МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
 ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**на курсовую работу**

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

**Тема:** Компьютерная логическая игра «Евразийские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 22/2385 20 ТЗ-1

Листов: 36

Руководитель разработки:

к. т. н., доцент

Шишкин Вадим Викторович

«29» декабря 2023 г.

Исполнитель:

студент гр. ИСТбд-22

Белов Даниил Сергеевич

«29» декабря 2023 г.

2023 г.

Содержание

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

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

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

Руководство программиста……………………………….………….......15

Текст программы…..………………………………………………….......22

**Аннотация**

Данный документ представляет собой пояснительную записку на курсовую работу на тему логическая компьютерная игра «Евразийские шашки». Документ содержит следующие разделы: техническое задание, пояснительная записка и руководство программиста, код программы; в нем излагается постановка задачи и описание реализуемой программы, ее 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

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

по дисциплине «Алгоритмы и структуры данных»

Тема: Компьютерная логическая игра «Евразийские шашки»

Р.02069337. 22/2385-20 ТЗ-2

Листов: 4

**Исполнитель**:

студент гр. ИСТбд-22

Белов Даниил Сергеевич

«29» декабря 2023 г.

2023 г.

**Введение**

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализуемой игры. Даётся краткий свод правил игры, общая характеристика функциональных возможностей, которое должно предоставлять приложение.

Курсовая работа представляет собой однопользовательское десктопное приложение, реализующее игру в Евразийские шашки.

Краткие правила игры

В Евразийских шашках работают правила русских шашек за следующими исключениями:

* Игра ведётся на обычной 64-клеточной доске с традиционной расстановкой шашек (на чёрных полях).
* Шашки ходят только вперёд на свободное соседнее поле по диагонали и превращаются в дамки, достигнув последней горизонтали.
* Взятия происходят иначе чем в русских шашках, то есть бой идёт не по диагонали, а вертикально и горизонтально по чёрным полям.
* У обоих игроков есть две дамки.
* Бить можно как вперёд, так и назад.

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. Функциональное назначение**

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

**2.2 Требования к функциональным характеристикам**

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

Язык: Python 3.9.1

Среда разработки: PyCharm

**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 «Евразийские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

**Пояснительная записка**

Р.02069337. 22/2385-20 ПЗ-01

Листов: 8

**Исполнитель**:

студент гр. ИСТбд-22

Белов Даниил Сергеевич

«29» декабря 2023 г.

2023 г.

**Введение**

Курсовая работа представляет собой десктопное приложение по теме игры “Евразийские шашки”

Краткое описание реализованного приложения:

* Графический интерфейс взаимодействия с пользователем.
* Регистрация/авторизация пользователя.
* Возможность перезапуска игры
* Проверка правильности и отрисовка ходов пользователя и компьютера.
* Оценка и выбор наилучшего хода.
* Определение победителя и возможность переигровки.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

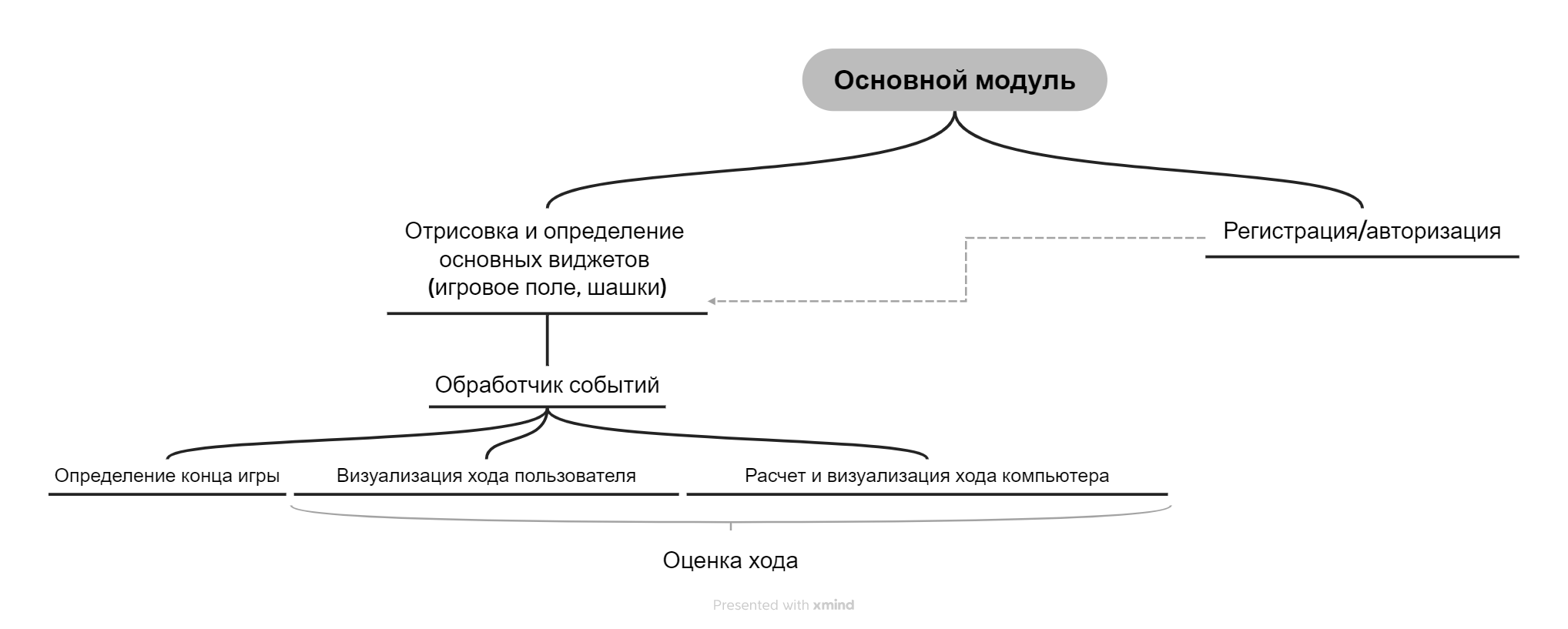
Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании.

**1.2 Математические методы**

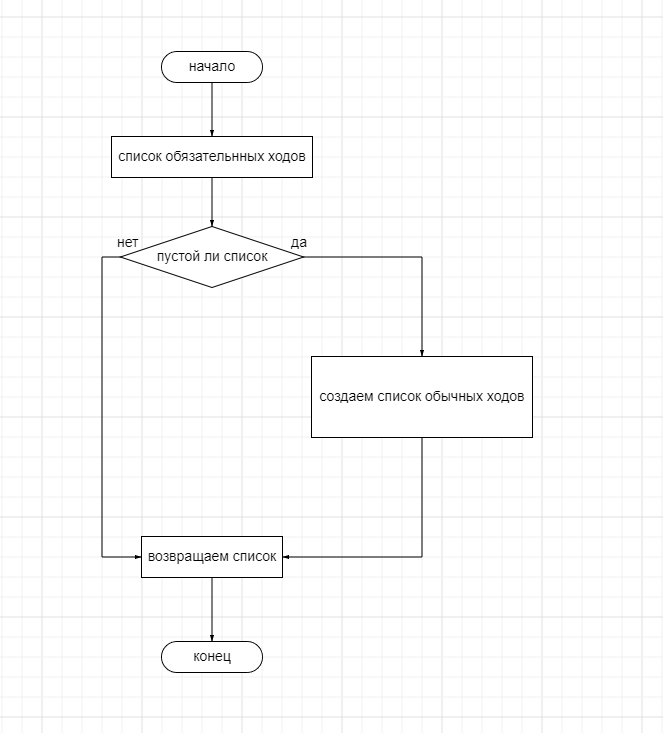
Математических методов нет

**1.3 Архитектура и алгоритмы**

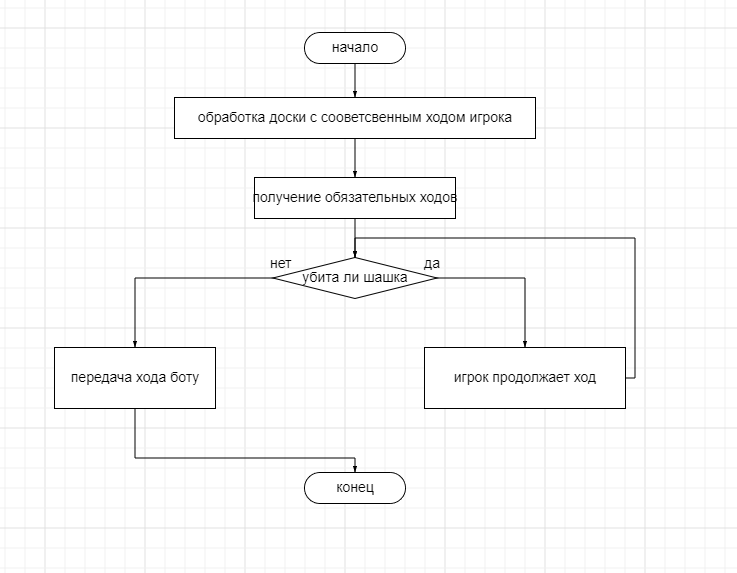
1.3.1. Архитектура



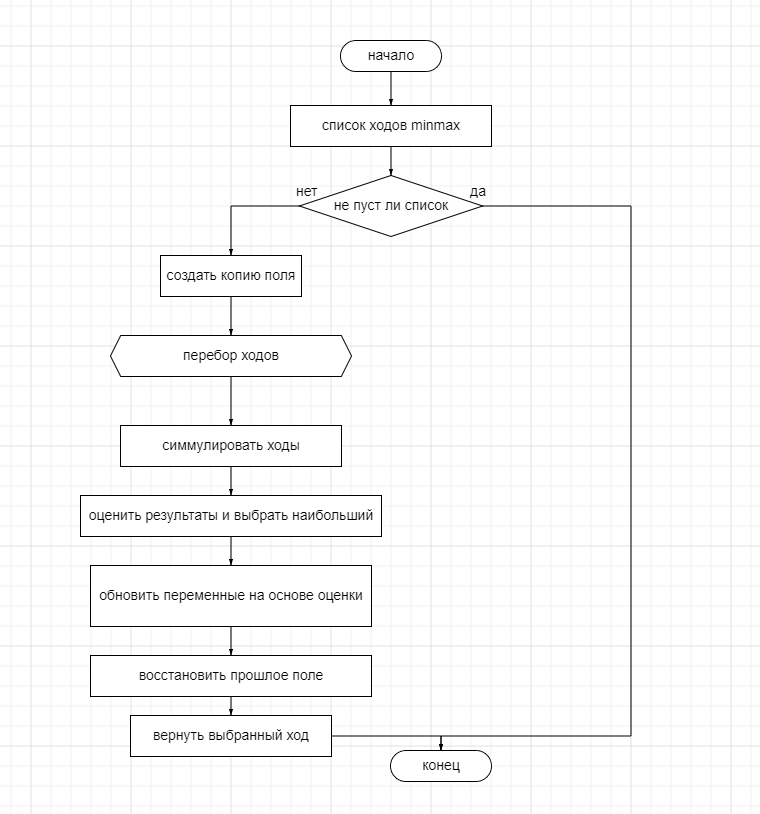
1.3.2. Алгоритм cписка ходов



1.3.3. Алгоритм обработки хода игрока



1.3.4. Алгоритм выбора лучшего хода



1.3.6. Алгоритм входа и регистрации

Данный алгоритм включает в себя два алгоритма и предназначены для осуществления регистрации пользователя и последующего входа в главное окно игры.

**1.4 Тестирование**

1.4.1 Описание отчета о тестировании

В данном отчете представлены результаты тестирования программы на основе функционального тестирования, тестирования удобства пользования, тестирования на отказ и восстановление. Описаны проведенные тесты, их результаты и обнаруженные дефекты.

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

* Функциональное тестирование;
* Тестирование удобства пользования;
* Тестирование на отказ и восстановление.

1.4.4 Проведенные тесты

Отработка авторизации.

Предварительные шаги:

Зарегистрироваться с Имя пользователя: 123 и Пароль 123

Шаги:

1. Запустить приложение.

2. В окне регистрации, в поле «Имя пользователя» ввести 123, а в поле «Пароль» - 123

3. Нажать кнопку «Войти».

Ожидаемый результат: Пользователь начнет игру.

Фактический результат: Пользователь начал новую игру.

Тестирование системы определения победителя

Сценарий 1: У одной из сторон не осталось ходов

Ожидаемый результат: после совершения хода приложение просчитывает все возможные ходы одной из сторон (зависит от очереди хода),

если ходов нет - сообщает о завершении игры и победителе.

Фактический результат: после совершения хода приложение просчитывает все возможные ходы одной из сторон (зависит от очереди хода),

если ходов нет - сообщает о завершении игры и победителе.

1.4.5 Выводы

На основе проведенных тестов сделаны следующие выводы:

– Программа успешно прошла все тесты и работает корректно.

– Рекомендации по дальнейшему улучшению программы: добавление звукового сопровождения, таблицы лидеров.

**2. Источники, использованные при разработке**

1. Введение в Tkinter // Habr URL: https://habr.com/ru/post/133337/ (дата обращения: 26.10.2023).

2. "Крестики-нолики" с алгоритмом "Минимакс" URL: https://www.youtube.com/watch?v=JoJI10CFLzI (дата обращения: 18.11.2023).

3. Tkinter — создание графического интерфейса в Python // python-scripts URL: https://python-scripts.com/tkinter (дата обращения: 02.12.2023).

4. Harvard University. "CS50's Introduction to Artificial Intelligence with Python."//YouTubeURL:https://www.youtube.com/watch?v=WbzNRTTrX0gYouTube (дата обращения: 16.11.2023).

5. GeeksforGeeks. "Минимакс-алгоритм в теории игр - введение. // URL: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/ (дата обращения: 20.11.2023).

6. Checkers-Python // Medium URL: https://medium.com/analytics-vidhya/checkers-python-eff2786b985b (дата обращения: 21.11.2023)

**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 «Евразийские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

**Руководство программиста**

Р.02069337. 22/2385-20 РП-01

Листов: 6

Исполнитель:

студент гр. ИСТбд-22

Белов Даниил Сегреевич

«29» декабря 2023 г.

2023 г.

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Десктопное приложение по теме игры Евразийские шашки.

Краткие правила игры:

В Евразийских шашках работаю все те же правила, что и в русских шашках, но с некоторыми отличиями:

* Игра ведётся на обычной 64-клеточной доске с традиционной расстановкой шашек (на чёрных полях). Есть шашку всегда обязательно, обычная же шашка ест только вперед.
* Шашки ходят только вперёд на свободное соседнее поле по диагонали и превращаются в дамки, достигнув последней горизонтали.

Функциональные возможности:

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

**1.2 Условия, необходимые для использования приложения**

Приложение можно использовать на персональном компьютере. Для использования приложения необходимы:

1. OC Windows 7,8,10,11;
2. Язык Python версии 3.9.
3. Библиотеки: tkinter

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк кода – 623.

Количество алгоритмов – 12.

Библиотеки tkinter, random, os, sys

Порядок работы:

После запуска на экране монитора появится окно авторизации (рис. 1), на котором есть кнопки «Войти» и «Зарегистрироваться».

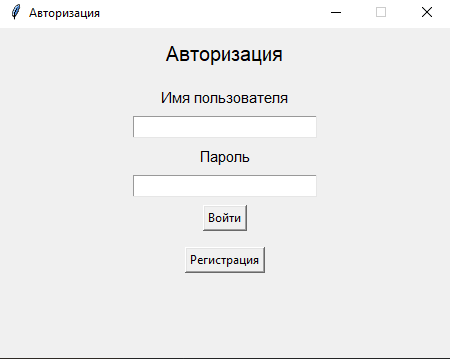


Рис. 1. Окно авторизации.

После введения данных и нажатия на кнопку «Зарегистрироваться» появляется окно с текстом об успешной регистрации аккаунта и просьбой заново войти в свой аккаунт с именем пользователем и паролем.

При успешной авторизации открывается окно игры. (Рис. 2)

Рис. 2. Окно игры.

Далее пользователю следует левой кнопкой мыши выбрать шашку, которой он хочет пойти, и далее указать соседнюю клетку с ней для хода (рис. 3).

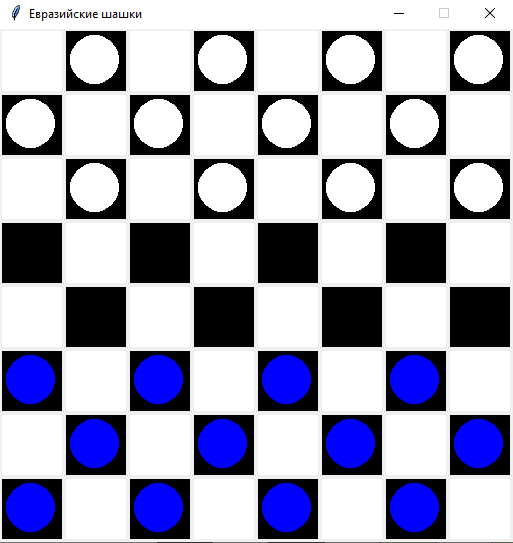


Рис. 2. Окно игры.

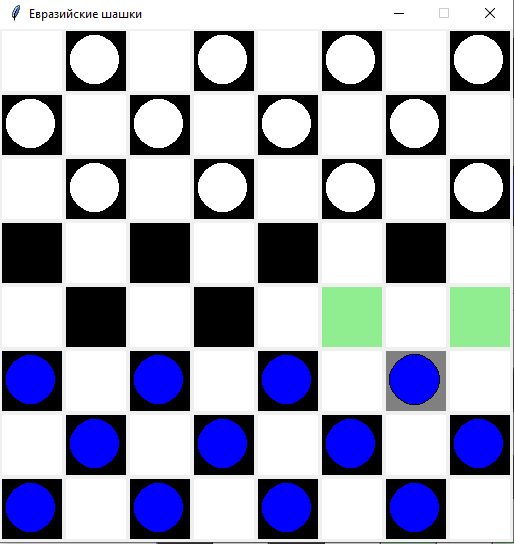


Рис. 3 Подсчет ходов

Рис. 3. Шашка выбрана игроком и ее возможные ходы подсвечены

После хода синими шашками, право хода приходит белым, ход будет делать компьютер. (Рис. 4)

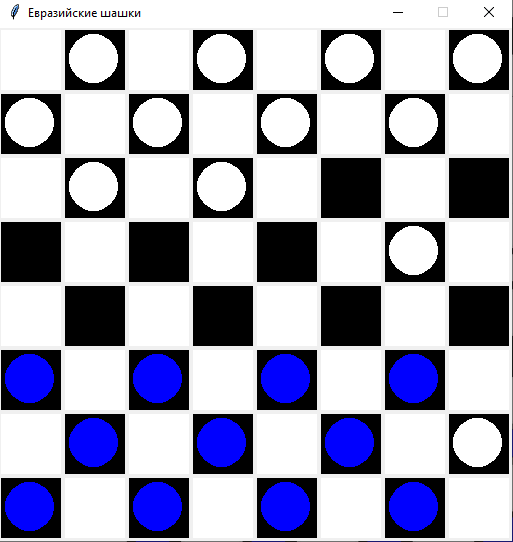


Рис. 4. Шашка компьютера сделала ход.

После того как у кого-то из игроков закончились шашки, либо кто-то заблокирует ходы соперника, то программа выдаст сообщение о победе соответствующей стороны. (Рис. 5)

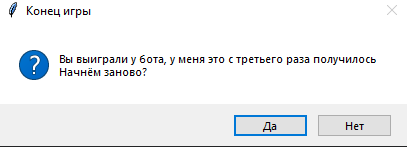


Рис. 5. Игра завершена.

При нажатии на кнопку «Да» пользователю предоставляется возможность сыграть еще партию.

При нажатии на кнопку «Нет» игра закроется.

**2.2 Особенности реализации приложения**

В программе используются массивы, отвечающие за координаты игрового поля , наличие ходов, нахождение шашек.

**3. Обращение к программе**

Алгоритмы и библиотеки.

**4. Сообщения**

При победе программа отображает победителя.

При вводе неправильного логина или пароля всплывает окно с сообщением при авторизации «Неверно введен пароль или логин».

При пустых или пустом поле при авторизации всплывает окно «Неверно введен пароль или логин».

При попытке регистрации с существующим логином всплывет окно с сообщением «Измените имя пользователя или пароль».

При успешной регистрации всплывает окно ”Пользователь добавлен”

При регистрации при пустых или пустом поле всплывает окно «Неверно введен пароль или логин».

**Текст программы:**

Board.py

import random  
import sys  
import tkinter as tk  
from checker import Checker  
from tkinter import messagebox  
  
  
def get\_cell\_color(row, col):  
 return "white" if (row + col) % 2 == 0 else "black"  
  
  
def is\_valid\_cell(row, col):  
 return 0 <= row < 8 and 0 <= col < 8  
  
  
class CheckersBoard:  
 def \_\_init\_\_(self, master):  
 # Шашки  
 self.cells = [[None for \_ in range(8)] for \_ in range(8)]  
 # Канвасы, где мы ррисуем, по сути квадратики  
 self.canvas\_cells = [[None for \_ in range(8)] for \_ in range(8)]  
 # Если мы выбрали шашку  
 self.prev\_highlighted = None  
 # Окно, где у нас всё находится  
 self.master = master  
 # Возможные ходы, подсвечиваются  
 self.highlighted\_moves = []  
 # Конец игры  
 self.isOver = False  
 # Ход игрока  
 self.isPlayerTurn = True  
 # Цвет игрока  
 self.playerColor = "blue"  
 # Цвет автобота  
 self.autoColor = "white"  
 # Ходы, которые нужно сделать  
 self.required\_highlighted = []  
  
 # Перемещаем шашку на доске  
 def place\_checker(self, checker):  
 if is\_valid\_cell(checker.row, checker.col):  
 self.cells[checker.row][checker.col] = checker  
  
 # Создаём шашки, а также рисуем их в канвасах  
 def draw\_starting\_position(self):  
 for row in range(8):  
 for col in range(8):  
 if get\_cell\_color(row, col) == "black" and (row + col) % 2 != 0:  
 if row < 3:  
 checker\_color = "white"  
 elif row > 4:  
 checker\_color = "blue"  
 else:  
 checker\_color = None  
 if checker\_color:  
 checker = Checker(checker\_color, row, col)  
 self.place\_checker(checker)  
 self.canvas\_cells[row][col].create\_oval(5, 5, 55, 55, fill=checker\_color, tags="checker")  
  
 # Создаем канвасы и окрашиваем их  
 def create\_gui(self):  
 for row in range(8):  
 for col in range(8):  
 cell = tk.Canvas(self.master, width=60, height=60, bg=get\_cell\_color(row, col))  
 cell.grid(row=row, column=col)  
 cell.bind("<Button-1>", lambda event, r=row, c=col: self.on\_cell\_click(r, c, self.playerColor))  
 self.canvas\_cells[row][col] = cell  
 self.draw\_starting\_position()  
  
 # Отслеживание нажатий на канвас  
 def on\_cell\_click(self, row, col, color):  
 item\_id = self.canvas\_cells[row][col].find\_withtag("checker")  
 if item\_id:  
 # Полчучаем цвет  
 current\_color = self.canvas\_cells[row][col].itemcget(item\_id, "fill")  
 if current\_color == color:  
 # Когда можно перешагнуть, нельзя выбирать другие ходы  
 if self.required\_highlighted and (row, col) not in self.required\_highlighted:  
 if len(self.find\_possibilities\_to\_attack(row, col)) == 0:  
 print(f"Есть подсвеченные ячейки, попытка выбрать неподсвеченную")  
 return  
  
 # Снимаем подсветку с предыдущей ячейки, если она была подсвечена  
 self.clear\_highlighted\_moves()  
 # Выделяем необходимые  
 self.light\_required\_moves()  
 # Подсвечиваем для хода с этой ячейки  
 self.light\_current\_cell(row, col)  
  
 else:  
 # Берем выделенную  
 if self.prev\_highlighted:  
 prev\_row, prev\_col = self.prev\_highlighted  
 if (row, col) in self.highlighted\_moves:  
 # Получаем цвет ходящего  
 if not self.isPlayerTurn:  
 color = self.playerColor  
 else:  
 color = self.autoColor  
 # Двигаем  
 self.move\_checker(prev\_row, prev\_col, row, col)  
 deleted = False  
 # Если далеко пошли, значит надо проверить, кого убили  
 if abs(prev\_row - row) > 2 or abs(prev\_col - col) > 2:  
 deleted = True  
 self.delete\_checker(prev\_row, prev\_col, row, col, color)  
 # Отрисовка  
 self.canvas\_cells[prev\_row][prev\_col].delete('checker')  
 self.canvas\_cells[prev\_row][prev\_col].delete('crown')  
 self.canvas\_cells[row][col].create\_oval(5, 5, 55, 55, fill=self.cells[row][col].color,  
 tags="checker")  
 if self.cells[row][col].is\_queen:  
 self.canvas\_cells[row][col].create\_oval(10, 10, 50, 50, fill='yellow',  
 tags="crown")  
 self.canvas\_cells[prev\_row][prev\_col].config(bg="gray")  
 if self.is\_game\_over(color):  
 return  
 if not self.isPlayerTurn:  
 # Для автобота  
 if deleted and len(self.find\_possibilities\_to\_attack(row, col)) > 0:  
 # Если мы можем продолжить кушать шашки  
 self.clear\_highlighted\_moves()  
 self.clear\_saved\_moves()  
 self.auto\_turn()  
 else:  
 # Очищаем все и передаём ход  
 self.clear\_highlighted\_moves()  
 self.clear\_saved\_moves()  
 self.find\_necessary\_moves(self.playerColor)  
 self.light\_required\_moves()  
 self.isPlayerTurn = True  
 else:  
 # Для юзеробота  
 if deleted and len(self.find\_possibilities\_to\_attack(row, col)) > 0:  
 # Если можно дальше кушать  
 self.isPlayerTurn = True  
 self.clear\_highlighted\_moves()  
 self.clear\_saved\_moves()  
 self.required\_highlighted.append((row, col))  
 self.light\_required\_moves()  
 else:  
 # Очищаем и передаём  
 self.isPlayerTurn = False  
 self.clear\_highlighted\_moves()  
 self.clear\_saved\_moves()  
 self.auto\_turn()  
  
 # Очистка пометок на доске  
 def light\_current\_cell(self, row, col):  
 # Подсвечиваем текущую ячейку  
 self.canvas\_cells[row][col].config(bg="gray")  
 # Получаем доступные ходы для выбранной шашки  
 selected\_checker = self.cells[row][col]  
 available\_moves = self.get\_available\_moves(self.cells[row][col])  
 # Подсвечиваем клетки для доступных ходов  
 for move\_row, move\_col in available\_moves:  
 self.canvas\_cells[move\_row][move\_col].config(bg="lightgreen")  
 self.highlighted\_moves.append((move\_row, move\_col))  
 # Обновляем предыдущую подсвеченную ячейку  
 self.prev\_highlighted = (row, col)  
  
 # Очистка необходимых ходов  
 def light\_required\_moves(self):  
 for val in self.required\_highlighted:  
 self.canvas\_cells[val[0]][val[1]].config(bg='red')  
  
 # Для выбранной фишки находим подходящие ходы  
 def get\_available\_moves(self, checker):  
 moves = []  
 row, col = checker.row, checker.col  
 req\_moves = self.find\_possibilities\_to\_attack(row, col)  
 if len(req\_moves) > 0:  
 if checker.is\_queen:  
 moves.extend(req\_moves)  
 else:  
 return req\_moves  
 # Шашки могут двигаться по диагонали  
 if checker.is\_queen:  
 # добавляем ходы королевы  
 moves.extend(self.find\_queen\_moves(row, col, -1, 1))  
 moves.extend(self.find\_queen\_moves(row, col, -1, -1))  
 moves.extend(self.find\_queen\_moves(row, col, 1, 1))  
 moves.extend(self.find\_queen\_moves(row, col, 1, -1))  
 else:  
 if checker.color == "white":  
 moves.append((row + 1, col - 1))  
 moves.append((row + 1, col + 1))  
 elif checker.color == "blue":  
 moves.append((row - 1, col - 1))  
 moves.append((row - 1, col + 1))  
 # Теперь уберем из списка те ходы, которые выходят за границы доски  
 moves = [(r, c) for r, c in moves if 0 <= r < 8 and 0 <= c < 8]  
 # Уберем из списка ходы, которые уже заняты другими шашками  
 moves = [(r, c) for r, c in moves if self.cells[r][c] is None]  
 return moves  
  
 # Ходы королевы  
 def find\_queen\_moves(self, row, col, row\_dif, col\_dif):  
 moves = []  
 ch\_row = row + row\_dif  
 ch\_col = col + col\_dif  
 while 8 > ch\_col >= 0 and 8 > ch\_row >= 0:  
 if self.cells[ch\_row][ch\_col] is None and get\_cell\_color(ch\_row, ch\_col) == 'black':  
 moves.append((ch\_row, ch\_col))  
 else:  
 break  
 ch\_row += row\_dif  
 ch\_col += col\_dif  
 return moves  
  
 # Можно ли ходить, для оценки конца игры  
 def is\_there\_moves(self, color):  
 return len(self.get\_possible\_checkers(color)) > 0  
  
 # Смотри количество, которым можем ходить, тоже для оценки конца игры  
 def count\_checkers\_by\_color(self, color):  
 count = 0  
 for inner\_ar in self.cells:  
 for checker in inner\_ar:  
 if checker is not None and checker.color == color:  
 count += 1  
 return count  
  
 # Проверка на конец игры  
 def is\_game\_over(self, color):  
 checkers\_count = self.count\_checkers\_by\_color(color)  
 is\_game\_over = False  
 is\_lose = False  
 if checkers\_count == 0:  
 is\_game\_over = True  
 if color == self.playerColor:  
 is\_lose = True  
  
 if not self.is\_there\_moves(self.autoColor) or not self.is\_there\_moves(self.playerColor):  
 is\_game\_over = True  
 if self.count\_checkers\_by\_color(self.playerColor) < self.count\_checkers\_by\_color(self.autoColor):  
 is\_lose = True  
  
 if is\_game\_over:  
 if is\_lose:  
 message = 'Вы програли у бота, советую не думать, тогда норм получится\nНачнём заново?'  
 else:  
 message = 'Вы выиграли у бота, у меня это с третьего раза получилось\nНачнём заново?'  
 if messagebox.askyesno('Конец игры', message):  
 self.restart()  
 else:  
 sys.exit(1)  
  
 return is\_game\_over  
  
 # Очищаем выделенные клетки  
 def clear\_highlighted\_moves(self):  
 for inner\_list in self.canvas\_cells:  
 for val in inner\_list:  
 if val["background"] != 'white':  
 val.config(bg='black')  
  
 # Убираем почетки  
 def clear\_saved\_moves(self):  
 self.prev\_highlighted = []  
 self.required\_highlighted = []  
 self.highlighted\_moves = []  
  
 # Перемещаем фишки доске, также меняем координаты внутри фишек  
 def move\_checker(self, prev\_row, prev\_col, row, col):  
 checker = self.cells[prev\_row][prev\_col]  
 self.cells[prev\_row][prev\_col] = None  
 checker.move(row, col)  
 self.cells[row][col] = checker  
 if (checker.color == self.playerColor and row == 0) or (checker.color == self.autoColor and row == 7):  
 checker.is\_queen = True  
  
 # Символический метод начала игры  
 def start\_game(self):  
 self.create\_gui()  
  
 # Ищем ходы, которыми нужно ходить за игрока цвета  
 def find\_necessary\_moves(self, color):  
 self.clear\_highlighted\_moves()  
 for row in range(8):  
 for col in range(8):  
 if self.cells[row][col] is None:  
 continue  
 if self.cells[row][col].color != color:  
 continue  
 if self.cells[row][col].is\_queen:  
 continue  
 if len(self.find\_possibilities\_to\_attack(row, col)) > 0:  
 self.required\_highlighted.append((row, col))  
  
 # Берем шашку и оцениваем, можно ли ходить  
 def find\_possibilities\_to\_attack(self, row, col):  
 color = self.cells[row][col].color  
 moves = []  
 if not self.cells[row][col].is\_queen:  
 if (col - 4 >= 0) and (self.cells[row][col - 2] is not None) and (  
 self.cells[row][col - 2].color != color) and (self.cells[row][col - 4] is None):  
 moves.append((row, col - 4))  
 if (col + 4 < 8) and (self.cells[row][col + 2] is not None) and (  
 self.cells[row][col + 2].color != color) and (self.cells[row][col + 4] is None):  
 moves.append((row, col + 4))  
 if (row - 4 >= 0) and (self.cells[row - 2][col] is not None) and (  
 self.cells[row - 2][col].color != color) and (self.cells[row - 4][col] is None):  
 moves.append((row - 4, col))  
 if (row + 4 < 8) and (self.cells[row + 2][col] is not None) and (  
 self.cells[row + 2][col].color != color) and (self.cells[row + 4][col] is None):  
 moves.append((row + 4, col))  
 else:  
 moves.extend(self.find\_queen\_attack(row, col, -1, 0, color))  
 moves.extend(self.find\_queen\_attack(row, col, 1, 0, color))  
 moves.extend(self.find\_queen\_attack(row, col, 0, -1, color))  
 moves.extend(self.find\_queen\_attack(row, col, 0, 1, color))  
 return moves  
  
 def find\_queen\_attack(self, row, col, row\_dif, col\_dif, color):  
 enemy\_has\_met = False  
 moves = []  
 ch\_row = row + row\_dif  
 ch\_col = col + col\_dif  
 while 8 > ch\_col >= 0 and 8 > ch\_row >= 0:  
 if self.cells[ch\_row][ch\_col] is None and get\_cell\_color(ch\_row, ch\_col) == 'black' and enemy\_has\_met:  
 moves.append((ch\_row, ch\_col))  
 if self.cells[ch\_row][ch\_col] is not None:  
 if enemy\_has\_met:  
 break  
 else:  
 if self.cells[ch\_row][ch\_col].color == color:  
 break  
 else:  
 enemy\_has\_met = True  
 ch\_row += row\_dif  
 ch\_col += col\_dif  
 return moves  
  
 def auto\_turn(self):  
 if len(self.required\_highlighted) == 0:  
 self.find\_necessary\_moves(self.autoColor)  
 if len(self.required\_highlighted) > 0:  
 move = random.choice(self.required\_highlighted)  
 prev\_row = move[0]  
 prev\_col = move[1]  
 self.on\_cell\_click(prev\_row, prev\_col, self.autoColor)  
 row, col = random.choice(self.highlighted\_moves)  
 else:  
 checker = random.choice(self.get\_possible\_checkers(self.autoColor))  
 prev\_row, prev\_col = checker.row, checker.col  
 self.on\_cell\_click(prev\_row, prev\_col, self.autoColor)  
 next\_move = random.choice(self.highlighted\_moves)  
 row, col = next\_move[0], next\_move[1]  
 self.on\_cell\_click(row, col, self.autoColor)  
  
 # Проверяем, есть ли у нас шашки, которые могут ходить  
 def get\_possible\_checkers(self, color):  
 checkers = []  
 for row in range(8):  
 for col in range(8):  
 moves = []  
 checker = self.cells[row][col]  
 if checker is None:  
 continue  
 if checker.color != color:  
 continue  
 row, col = checker.row, checker.col  
 # Шашки могут двигаться по диагонали  
 if checker.is\_queen:  
 moves.extend(self.find\_queen\_moves(row, col, -1, 1))  
 moves.extend(self.find\_queen\_moves(row, col, -1, -1))  
 moves.extend(self.find\_queen\_moves(row, col, 1, 1))  
 moves.extend(self.find\_queen\_moves(row, col, 1, -1))  
 else:  
 if self.autoColor == checker.color:  
 moves.append((row + 1, col - 1))  
 moves.append((row + 1, col + 1))  
 elif self.playerColor == checker.color:  
 moves.append((row - 1, col - 1))  
 moves.append((row - 1, col + 1))  
 # Теперь уберем из списка те ходы, которые выходят за границы доски  
 moves = [(r, c) for r, c in moves if 0 <= r < 8 and 0 <= c < 8]  
 # Уберем из списка ходы, которые уже заняты другими шашками  
 moves = [(r, c) for r, c in moves if self.cells[r][c] is None]  
 if len(moves) > 0 or len(self.find\_possibilities\_to\_attack(row, col)) > 0:  
 checkers.append(checker)  
 return checkers  
  
 def delete\_checker(self, prev\_row, prev\_col, row, col, enemy\_color):  
 if prev\_row > row:  
 low\_row = row  
 high\_row = prev\_row  
 else:  
 low\_row = prev\_row  
 high\_row = row  
 if prev\_col > col:  
 low\_col = col  
 high\_col = prev\_col  
 else:  
 low\_col = prev\_col  
 high\_col = col  
  
 for r in range(low\_row, high\_row + 1):  
 for c in range(low\_col, high\_col + 1):  
 checker = self.cells[r][c]  
 if checker is not None and checker.color == enemy\_color:  
 self.cells[r][c] = None  
 self.canvas\_cells[r][c].delete('checker')  
 self.canvas\_cells[r][c].delete('crown')  
  
 def restart(self):  
 self.cells = [[None for \_ in range(8)] for \_ in range(8)]  
 self.canvas\_cells = [[None for \_ in range(8)] for \_ in range(8)]  
 self.prev\_highlighted = None  
 self.highlighted\_moves = []  
 self.isOver = False  
 self.isPlayerTurn = True  
 self.required\_highlighted = []  
 self.start\_game()

Checker.py

class Checker:  
  
 def \_\_init\_\_(self, color, row, col):  
 self.color = color  
 self.row = row  
 self.col = col  
 self.is\_queen = False  
  
 def move(self, new\_row, new\_col):  
 self.row = new\_row  
 self.col = new\_col

Encode.py

class Coder:  
 def \_\_init\_\_(self):  
 self.alfavit\_EU = 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890wertyuioqasdfghjklzxcvbnm-!"#$%&()\*,./:;?@[]^\_`{|}~+<=>'  
 self.smeshenie = 5  
 self.itog = ''  
  
 def encode(self, message):  
 self.itog = ''  
 for i in message:  
 mesto = self.alfavit\_EU.find(i) # Алгоритм для шифрования сообщения на английском  
 new\_mesto = mesto + self.smeshenie  
 if i in self.alfavit\_EU:  
 self.itog += self.alfavit\_EU[new\_mesto % len(self.alfavit\_EU)]  
 else:  
 self.itog += i  
 return self.itog

enter\_window.py

import tkinter as tk  
from pass\_manager import Manager  
from register\_window import RegisterWindow  
from tkinter import messagebox  
  
  
class EnterWindow(tk.Tk):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.manager = Manager()  
 self.response = False  
  
 # Создание и настройка окна  
 self.title('Авторизация')  
 self.geometry('450x330')  
 self.resizable(False, False)  
 font\_header = ('Arial', 15)  
 font\_entry = ('Arial', 12)  
 label\_font = ('Arial', 11)  
 base\_padding = {'padx': 10, 'pady': 8}  
 header\_padding = {'padx': 10, 'pady': 12}  
  
 # обработчик нажатия на клавишу 'Войти'  
 def clicked\_auth():  
 username = username\_entry.get()  
 password = password\_entry.get()  
 if self.manager.get\_user(username, password):  
 self.response = True  
 self.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Неверно введен пароль или логин')  
  
 # обработчик нажатия на клавишу 'Войти'  
 def clicked\_register():  
 register\_window = RegisterWindow()  
  
 main\_label = tk.Label(self, text='Авторизация', font=font\_header, justify=tk.CENTER, \*\*header\_padding)  
 main\_label.pack()  
 # метка для поля ввода имени  
 username\_label = tk.Label(self, text='Имя пользователя', font=label\_font, \*\*base\_padding)  
 username\_label.pack()  
 # поле ввода имени  
 username\_entry = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)  
 username\_entry.pack()  
 # метка для поля ввода пароля  
 password\_label = tk.Label(self, text='Пароль', font=label\_font, \*\*base\_padding)  
 password\_label.pack()  
 # поле ввода пароля  
 password\_entry = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)  
 password\_entry.pack()  
 # кнопка отправки формы  
 send\_btn1 = tk.Button(self, text='Войти', command=clicked\_auth)  
 send\_btn1.pack(\*\*base\_padding)  
 # кнопка отправки регистрации  
 send\_btn2 = tk.Button(self, text='Регистрация', command=clicked\_register)  
 send\_btn2.pack(\*\*base\_padding)

Game.py

import tkinter as tk  
from board import CheckersBoard  
  
  
class Game(tk.Tk):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 # Создание и настройка окна  
 self.title('Евразийские шашки')  
 self.geometry('513x513')  
 self.resizable(False, False)  
 self.board = CheckersBoard(self)  
 self.start\_the\_game()  
  
 def mainloop(self, n=0):  
 super().mainloop()  
 self.start\_the\_game()  
  
 def start\_the\_game(self):  
 self.board.start\_game()

Main.py

import sys  
from enter\_window import EnterWindow  
from game import Game  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 enter\_window = EnterWindow()  
 enter\_window.mainloop()  
 if not enter\_window.response:  
 sys.exit(1)  
 game\_window = Game()  
 game\_window.mainloop()

pass\_manager.py

from encode import Coder  
import os  
  
  
class Manager:  
 def \_\_init\_\_(self):  
 self.coder = Coder()  
 self.file\_name = 'users.txt'  
 self.init\_file()  
  
 def init\_file(self): # Инициализация файла, если этого не сделать програма вылетит м ошибкой, что файла нет  
 *"""Создает файл пользователей"""* if not os.path.exists(self.file\_name):  
 with open('users.txt', 'w'):  
 pass  
  
 def add\_user(self, login: str, password: str) -> bool:  
 *"""Добавляет пользователя в файл"""* with open(self.file\_name, 'r') as f:  
 users = f.read().splitlines() # Считываем всех пользователей из файла  
  
 for user in users:  
 args = user.split(':')  
 if login == args[  
 0]: # Если логин уже есть, парль не проверяем, шанс взлома увеличится(кто-то мб узнает пароль)  
 return False # Тут можно написать что угодно, будь то HTML статус(409 - conflict), либо просто фразу ошибки  
  
 result = self.coder.encode(password)  
 with open('users.txt', 'a') as f:  
 f.write(f'{login}:{result}\n') # Добавляем нового пользователя  
 return True  
  
 def get\_user(self, login: str, password: str) -> bool:  
 *"""Проверяет логин и пароль пользователя"""* with open('users.txt', 'r') as f:  
 users = f.read().splitlines() # Считываем всех пользователей из файла  
  
 result = self.coder.encode(password)  
 for user in users:  
 args = user.split(':')  
 if login == args[0] and result == args[1]: # Если пользователь с таким логином и паролем существует  
 return True  
 return False

Register\_window.py

import tkinter as tk  
from pass\_manager import Manager  
from tkinter import messagebox  
  
  
class RegisterWindow(tk.Tk):  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.manager = Manager()  
  
 # Создание и настройка окна  
 self.title('Регистрация')  
 self.geometry('450x330')  
 self.resizable(False, False)  
 font\_header = ('Arial', 15)  
 font\_entry = ('Arial', 12)  
 label\_font = ('Arial', 11)  
 base\_padding = {'padx': 10, 'pady': 8}  
 header\_padding = {'padx': 10, 'pady': 12}  
  
 # обработчик нажатия на клавишу 'Регистрация'  
 def clicked\_register():  
 password1 = password\_entry\_1.get()  
 password2 = password\_entry\_2.get()  
 if len(password1) == 0:  
 return  
 if password2 != password1:  
 messagebox.showerror('Ошибка', 'Пароли не совпадают')  
 return  
 username = username\_entry.get()  
 if self.manager.add\_user(username, password1):  
 messagebox.showinfo('Успешно', 'Пользователь добавлен')  
 self.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Измените имя пользователя или пароль')  
  
 main\_label = tk.Label(self, text='Регистрация', font=font\_header, justify=tk.CENTER, \*\*header\_padding)  
 main\_label.pack()  
 # метка для поля ввода имени  
 username\_label = tk.Label(self, text='Имя пользователя', font=label\_font, \*\*base\_padding)  
 username\_label.pack()  
 # поле ввода имени  
 username\_entry = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)  
 username\_entry.pack()  
 # метка для поля ввода пароля  
 password\_label\_1 = tk.Label(self, text='Пароль', font=label\_font, \*\*base\_padding)  
 password\_label\_1.pack()  
 # поле ввода пароля  
 password\_entry\_1 = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)  
 password\_entry\_1.pack()  
 # метка для поля ввода пароля  
 password\_label\_2 = tk.Label(self, text='Повторите пароль', font=label\_font, \*\*base\_padding)  
 password\_label\_2.pack()  
 # поле ввода пароля  
 password\_entry\_2 = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)  
 password\_entry\_2.pack()  
 # кнопка отправки регистрации  
 send\_btn = tk.Button(self, text='Зарегистрироваться', command=clicked\_register)  
 send\_btn.pack(\*\*base\_padding)  
 # запускаем главный цикл окна  
 #self.window.mainloop()