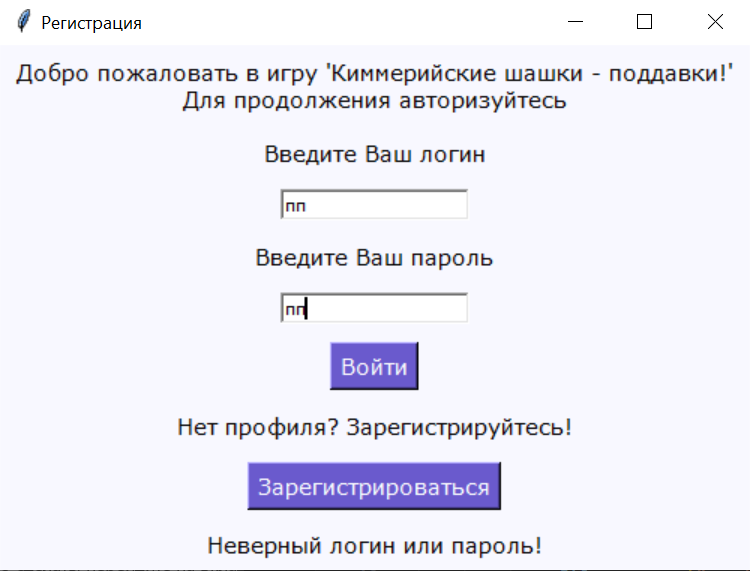
4. Метод `accesf(self)`: Этот метод возвращает значение переменной `acces`, которая указывает на успешность авторизации пользователя.

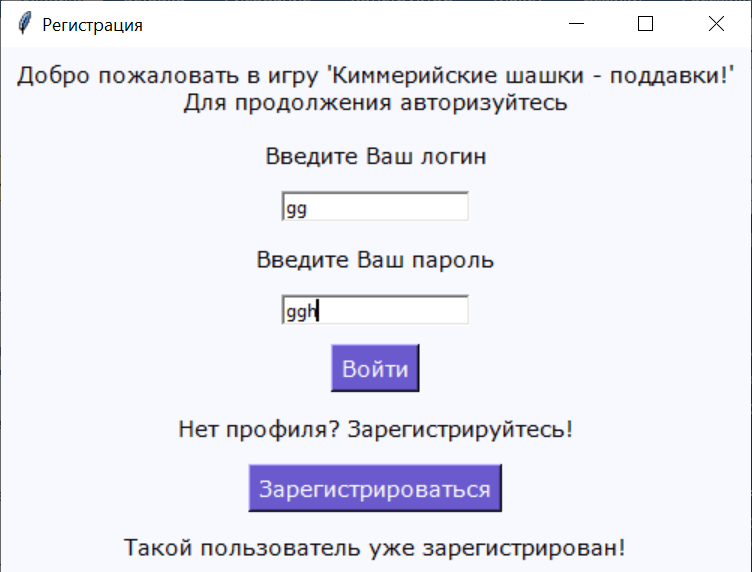
**4. Сообщения**

Сообщения выдаваемые по результатам контроля корректности ввода/вывода.









Полный код программы:

from tkinter import \*

from tkinter import font

import os as os

import numpy as np

import random

class Reg:

    def StartGame(self): #Создаём окно с игровым полем и открываем его

        self.root.destroy() #Закрываем лишние окна

        self.start.destroy()

        Game()

    def Autorize(self):

        font1 = font.Font(family= "Verdana", size=11, weight="normal", slant="roman")

        if os.path.exists("register.txt"):

            file = open("register.txt", "r+") #Открываем файл с данными пользователей

            if not (self.login.get() or self.password.get()):

                self.message["text"] = "Заполните все поля!"

            else:

                if (self.login.get() and self.password.get()) in file.read().split(): #Если введённые данные уже есть

                    self.start = Tk()

                    self.start.title("Начать игру")

                    self.start.geometry("300x100")

                    self.start.configure(background="#F8F8FF")

                    message = Label(master=self.start, anchor=W, bg="#F8F8FF", text="Вы успешно авторизовались!", font=font1)  #Авторизуем пользователя

                    message.pack(padx=6, pady=6)

                    btn = Button(master=self.start, text="Начать игру", anchor=W, bg="#6A5ACD", fg="#FFFFFF", font=font1, command=self.StartGame)

                    btn.pack(padx=6, pady=6)

                    self.start.mainloop()

                else:

                    self.message["text"] = "Неверный логин или пароль!"

        else:  #Если данные введены впервые (файл данных пользователя отсутствует)

            self.message["text"] = "Неверный логин или пароль!"

    def Register(self):

        if self.login.get()=='' or self.password.get()=='':

            self.message["text"] = "Заполните все поля!"

        elif os.path.exists("register.txt"):

            file = open("register.txt", "r+") #Открываем файл с данными пользователей

            if self.login.get() in file.read().split(): #Если введённые данные уже есть

                self.message["text"] = "Такой пользователь уже зарегистрирован!"

            else: #Если данные введены впервые

                file.write(self.login.get() + " " + self.password.get() + "\n") #Регистрируем пользователя

                file.close()

                self.message["text"] = "Вы успешно зарегистрировались!"

        else:

            file = open("register.txt", "w") #Создаём файл с данными пользователей

            file.write(self.login.get() + " " + self.password.get() + "\n") #Регистрируем пользователя

            file.close()

            self.message["text"] = "Вы успешно зарегистрировались!"

    def \_\_init\_\_(self):

        self.root = Tk()     # создаем корневой объект - окно

        self.root.title("Регистрация")     # устанавливаем заголовок окна

        self.root.geometry("500x350")    # устанавливаем размеры окна

        self.root.configure(background="#F8F8FF")

        font1 = font.Font(family= "Verdana", size=11, weight="normal", slant="roman")

        hello = Label(font=font1, anchor=W, background="#F8F8FF", text="Добро пожаловать в игру 'Киммерийские шашки - поддавки!'\nДля продолжения авторизуйтесь") # создаем текстовую метку

        hello.pack(padx=6, pady=6)

        llogin = Label(font=font1, anchor=W, background="#F8F8FF", text="Введите Ваш логин") # создаем текстовую метку

        llogin.pack(padx=6, pady=6) # размещаем метку в окне

        self.login=Entry(bd=2)

        self.login.pack(padx=6, pady=6)

        lpassword = Label(font=font1, anchor=W, background="#F8F8FF", text="Введите Ваш пароль")

        lpassword.pack(padx=6, pady=6)

        self.password=Entry(bd=2)

        self.password.pack(padx=6, pady=6)

        btn1 = Button(text="Войти", bg="#6A5ACD", fg="#FFFFFF", font=font1, command=self.Autorize) #создаём кнопки и устанавливаем внутри окна

        btn1.pack(padx=6, pady=6)

        plsreg = Label(font=font1, anchor=W, background="#F8F8FF", text="Нет профиля? Зарегистрируйтесь!") # создаем текстовую метку

        plsreg.pack(padx=6, pady=6)

        btn2 = Button(text="Зарегистрироваться", bg="#6A5ACD", fg="#FFFFFF", font=font1, command=self.Register) #создаём кнопки и устанавливаем внутри окна

        btn2.pack(padx=6, pady=6)

        self.message = Label(anchor=W, bg="#F8F8FF", font=font1)

        self.message.pack(padx=6, pady=6)

        self.root.mainloop()

class Game:

    class Tile:

            def Create(self, i, j, canvas):

                    def click(event): #клик на клетку с шашкой

                        position = canvas.coords(button\_window) #получаем координаты шашки

                        print(position)

                        able\_white = [key for key, value in self.white\_coords.items() if value == True]

                        print("Доступные ходы")

                        print(able\_white)

                        def check\_step(able):

                            if able == able\_white:

                                print("Белые")

                            else:

                                print("Чёрные")

                            def eliminate(len, step, coord):

                                for i in range(len-1):

                                    if able == able\_white:

                                        if step[i] == -1 and step[i+1] == 0:

                                            self.priority.append(coord)

                                    else:

                                        if step[i] == 1 and step[i+1] == 0:

                                            self.priority.append(coord)

                            for coord in able:

                                x = coord[0]

                                y = coord[1]

                                diag1 = np.diag(self.pos, int(x/100)-int(y/100))

                                diag2 = np.diag(np.fliplr(self.pos), 7-(int(x/100)+int(y/100)))

                                if coord in self.king\_coords:

                                    step11 = diag1[:int(y)]

                                    step12 = diag1[(-1)\*int(y)+1:]

                                    step21 = diag2[:int(y)]

                                    step22 = diag2[(-1)\*int(y)+1:]

                                    step11\_len = len(step11)

                                    step12\_len = len(step12)

                                    step21\_len = len(step21)

                                    step22\_len = len(step22)

                                    if step11\_len >= 3:

                                        eliminate(step11\_len, step11, coord)

                                    elif step12\_len >= 3:

                                        eliminate(step12\_len, step12, coord)

                                    elif step21\_len >= 3:

                                        eliminate(step21\_len, step21, coord)

                                    elif step22\_len >= 3:

                                        eliminate(step22\_len, step22, coord)

                                else:

                                    step11 = diag1[int(y)-2:len(diag1)-int(y)]

                                    step12 = diag1[int(x):len(diag1)-int(y)-2]

                                    step21 = diag2[int(y)-2:len(diag2)-int(y)]

                                    step22 = diag2[int(x):len(diag2)-int(y)-2]

                                    step11\_len = len(step11)

                                    step12\_len = len(step12)

                                    step21\_len = len(step21)

                                    step22\_len = len(step22)

                                    if step11\_len == 2:

                                        eliminate(step11\_len, step11, coord)

                                    elif step12\_len == 2:

                                        eliminate(step12\_len, step12, coord)

                                    elif step21\_len == 2:

                                        eliminate(step21\_len, step21, coord)

                                    elif step22\_len == 2:

                                        eliminate(step22\_len, step22, coord)

                            print("Приоритетные ходы")

                            print(self.priority)

                            if able == able\_white:

                                if len(self.priority) > 0 and not ((position[0], position[1]) in self.priority) or not ((position[0], position[1]) in able):

                                    print("Ошибка!")

                                else:

                                    self.user\_stepped = 1

                                    self.user\_position = position

                                print(self.user\_stepped)

                            else:

                                if len(self.priority) > 0:

                                    random\_black = random.choice(self.priority)

                                else:

                                    random\_black = random.choice(able)

                                return random\_black

                        def computer\_step():

                            print("Ход компьютера")

                            able\_black = [key for key, value in self.black\_coords.items() if value == True]

                            random\_black = check\_step(able\_black)

                            print(random\_black)

                        def user\_step(): #ход игрока обычной шашкой

                            print("Ход фишкой")

                            if self.pos[int(position[1]/100)][int(position[0]/100)] == 0 and position[1] == self.user\_position[1]-100 and (position[0] == self.user\_position[0]+100 or position[0] == self.user\_position[0]-100):

                                self.tiles[int(position[0]), int(position[1])][0]["image"] = self.i1

                                self.pos[int(position[1]/100)][int(position[0]/100)] = 1

                                self.tiles[int(self.user\_position[0]), int(self.user\_position[1])][0]["image"] = ''

                                self.pos[int(self.user\_position[1]/100)][int(self.user\_position[0]/100)] = 0

                                if self.pos[int(position[1]/100)-1][int(position[0]/100)+1] == 0 or self.pos[int(position[1]/100-1)][int(position[0]/100)+1] == 0:

                                    self.white\_coords[position[0], position[1]] = True

                            self.user\_stepped = 0

                            print(self.pos)

                            computer\_step()

                        def user\_king\_step():

                            print("Ход дамкой")

                            if self.pos[int(position[1]/100)][int(position[0]/100)] == 0 and abs(self.user\_position[0] - position[0]) == abs(self.user\_position[1] - position[1]):

                                self.tiles[int(position[0]), int(position[1])][0]["image"] = self.i2

                                self.pos[int(position[1]/100)][int(position[0]/100)] = 1

                                self.tiles[int(self.user\_position[0]), int(self.user\_position[1])][0]["image"] = ''

                                self.pos[int(self.user\_position[1]/100)][int(self.user\_position[0]/100)] = 0

                                self.user\_stepped = 0

                                self.king\_coords.remove(self.user\_position)

                                self.king\_coords.append(position)

                                computer\_step()

                        if self.user\_stepped == 0:

                            if self.pos[int(position[1]/100)][int(position[0]/100)] == 1:

                                check\_step(able\_white)

                        elif self.user\_stepped == 1:

                            if not self.user\_position in self.king\_coords:

                                user\_step()

                            else:

                                user\_king\_step()

                            print(self.user\_stepped)

                    if (i%2==0 and j%2==0) or (i%2!=0 and j%2!=0):

                        self.tile = Button(master=canvas, width=100, height=100, bg="#000000", anchor=NW)

                    else:

                        self.tile = Button(master=canvas, width=100, height=100, bg="#FFFAFA", anchor=NW)

                        if j > 3:

                            if j==6 and (i==5 or i == 3):

                                self.tile["image"] = self.i2

                                self.king\_coords.append([i\*100, j\*100]) #сохраняем координаты данных

                                self.white\_coords[i\*100, j\*100] = True

                            else:

                                self.tile["image"] = self.i1

                            if j > 0 and (i < 7 and self.pos[j-1][i+1] == 0 or i > 0 and self.pos[j-1][i-1] == 0): #Проверяем, может ли шашка совершить ход

                                self.white\_coords[i\*100, j\*100] = True #сохраняем координаты белых шашек

                            else:

                                self.white\_coords[i\*100, j\*100] = False

                        elif j < 3:

                            if j==0 and (i==5 or i == 3):

                                self.tile["image"] = self.i4

                                self.king\_coords.append([i\*100, j\*100]) #сохраняем координаты данных

                                self.black\_coords[i\*100, j\*100] = True

                            else:

                                self.tile["image"] = self.i3

                            if j < 6 and (i < 7 and self.pos[j+1][i+1] == 0 or i > 0 and self.pos[j+1][i-1] == 0): #Проверяем, может ли шашка совершить ход

                                self.black\_coords[i\*100, j\*100] = True #сохраняем координаты чёрных шашек

                            else:

                                self.black\_coords[i\*100, j\*100] = False

                        self.tile.bind('<Button-1>', click)

                        self.tiles[i\*100, j\*100]= [self.tile]

                    button\_window = canvas.create\_window(i\*100, j\*100, anchor=NW, window=self.tile)

    def \_\_init\_\_(self):

            self.pos = np.array([[0, -1, 0, -1, 0, -1, 0, -1],

                                [-1, 0, -1, 0, -1, 0, -1, 0],

                                [0, -1, 0, -1, 0, -1, 0, -1],

                                [0, 0, 0, 0, 0, 0, 0, 0],

                                [0, 1, 0, 1, 0, 1, 0, 1],

                                [1, 0, 1, 0, 1, 0, 1, 0],

                                [0, 1, 0, 1, 0, 1, 0, 1]])

            self.white\_coords = {}

            self.black\_coords = {}

            self.tiles = {}

            self.priority = []

            self.king\_coords = []

            self.user\_stepped = 0

            self.flag = 0

            self.game = Tk()

            self.game.title("Игра")

            self.canvas = Canvas(height=700, width=800)

            self.canvas.pack()

            self.i1=PhotoImage(file="1b.gif")

            self.i2=PhotoImage(file="1bk.gif")

            self.i3=PhotoImage(file="black.gif")

            self.i4=PhotoImage(file="black1.gif")

            for i in range(8):

                for j in range(7):

                    Game.Tile.Create(self, i, j, self.canvas)

            self.game.mainloop()

Reg()