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 «Киммерийские шашки»

Р.02069337. 22/2385 20 ТЗ-1

Листов: 36

Руководитель разработки:

к. т. н., доцент

Шишкин Вадим Викторович

«15» февраля 2024 г.

Исполнитель:

студент гр. ИСТбд-22

Бутузова Дарья Евгеньевна

«15» февраля 2024 г.

2023 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Содержание

Аннотация

Данный документ представляет собой пояснительную записку на курсовую работу на тему логическая компьютерная игра «Киммерийские 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

по дисциплине «Алгоритмы и структуры данных»

Тема: Компьютерная логическая игра «Киммерийские шашки»

Р.02069337. 22/2385-20 ТЗ-2

Листов: 4

Исполнитель:
студент гр. ИСТбд-22

Бутузова Дарья Евгеньевна
«15» февраля 2024 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 Киммерийские шашки.

Краткие правила игры

В Киммерийских шашках работают правила русских шашек за следующими исключениями:

- Игра ведётся на 56-клеточной доске с расстановкой шашек на белых полях.
- Шашки ходят только вперёд на свободное соседнее поле по диагонали, но не превращаются в дамки, достигнув последней горизонтали.
- У обоих игроков есть две дамки с начала игры.
- 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 Киммерийские 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 «Киммерийские шашки»

Пояснительная записка

Р.02069337. 22/2385-20 ПЗ-01

Листов: 8

Исполнитель:
студент гр. ИСТбд-22
Бутузова Дарья Евгеньевна
«15» февраля 2024 г.

2023 г.

Введение

Курсовая работа представляет собой десктопное приложение по теме игры “Киммерийские 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

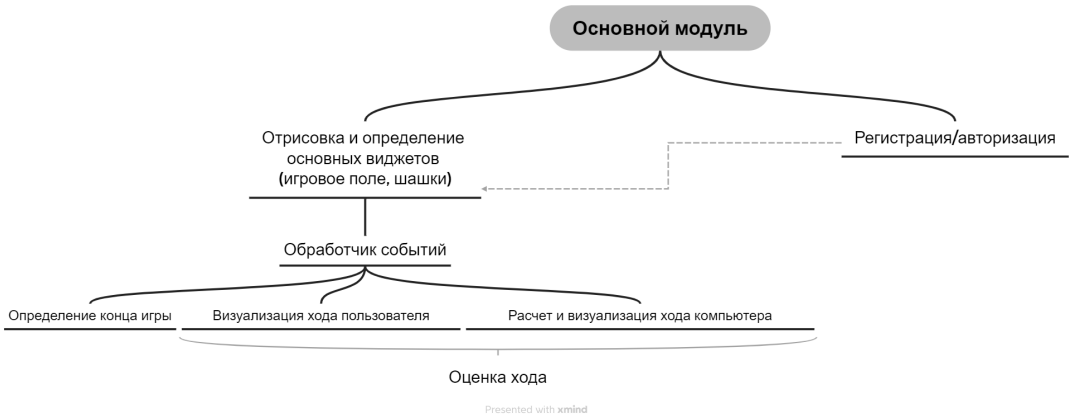
Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании.

1.2 Математические методы

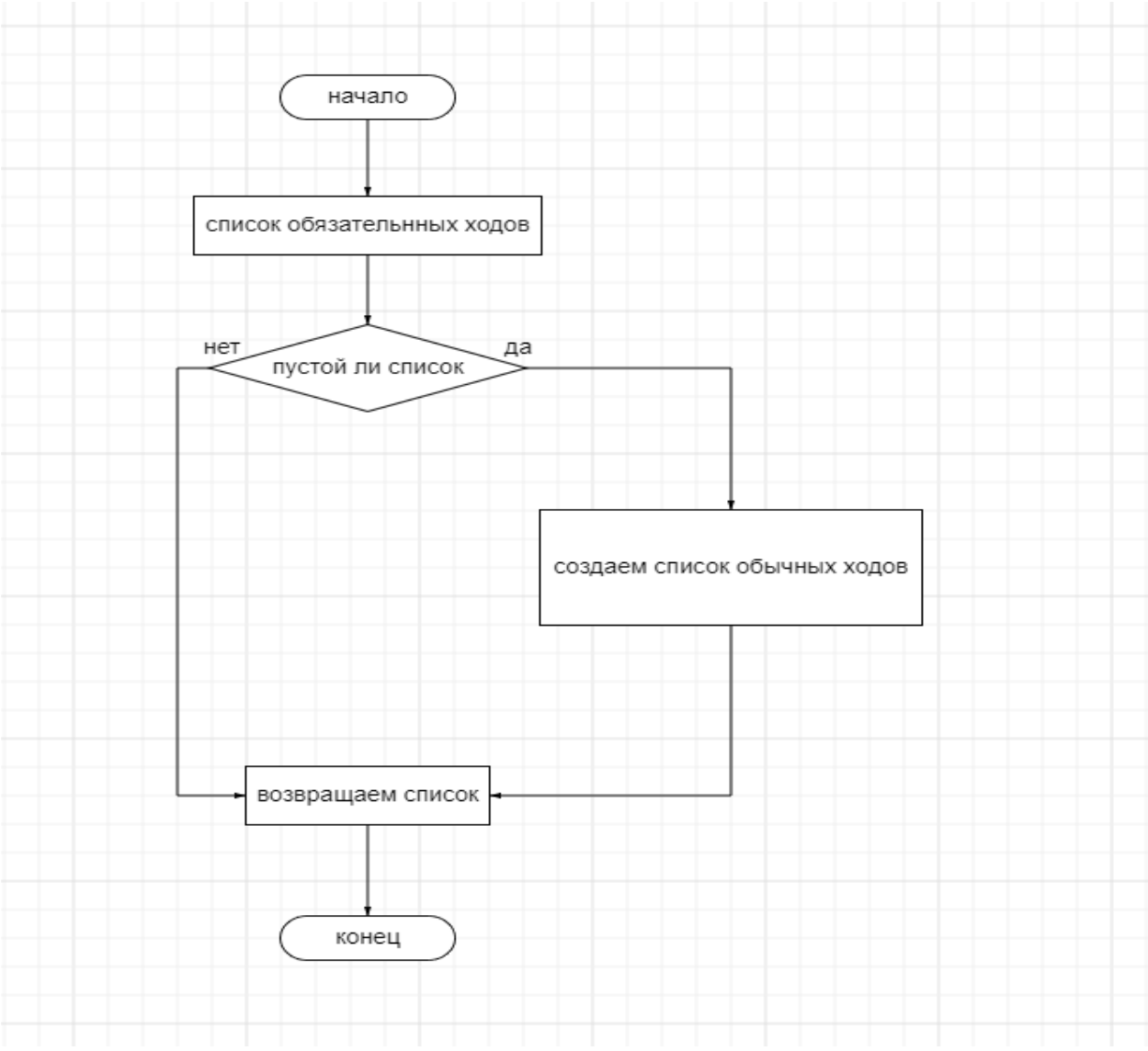
Математических методов нет

1.3 Архитектура и алгоритмы

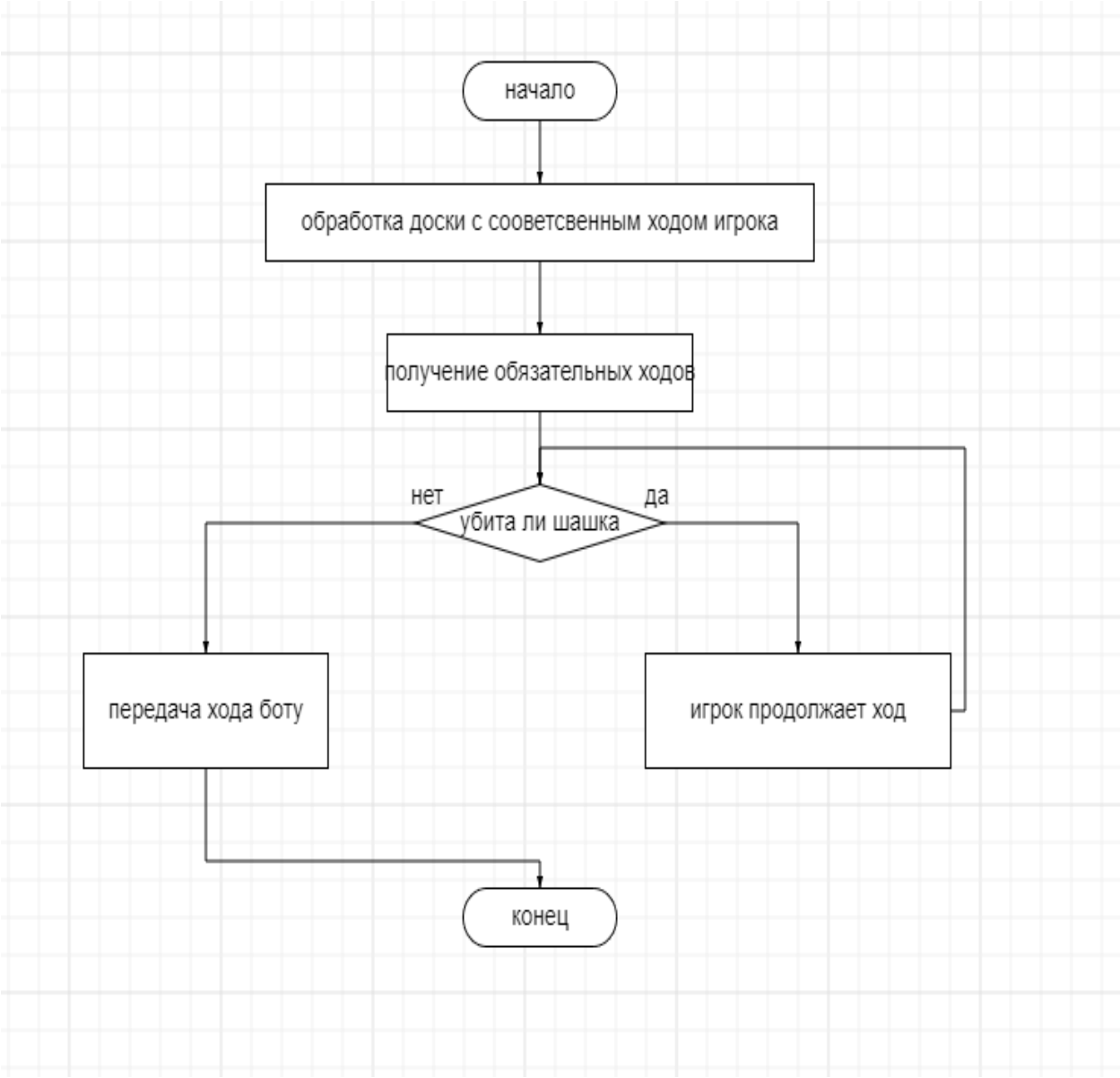
1.3.1. Архитектура



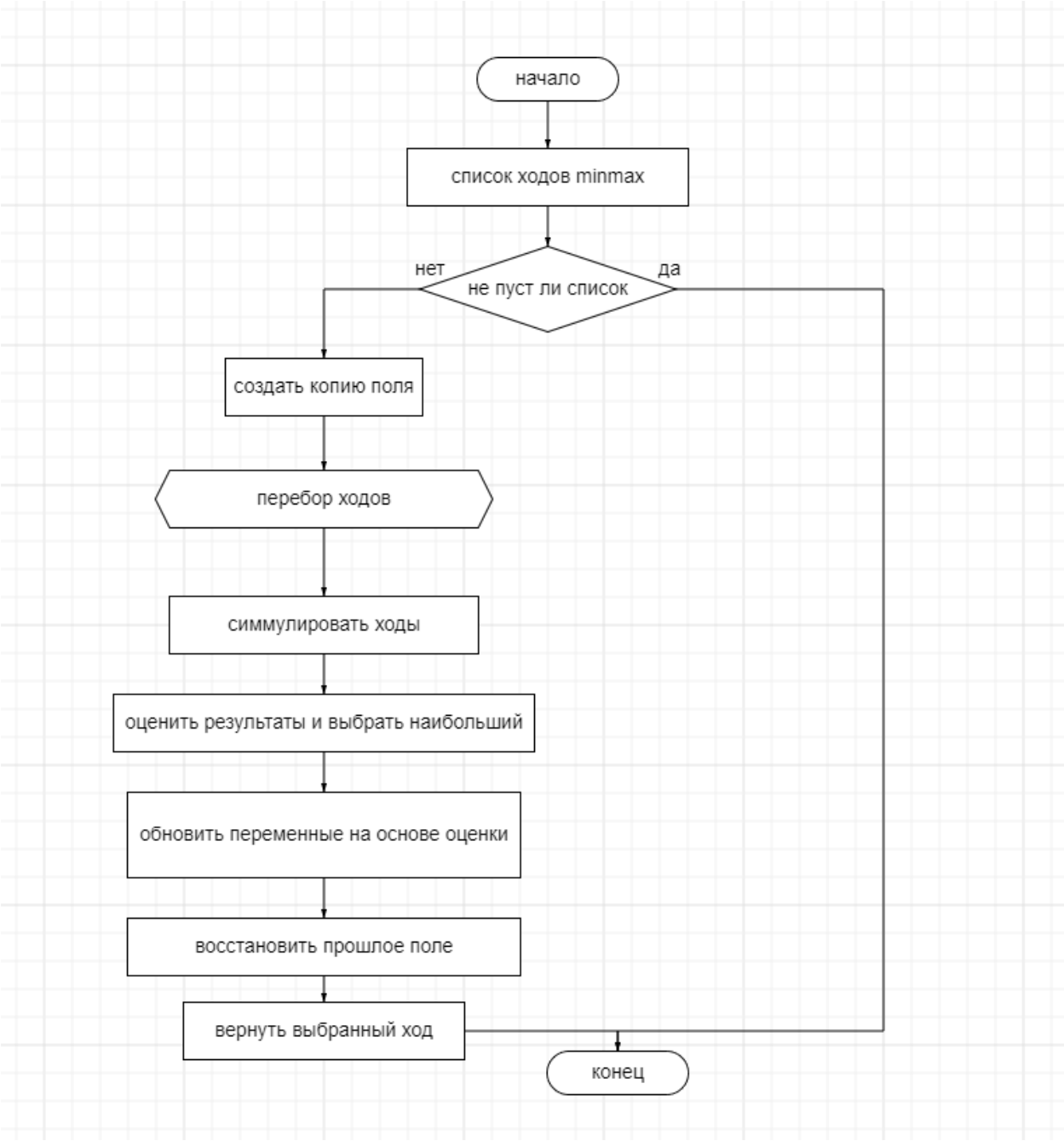
1.3.2. Алгоритм списка ходов



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 «Киммерийские шашки»

Руководство программиста

Р.02069337. 22/2385-20 РП-01

Листов: 6

Исполнитель:
студент гр. ИСТбд-22
Бутузова Дарья Евгеньевна
«15» февраля 2024 г.

2023 г.

1. Назначение и условия применения программы

1.1 Назначение и функции, выполняемые приложением

Десктопное приложение по теме игры Киммерийские 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. ОС 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».

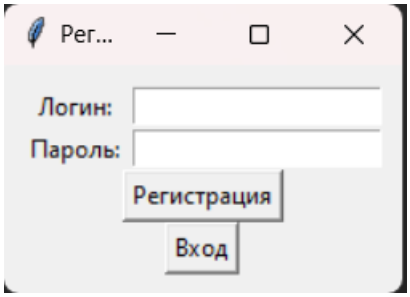


Рис. 1. Окно авторизации.

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. 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)

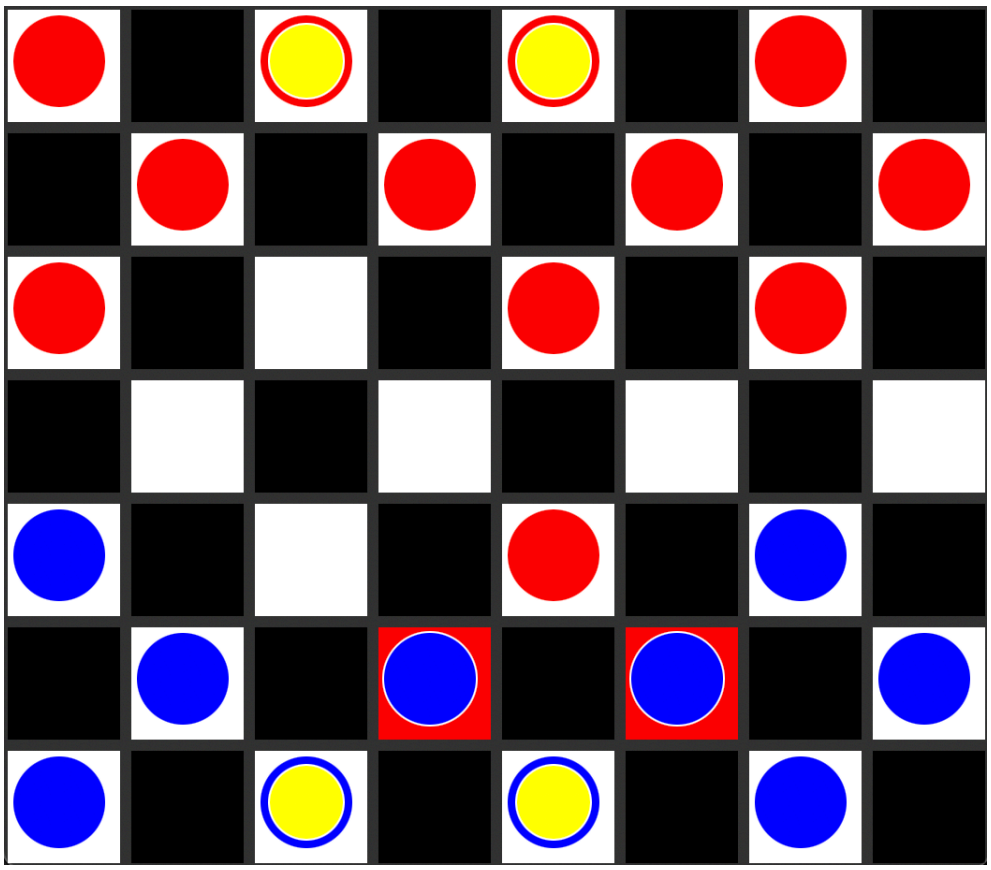


Рис. 4. Шашка компьютера сделала ход.

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)

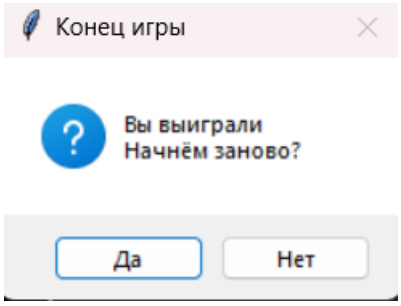


Рис. 5. Игра завершена.

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

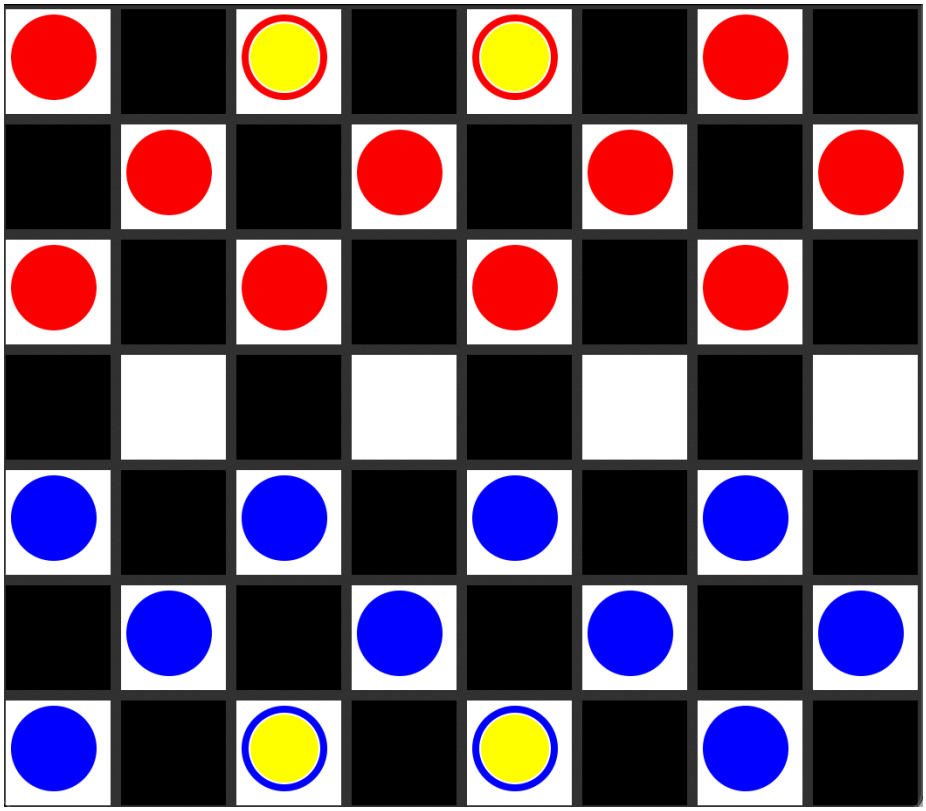
Алгоритмы и библиотеки.

4. Сообщения

При победе программа отображает победителя.

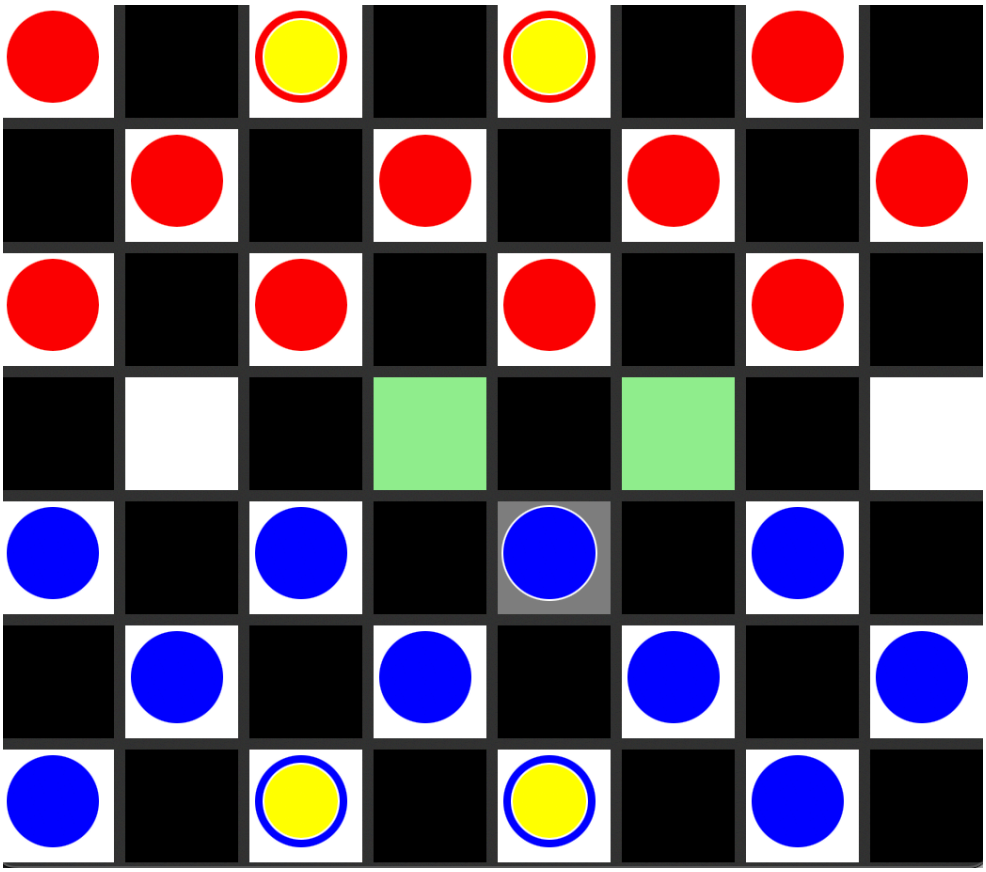
При вводе неправильного логина или пароля всплывает окно с сообщением при авторизации «Неверно введен пароль или логин».

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:

Checkers.py

```
import random
import sys
import tkinter as tk
from tkinter import messagebox


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



def get_cell_color(row, col):
    return "white" if (row + col) % 2 == 0 else "black"


def is_valid_cell(row, col):
    return 0 <= row < 7 and 0 <= col < 8


class CheckersBoard:
    def __init__(self, master):
        # Шашки
        self.cells = [[None for _ in range(8)] for _ in range(7)]
        # Канвасы, где мы ррисуем, по сути квадратики
        self.canvas_cells = [[None for _ in range(8)] for _ in range(7)]
        # 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

    def get_cell_color(self, row, col):
```

```
        if self.cells[row][col] is not None:
            return self.cells[row][col].color
        else:
            return "white" if (row + col) % 2 == 0 else "black"

    # Создаём шашки, а также рисуем их в канвасах
    def draw_starting_position(self):
        for row in range(7):
            for col in range(8):
                if get_cell_color(row, col) == "white" and (row + col) % 2 ==
0:
                    if row < 3:
                        checker_color = "white"
                        if row == 0 and (col == 2 or col == 4):
                            is_queen = True
                            checker = Checker(checker_color, row, col)
                            checker.is_queen = is_queen
                            self.place_checker(checker)

                            self.canvas_cells[row][col].create_oval(5, 5, 55,
55, fill=checker_color, tags="checker")
                            self.canvas_cells[row][col].create_oval(10, 10,
50, 50, fill='yellow',

tags="crown")
                            self.canvas_cells[row][col].is_queen = is_queen
                        else:
                            checker = Checker(checker_color, row, col)
                            self.place_checker(checker)
                            is_queen = False
                            self.canvas_cells[row][col].create_oval(5, 5, 55,
55, fill=checker_color, tags="checker")
                            self.canvas_cells[row][col].is_queen = is_queen
                    elif row > 3:
                        checker_color = "blue"
                        if row == 6 and (col == 2 or col == 4):
                            is_queen = True
                            checker = Checker(checker_color, row, col)
                            checker.is_queen = is_queen
                            self.place_checker(checker)

                            self.canvas_cells[row][col].create_oval(5, 5, 55,
55, fill=checker_color, tags="checker")
                            self.canvas_cells[row][col].create_oval(10, 10,
50, 50, fill='yellow',

tags="crown")
                            self.canvas_cells[row][col].is_queen = is_queen
                        else:
                            checker = Checker(checker_color, row, col)
                            self.place_checker(checker)
                            is_queen = False
                            self.canvas_cells[row][col].create_oval(5, 5, 55,
55, fill=checker_color, tags="checker")
                            self.canvas_cells[row][col].is_queen = is_queen

    # Создаем канвасы и окрашиваем их
    def create_gui(self):
        for row in range(7):
            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:
            # Получаем 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) > 1 and abs(prev_col - col) > 1:
                        # Проверяем, что атака происходит по диагонали
                        if abs(prev_row - row) == abs(prev_col - col):
                            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 7 > ch_row >= 0:
            if self.cells[ch_row][ch_col] is None and get_cell_color(ch_row,
ch_col) == 'white':
                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 проиграли\nНачнём заново?'
            else:
                message = 'Вы выиграли\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"] != 'black':
                    val.config(bg='white')

    # 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

    # 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(7):
            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 - 2 >= 0) and (row - 2 >= 0) and (self.cells[row - 1][col
- 1] is not None) and (
                    self.cells[row - 1][col - 1].color != color) and
(self.cells[row - 2][col - 2] is None):
                moves.append((row - 2, col - 2))
            if (col + 2 < 8) and (row - 2 >= 0) and (self.cells[row - 1][col
+ 1] is not None) and (
                    self.cells[row - 1][col + 1].color != color) and
(self.cells[row - 2][col + 2] is None):
                moves.append((row - 2, col + 2))
```

```
            if (col - 2 >= 0) and (row + 2 < 7) and (self.cells[row + 1][col
- 1] is not None) and (
                    self.cells[row + 1][col - 1].color != color) and
(self.cells[row + 2][col - 2] is None):
                moves.append((row + 2, col - 2))
            if (col + 2 < 8) and (row + 2 < 7) and (self.cells[row + 1][col +
1] is not None) and (
                    self.cells[row + 1][col + 1].color != color) and
(self.cells[row + 2][col + 2] is None):
                moves.append((row + 2, col + 2))
        else:
            moves.extend(self.find_queen_attack(row, col, -1, -1, color))
            moves.extend(self.find_queen_attack(row, col, -1, 1, color))
            moves.extend(self.find_queen_attack(row, col, 1, -1, color))
            moves.extend(self.find_queen_attack(row, col, 1, 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 7 > ch_row >= 0:  # Изменили условие на 7
для высоты
            if self.cells[ch_row][ch_col] is None and get_cell_color(ch_row,
ch_col) == 'white' 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(7):
            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 < 7 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

        deleted = False  # Флаг, указывающий, была ли фишка срублена
        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:
                    # Проверяем, что атака происходит по диагонали
                    if abs(prev_row - row) == abs(prev_col - col):
                        deleted = True
                        self.cells[r][c] = None
                        self.canvas_cells[r][c].delete('checker')
                        self.canvas_cells[r][c].delete('crown')

        return deleted

    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()
```

auth.py

```
from tkinter import *
from tkinter.messagebox import showerror, askyesno
import os

def check_file_exist():
    if not os.path.isfile("Reg.txt"):
        choice = askyesno("Ошибка!", " Отсутствует 'Reg.txt' файл. Хотите
создать его?")

        if choice:
            with open("Reg.txt", "w"):
                pass
        else:
            return False  # Возвращаем False, если окно было закрыто
пользователем
    return True

def check_login(login) -> bool:
    if check_file_exist():
        with open("Reg.txt", "r") as file:
            lines = file.readlines()
            login_input = login.get()
            for line in lines:
                if login_input in line:
                    return True
        return False

def check_users(login, password) -> bool:
    if check_file_exist():
        with open("Reg.txt", "r") as file:
            lines = file.readlines()
            login_input = login.get()
            password_input = password.get()
            for line in lines:
                parts = line.strip().split(':')
                if len(parts) == 2:
                    stored_login, stored_password = parts
                    if login_input == stored_login and password_input ==
stored_password:
                        return True
        return False

def registration_user(login, password):
    if not login.get() or not password.get():
        showerror("Ошибка", "Поля 'Логин' и 'Пароль' должны быть заполнены.")
        return False
    elif check_login(login):
        showerror("Ошибка", "Учетная запись с таким логином уже существует.")
        return False
    else:
        with open("Reg.txt", "a") as file:
            file.write(f"{login.get()}:{password.get()}\n")
        return True
```

```
def enter_users(login, password):
    if check_users(login=login, password=password):
        return True
    else:
        showerror("Ошибка", "Неверный логин или пароль.")
        return False

def create_window():
    root = Tk()
    w, h = root.winfo_width(), root.winfo_height()
    root.geometry(
        f"+{(root.winfo_screenwidth()-w)//2}+{(root.winfo_screenheight()-h)//
2}")
    root.title("Регистрация/Вход")

    frame = Frame(root)
    frame.grid(row=0, column=0, padx=10, pady=10)

    Label(frame, text="Логин: ").grid(row=0, column=0)
    Label(frame, text="Пароль: ").grid(row=1, column=0)

    login = Entry(frame)
    password = Entry(frame, show="*")

    login.grid(row=0, column=1)
    password.grid(row=1, column=1)

    def on_register():
        if registration_user(login, password):
            root.destroy()

    def on_enter():
        if enter_users(login, password):
            root.destroy()

    register_button = Button(frame, text="Регистрация", command=on_register)
    register_button.grid(row=2, columnspan=2)
    enter_button = Button(frame, text="Вход", command=on_enter)
    enter_button.grid(row=3, columnspan=3)

    root.protocol("WM_DELETE_WINDOW", lambda: root.quit())  # Обработка
закрытия окна пользователем

    root.mainloop()

    # Возвращаем True при успешном входе
    return True
```

main,py

```
import tkinter as tk
from Checkers import CheckersBoard
from auth import create_window


if __name__ == "__main__":
    if create_window():
        root = tk.Tk()
        checkers_board = CheckersBoard(root)
        checkers_board.start_game()
        root.mainloop()
```