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 «Мак Йек»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 22/256 20 ТЗ-1

Листов: 42

Руководитель разработки:

к. т. н., доцент

Шишкин Вадим Викторинович

«29» декабря 2023 г.

Исполнитель:

студент гр. ИСТбд-22

Зайцев Николай Сергеевич

«29» декабря 2023 г.

2023 г.

Содержание

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

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

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

Руководство программиста……………………………….………….......17

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

**Аннотация**

Данный документ представляет собой пояснительную записку на курсовую работу на тему логическая компьютерная игра «Мак Йек». 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 «Мак Йек»

Р.02069337. 22/256-20 ТЗ-1

Листов: 4

**Исполнитель**:

студент гр. ИСТбд-22

Зайцев Николай Сергеевич

«29» декабря 2023 г.

2023 г.

**Введение**

Наименование разрабатываемого приложения – Компьютерная логическая игра «Мак Йек»

На 64-клеточной доске соперники расставляют свои 8 шашек, которые размещаются на двух ближайших к человеку диагоналях, начиная с левого угла с промежутком на 1 клетку. Простые шашки ходят только вперед и рубят только вперед. Взятие шашки соперника обязательно и возможно, если за ней есть свободное поле. Если после взятия возможно продолжить брать шашки противника, то вы обязаны это делать. Ходить и есть назад простая шашка не может. Шашка, дошедшая до последней горизонтали, становится дамкой. Дамка ходит во всех четырех направлениях и рубит также, но тайская дамка обязана встать на поле сразу за срубленной шашкой. Дамка в процессе рубки может два раза пройти через пустое поле, но только если на этом поле не стояла шашка противника, которую дамка на этом ходу уже рубила. Цель игры в Мак Йек (они же тайские шашки) —заблокировать или захватить все фигуры противника, оставив его без права хода. Приложение предоставляет:

* Регистрация/авторизация пользователя.
* Правильно графически отображать ходы на доске.
* Проверять правильность ходов в соответствии с правилами и не давать возможность делать не правильные ходы.
* Проверять окончание игры и определять победителя.

**1. Основания для разработки**

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

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

**2.1. Функциональное назначение**

Требуется разработать однопользовательское десктопное приложение по игре в логическую игру “Мак Йек” с графическим интерфейсом в среде Windows.

Перечень автоматизируемых процессов:

* Отрисовка игрового поля;
* Загрузка изображений пешек;
* Отрисовка пешек;
* Проверка наличия доступных ходов;
* Контроль движения мышки;
* Создание и сохранение аккаунта в текстовый файл;
* Проверка на правильность ввода логина или пароля;
* Определение победителя

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Приложение должно состоять из модуля (блока), который выполняет определенные функции по организации пользовательского интерфейса и самого процесса игры. При необходимости модуль должен обладать дополнительными информационными файлами.

2.2.2 Требования к составу функций приложения

В приложении должны быть реализованы в графическом режиме следующие

основные функции:

- регистрация/авторизация пользователя;

- реализация игрового поля;

- взаимодействие с пользователем;

- проверка правильности и отрисовка хода пользователя;

- перезапуск игры

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода компьютера;

- сообщение об окончании игры и победителе.

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

В десктопном приложении должен быть реализован графический интерфейс для взаимодействия с пользователем. Окно с полями для ввода логина и пароля, которые вводятся с клавиатуры и хранятся в отдельном файле или базе данных в зашифрованном виде. Кнопка регистрации/авторизации пользователя. Вывод окна игрового поля – шашки. Изображения шашек могут храниться в отдельных файлах.

**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, pillow, math, random, copy, enum

Язык: Python 3.10.4

**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

Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы

2.6 3 Сроки хранения

Срок хранения – до окончания учебного 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 «Мак Йек»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

**Пояснительная записка**

Р.02069337. 22/256-20 ПЗ-01

Листов: 7

**Исполнитель**:

студент гр. ИСТбд-22

Зайцев Николай Сергеевич

«29» декабря 2023 г.

2023 г.

**Введение**

Курсовая работа представляет собой десктопное приложение по теме игры Мак Йек

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**

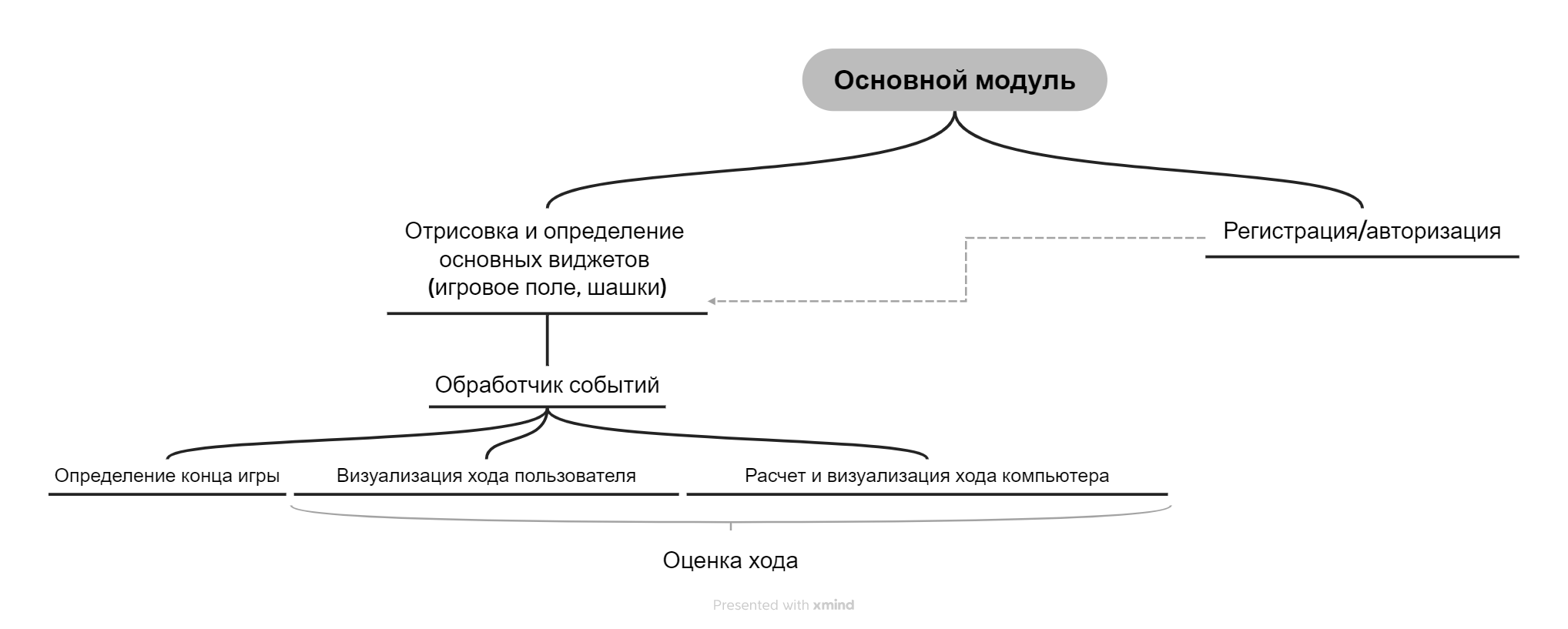
Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании.

**1.2 Математические методы**

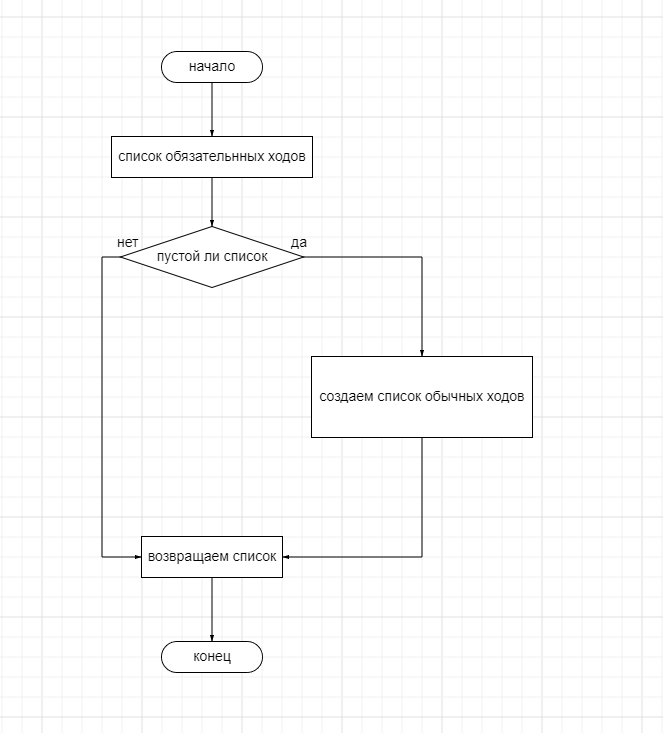
Математических методов нет

**1.3 Архитектура и алгоритмы**

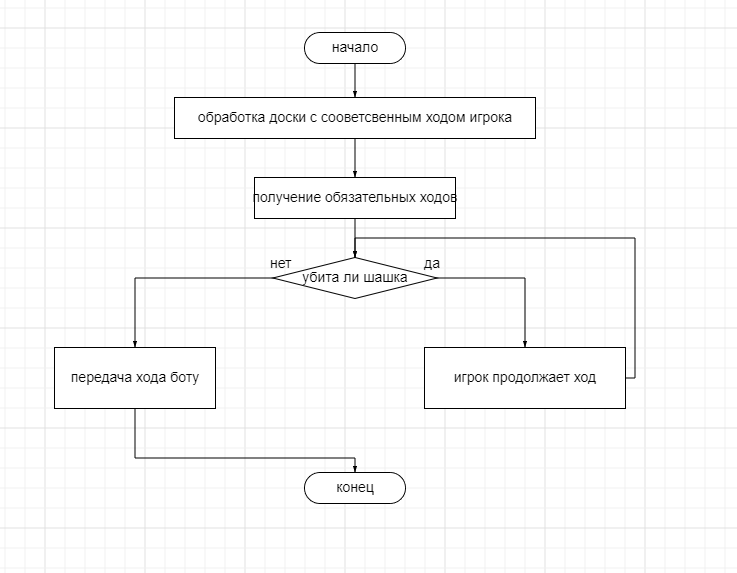
1.3.1. Архитектура



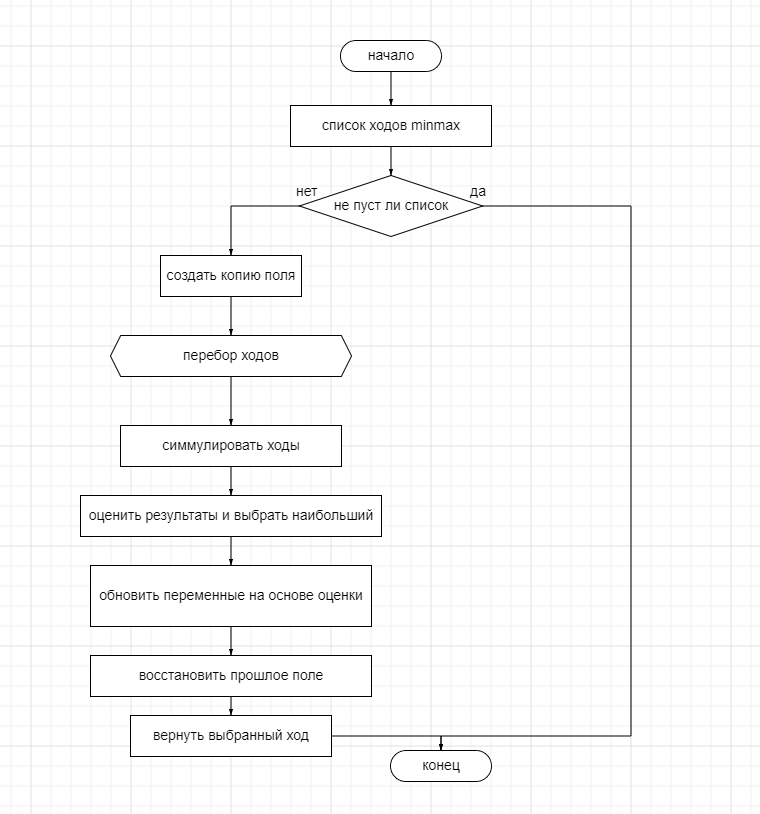
1.3.2. Алгоритм cписка ходов



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 логином: user и паролем 123user.

Шаги:

1. Запустить приложение.

2. В окне регистрации, в поле «логин» ввести user, а в поле «пароль» - 123user.

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)

7. Python Checkers// YouTube URL:  
https://youtube.com/playlist?list=PLzMcBGfZo4lkJr3sqpikNyVzbNZLRiT3&si=UkWBXgBuCdS6SnAo (дата обращения 13.12.2023)

**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 «Мак Йек»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

**Руководство программиста**

Р.02069337. 22/256-20 РП-01

Листов: 6

Исполнитель:

студент гр. ИСТбд-22

Зайцев Николай Сергеевич

«29» декабря 2023 г.

2023 г.

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Десктопное приложение по теме игры Мак Йек.

Краткие правила игры:

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

* В распоряжении игрока 8 шашек, а не привычные 12.
* Есть шашку всегда обязательно, обычная же шашка ест только вперед.
* Дамка всегда встает за съеденной шашкой

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, pillow

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк кода – 597.

Количество алгоритмов – 12.

Библиотеки tkinter, math, random, pillow, os, enum

Порядок работы:

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

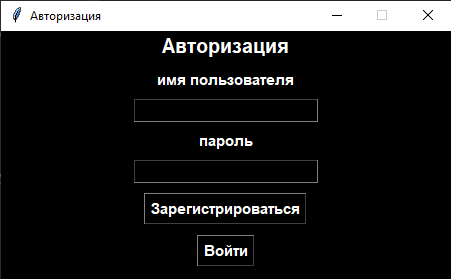


Рис. 1. Окно авторизации.

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)

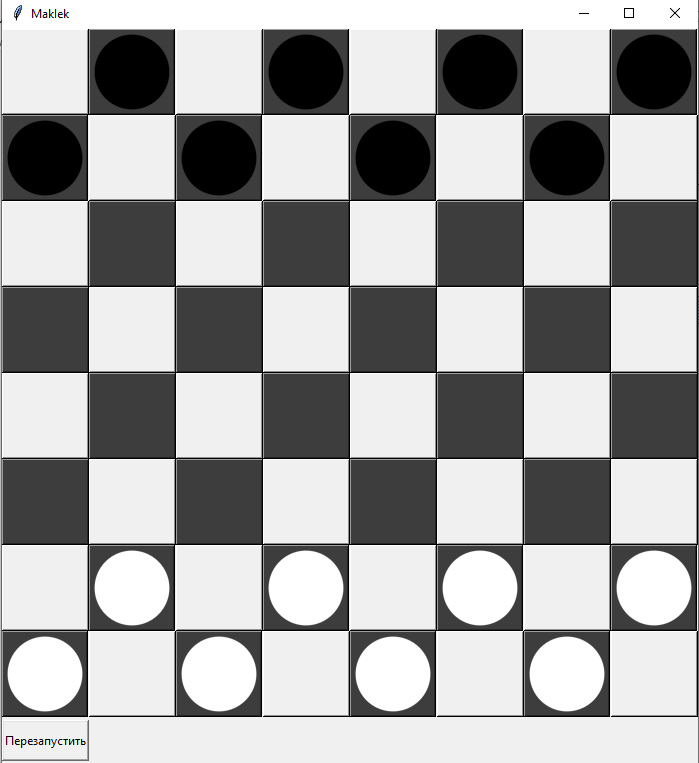


Рис. 2. Окно игры.

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

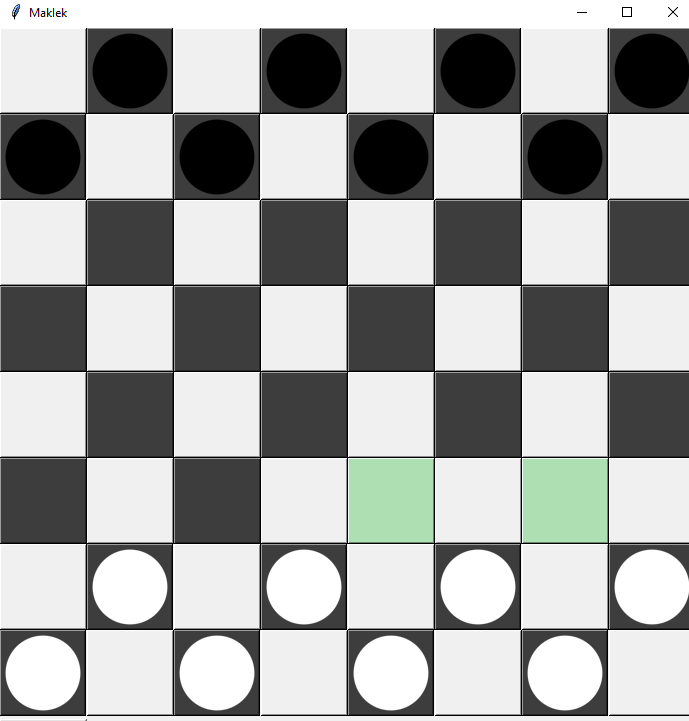


Рис. 3. Шашка выбрана игроком и ее возможные ходы подсвечены

После хода белыми шашками, право хода приходит черным, ход будет делать компьютер. (Рис. 4)

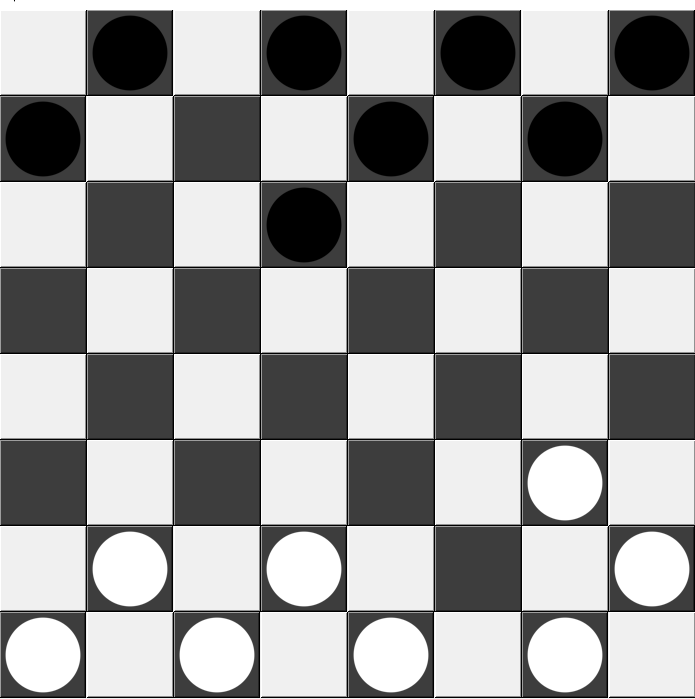


Рис. 4. Шашка компьютера сделала ход.

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)

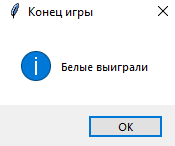


Рис. 5. Игра завершена.

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

**2.2 Особенности реализации приложения**

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

**3. Обращение к программе**

Алгоритмы:

1. «login\_user» - алгоритм, отвечающий за авторизацию пользователя.
2. «register\_user» - алгоритм, отвечающий за регистрацию пользователя
3. «button\_click» – алгоритм, отвечающий за обработку нажатия на доску
4. «handle\_move» - алгоритм, совершения хода.
5. «handle\_player\_turn» - алгоритм, обрабатывающий ход игрока.
6. «handle\_bot\_turn» - алгоритм, обрабатывающий ход бота
7. «check\_game\_over» - алгоритм, проверяющий концовку игры
8. «get\_minmax\_moves» - алгоритм, используется для принятия решений компьютерным игроком при выборе оптимальных ходов
9. «get\_minmax\_moves\_list» - алгоритм, подсчитывающий список оптимальных ходов
10. «get\_ moves\_list» - алгоритм, подсчитывающий список все ходов
11. «get\_required\_moves\_list» - алгоритм, подсчитывающий список обязательных ходов
12. «get\_optional\_moves\_list» - алгоритм, подсчитывающий список ходов(необязательных)

Библиотеки:

1. tkinter – для работы с графическим интерфейсом игры
2. copy - для копирования игрового поля
3. random – для выбора рандомного хода из доступных
4. pillow – для загрузки фоток
5. enum – для перечеслений
6. os – для работы с файлом данных о зарегистрированных пользователях

**4. Сообщения**

При победе программа отображает победителя, в виде сообщений: «Черные выиграли» или «Белые выиграли».

При вводе неправильного логина или пароля всплывает окно с сообщением при авторизации «Ошибка входа. Пожалуйста, проверьте ваше имя пользователя и пароль».

При успешной авторизации всплывает окно «Вход выполнен успешно»

При пустых или пустом поле при авторизации всплывает окно «Пожалуйста, заполните оба поля”.

При попытке регистрации с существующим логином всплывет окно с сообщением «Пользователь с таким именем уже существует».

При успешной регистрации всплывает окно «Пожалуйста, войдите с вашим логином и паролем

При регистрации при пустых или пустом поле всплывает окно «Пожалуйста, заполните оба поля»

**Текст программы:**

import os

import tkinter as tk # Библиотека для реализации интерфейса

from tkinter import messagebox

from enum import auto, Enum # Библиотека для реализации перечислений

from math import inf

from random import choice # Библиотека для реализации случайного выбора

from PIL import ImageTk

from PIL import Image

# Класс типа стороны игрока

class SideType(Enum):

WHITE = auto()

BLACK = auto()

# Противоположный тип

def opposite(self):

return SideType.BLACK if self == SideType.WHITE else SideType.WHITE

# Класс типа шашки

class CheckerType(Enum):

NONE = auto()

WHITE\_STANDARD = auto()

BLACK\_STANDARD = auto()

WHITE\_QUEEN = auto()

BLACK\_QUEEN = auto()

# Класс точки

class Point:

def \_\_init\_\_(self, x=-1, y=-1):

self.x = x

self.y = y

def \_\_eq\_\_(self, other):

return (self.x == other.x and

self.y == other.y)

# Количество ходов для minmax функции (чем больше, тем сильнее противник)

MINMAX\_DEPTH = 3

# Возможные dx и dy шашек

MOVE\_OFFSETS = [

Point(-1, -1),

Point(1, -1),

Point(-1, 1),

Point(1, 1)

]

# Типы белых и чёрных шашек

WHITE\_CHECKERS = [CheckerType.WHITE\_STANDARD, CheckerType.WHITE\_QUEEN]

BLACK\_CHECKERS = [CheckerType.BLACK\_STANDARD, CheckerType.BLACK\_QUEEN]

# Класс шашки

class Checker:

def \_\_init\_\_(self, type=CheckerType.NONE):

self.type = type

# Белая ли шашка

def is\_white(self):

return self.type == CheckerType.WHITE\_STANDARD or self.type == CheckerType.WHITE\_QUEEN

# Черная ли шашка

def is\_black(self):

return self.type == CheckerType.BLACK\_STANDARD or self.type == CheckerType.BLACK\_QUEEN

# Символ шашки

def symbol(self):

if self.type == CheckerType.WHITE\_STANDARD:

return 'w'

if self.type == CheckerType.WHITE\_QUEEN:

return 'W'

if self.type == CheckerType.BLACK\_STANDARD:

return 'b'

if self.type == CheckerType.BLACK\_QUEEN:

return 'B'

return ' '

# Класс доски

class Field:

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

self.size = size

# Инициализация матрицы доски

self.checkers = [[Checker() for x in range(self.size)] for y in range(self.size)]

# Инициализация начальных позиций пешек

for y in range(self.size):

for x in range(self.size):

if (y + x) % 2:

if y < 2:

self.checkers[y][x].type = CheckerType.BLACK\_STANDARD

elif y >= self.size - 2:

self.checkers[y][x].type = CheckerType.WHITE\_STANDARD

# Метод копирования доски

@classmethod

def copy(cls, field\_instance):

field\_copy = cls(field\_instance.size)

for y in range(field\_instance.size):

for x in range(field\_instance.size):

field\_copy.at(x, y).type = field\_instance.type\_at(x, y)

return field\_copy

# Тип шашки по координатам

def type\_at(self, x, y):

return self.checkers[y][x].type

# Шашка по координатам

def at(self, x, y):

return self.checkers[y][x]

# Находится ли точка в пределах доски

def is\_within(self, x, y):

return 0 <= x < self.size and 0 <= y < self.size

# Счет белых

def white\_score(self):

score = 0

for row in self.checkers:

for checker in row:

if checker.type == CheckerType.WHITE\_STANDARD:

score += 1

elif checker.type == CheckerType.WHITE\_QUEEN:

score += 3

return score

# Счет черных

def black\_score(self):

score = 0

for row in self.checkers:

for checker in row:

if checker.type == CheckerType.BLACK\_STANDARD:

score += 1

elif checker.type == CheckerType.BLACK\_QUEEN:

score += 3

return score

# Класс хода

class Move:

def \_\_init\_\_(self, from\_x=-1, from\_y=-1, to\_x=-1, to\_y=-1):

self.from\_x = from\_x

self.from\_y = from\_y

self.to\_x = to\_x

self.to\_y = to\_y

def \_\_eq\_\_(self, other):

return (self.from\_x == other.from\_x and

self.from\_y == other.from\_y and

self.to\_x == other.to\_x and

self.to\_y == other.to\_y)

# Класс игры

class MakIek(tk.Tk):

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

super().\_\_init\_\_()

self.white = ImageTk.PhotoImage(Image.open(r"Images\white.png").resize((80, 80)))

self.whiteq = ImageTk.PhotoImage(Image.open(r"Images\whiteq.png").resize((80, 80)))

self.dark = ImageTk.PhotoImage(Image.open(r"Images\dark.png").resize((80, 80)))

self.darkq = ImageTk.PhotoImage(Image.open(r"Images\darkq.png").resize((80, 80)))

self.title("MakIek")

self.button\_matrix = []

# Инициализация кнопок

for i in range(8):

row\_buttons = []

for j in range(8):

button = tk.Button(self, text="",

command=lambda i=i, j=j: self.button\_click(i, j),

width=11,

height=5)

button.grid(row=i,

column=j,

padx=0,

pady=0)

row\_buttons.append(button)

self.button\_matrix.append(row\_buttons)

self.restartBtn = tk.Button(self,

text="Перезапустить",

command=lambda: self.restart\_game(),

width=11, height=2)

self.restartBtn.grid(row=8, column=0, padx=0, pady=3)

self.restart\_game()

# Перезапуск игры

def restart\_game(self):

self.field = Field(8)

self.player\_turn = True

self.hovered\_cell = Point()

self.selected\_cell = Point()

self.animated\_cell = Point()

self.update\_buttons()

# Обновление текста кнопок

def update\_buttons(self):

for i in range(8):

for j in range(8):

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

bg = "#F0F0F0"

else:

bg = "#3D3D3D"

symbol = self.field.at(j, i).symbol()

if symbol == "w":

image = self.white

elif symbol == "W":

image = self.whiteq

elif symbol == "b":

image = self.dark

elif symbol == "B":

image = self.darkq

else:

moves = self.get\_moves\_list(SideType.WHITE)

if Move(self.selected\_cell.x, self.selected\_cell.y, j, i) in moves:

self.button\_matrix[i][j].config(height=5, width=11, image='', bg="#ADDFB3")

else:

self.button\_matrix[i][j].config(height=5, width=11, image='', bg=bg)

continue

self.button\_matrix[i][j].config(height=80, width=80, image=image, bg=bg)

self.update()

# Обработчик нажатий на кнопки доски

def button\_click(self, i, j):

if not self.player\_turn:

return

x, y = j, i

# Если произошло нажатие на белую пешку

if self.field.type\_at(x, y) in WHITE\_CHECKERS:

self.selected\_cell = Point(x, y)

self.update\_buttons()

# Выбрана позиция хода для выбранной пешки

elif self.player\_turn:

move = Move(self.selected\_cell.x, self.selected\_cell.y, x, y)

if move in self.get\_moves\_list(SideType.WHITE):

self.handle\_player\_turn(move)

self.update\_buttons()

if not self.player\_turn:

self.handle\_bot\_turn()

self.update\_buttons()

self.check\_for\_game\_over()

# Обработка хода

def handle\_move(self, move):

# Изменение типа шашки по достижении противоположной стороны доски

if move.to\_y == 0 and self.field.type\_at(move.from\_x, move.from\_y) == CheckerType.WHITE\_STANDARD:

self.field.at(move.from\_x, move.from\_y).type = CheckerType.WHITE\_QUEEN

elif (move.to\_y == self.field.size - 1 and self.field.type\_at(move.from\_x,

move.from\_y) == CheckerType.BLACK\_STANDARD):

self.field.at(move.from\_x, move.from\_y).type = CheckerType.BLACK\_QUEEN

# Перестановка

self.field.at(move.to\_x, move.to\_y).type = self.field.type\_at(move.from\_x, move.from\_y)

self.field.at(move.from\_x, move.from\_y).type = CheckerType.NONE

# Вектор направления движения пешки

dx = -1 if move.from\_x < move.to\_x else 1

dy = -1 if move.from\_y < move.to\_y else 1

has\_killed\_checker = False

x, y = move.to\_x, move.to\_y

# Пока не достигнута позиция хода

while x != move.from\_x or y != move.from\_y:

# Шаг

x += dx

y += dy

# Встречена пешка на пути

if self.field.type\_at(x, y) != CheckerType.NONE:

self.field.at(x, y).type = CheckerType.NONE

has\_killed\_checker = True

return has\_killed\_checker

# Обработка хода игрока

def handle\_player\_turn(self, move):

self.player\_turn = False

has\_killed\_checker = self.handle\_move(move)

required\_moves\_list = list(

filter(lambda required\_move: move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y,

self.get\_required\_moves\_list(SideType.WHITE)))

if has\_killed\_checker and required\_moves\_list:

self.player\_turn = True

self.selected\_cell = Point()

# Обработка хода бота

def handle\_bot\_turn(self):

self.player\_turn = False

optimal\_moves\_list = self.get\_minmax\_moves(SideType.opposite(SideType.WHITE))

for move in optimal\_moves\_list:

self.handle\_move(move)

self.player\_turn = True

# Проверка на окончание игры

def check\_for\_game\_over(self):

game\_over = False

white\_moves\_list = self.get\_moves\_list(SideType.WHITE)

if not white\_moves\_list:

messagebox.showinfo('Конец игры', 'Чёрные выиграли')

game\_over = True

black\_moves\_list = self.get\_moves\_list(SideType.BLACK)

if not black\_moves\_list:

messagebox.showinfo('Конец игры', 'Белые выиграли')

game\_over = True

if game\_over:

self.restart\_game()

# Оптимальные ходы

def get\_minmax\_moves(self, side):

best\_result = 0

optimal\_moves = []

minmax\_moves\_list = self.get\_minmax\_moves\_list(side)

# Есть оптимальные ходы

if minmax\_moves\_list:

field\_copy = Field.copy(self.field)

# Перебор оптимальных ходов

for moves in minmax\_moves\_list:

for move in moves:

# Имитация хода

self.handle\_move(move)

try:

if side == SideType.WHITE:

result = self.field.white\_score() / self.field.black\_score()

elif side == SideType.BLACK:

result = self.field.black\_score() / self.field.white\_score()

except ZeroDivisionError:

result = inf

# Оценка

if result > best\_result:

best\_result = result

optimal\_moves.clear()

optimal\_moves.append(moves)

elif result == best\_result:

optimal\_moves.append(moves)

self.field = Field.copy(field\_copy)

optimal\_move = []

if optimal\_moves:

for move in choice(optimal\_moves):

if side == SideType.WHITE and self.field.type\_at(move.from\_x, move.from\_y) in BLACK\_CHECKERS:

break

elif side == SideType.BLACK and self.field.type\_at(move.from\_x, move.from\_y) in WHITE\_CHECKERS:

break

optimal\_move.append(move)

return optimal\_move

# Список оптимальных ходов

def get\_minmax\_moves\_list(self, side, minmax\_depth=0, all\_moves\_list=[], current\_moves\_list=[],

required\_moves\_list=[]):

if current\_moves\_list:

all\_moves\_list.append(current\_moves\_list)

else:

all\_moves\_list.clear()

if required\_moves\_list:

moves\_list = required\_moves\_list

else:

moves\_list = self.get\_moves\_list(side)

# Есть ходы и максимальная глубина minmax функции еще не достигнута

if moves\_list and minmax\_depth < MINMAX\_DEPTH:

field\_copy = Field.copy(self.field)

# Перебор ходов

for move in moves\_list:

has\_killed\_checker = self.handle\_move(move)

required\_moves\_list = list(filter(

lambda required\_move: move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y,

self.get\_required\_moves\_list(side)))

# Рекурсивные вызовы для построения minmax дерева ходов

if has\_killed\_checker and required\_moves\_list:

self.get\_minmax\_moves\_list(side, minmax\_depth, all\_moves\_list,

current\_moves\_list + [move], required\_moves\_list)

else:

self.get\_minmax\_moves\_list(SideType.opposite(side), minmax\_depth + 1,

all\_moves\_list, current\_moves\_list + [move])

self.field = Field.copy(field\_copy)

return all\_moves\_list

# Список всех ходов

def get\_moves\_list(self, side):

# Обязательные ходы

moves\_list = self.get\_required\_moves\_list(side)

# Нет обязательных ходов

if not moves\_list:

# Обычные ходы

moves\_list = self.get\_optional\_moves\_list(side)

return moves\_list

# Обязательные ходы (рубка)

def get\_required\_moves\_list(self, side):

moves\_list = []

# Выбор дружественных шашек

if side == SideType.WHITE:

friendly\_checkers = WHITE\_CHECKERS

enemy\_checkers = BLACK\_CHECKERS

elif side == SideType.BLACK:

friendly\_checkers = BLACK\_CHECKERS

enemy\_checkers = WHITE\_CHECKERS

else:

return moves\_list

# Перебор всех координат доски

for y in range(self.field.size):

for x in range(self.field.size):

# Стандартная шашка

if self.field.type\_at(x, y) == friendly\_checkers[0]:

for offset in MOVE\_OFFSETS:

# Рубка не по правилам

if (side == SideType.WHITE and y <= y + offset.y or

side == SideType.BLACK and y >= y + offset.y):

continue

# Выход за пределы доски

if not (self.field.is\_within(x + offset.x \* 2, y + offset.y \* 2)):

continue

# Рубка по правилам

if self.field.type\_at(x + offset.x, y + offset.y) in enemy\_checkers and self.field.type\_at(

x + offset.x \* 2, y + offset.y \* 2) == CheckerType.NONE:

moves\_list.append(Move(x, y, x + offset.x \* 2, y + offset.y \* 2))

# Дамка

elif self.field.type\_at(x, y) == friendly\_checkers[1]:

for offset in MOVE\_OFFSETS:

# Выход за пределы доски

if not (self.field.is\_within(x + offset.x \* 2, y + offset.y \* 2)):

continue

has\_enemy\_checker\_on\_way = False

# Перебор пути дамки

for shift in range(1, self.field.size):

# Выход за пределы доски

if not (self.field.is\_within(x + offset.x \* shift, y + offset.y \* shift)):

continue

# Нет вражеской пешки на пути

if not has\_enemy\_checker\_on\_way:

# Найдена вражеская пешка

if self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) in enemy\_checkers:

has\_enemy\_checker\_on\_way = True

continue

# Найдена дружественная пешка

elif (self.field.type\_at(x + offset.x \* shift,

y + offset.y \* shift) in friendly\_checkers):

break

# Есть вражеская пешка на пути

if has\_enemy\_checker\_on\_way:

# Найдена пустая позиция

if (self.field.type\_at(x + offset.x \* shift,

y + offset.y \* shift) == CheckerType.NONE):

moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))

break

else:

break

return moves\_list

# Обычные ходы (перемещение)

def get\_optional\_moves\_list(self, side):

moves\_list = []

# Выбор дружественных шашек

if side == SideType.WHITE:

friendly\_checkers = WHITE\_CHECKERS

elif side == SideType.BLACK:

friendly\_checkers = BLACK\_CHECKERS

else:

return moves\_list

# Перебор всех координат доски

for y in range(self.field.size):

for x in range(self.field.size):

# Стандартная шашка

if self.field.type\_at(x, y) == friendly\_checkers[0]:

for offset in MOVE\_OFFSETS[:2] if side == SideType.WHITE else MOVE\_OFFSETS[2:]:

# Выход за пределы доски

if not (self.field.is\_within(x + offset.x, y + offset.y)):

continue

# Ход на пустую позицию

if self.field.type\_at(x + offset.x, y + offset.y) == CheckerType.NONE:

moves\_list.append(Move(x, y, x + offset.x, y + offset.y))

# Дамка

elif self.field.type\_at(x, y) == friendly\_checkers[1]:

for offset in MOVE\_OFFSETS:

# Выход за пределы доски

if not (self.field.is\_within(x + offset.x, y + offset.y)):

continue

# Перебор пути дамки

for shift in range(1, self.field.size):

# Выход за пределы доски

if not (self.field.is\_within(x + offset.x \* shift, y + offset.y \* shift)):

continue

# Ход на пустую позицию

if self.field.type\_at(x + offset.x \* shift, y + offset.y \* shift) == CheckerType.NONE:

moves\_list.append(Move(x, y, x + offset.x \* shift, y + offset.y \* shift))

else:

break

return moves\_list

def register\_user():

username = username\_entry.get()

password = password\_entry.get()

if username and password:

if user\_exists(username):

messagebox.showerror("Ошибка регистрации", "Пользователь с таким именем уже существует.")

else:

save\_user(username, password)

messagebox.showinfo("Регистрация успешна",

"Регистрация прошла успешно. Пожалуйста, войдите с вашим именем и паролем.")

clear\_entries()

else:

messagebox.showwarning("Ошибка регистрации", "Пожалуйста, заполните оба поля.")

def login\_user():

global isLogIn

username = username\_entry.get()

password = password\_entry.get()

if username and password:

if user\_exists(username) and check\_password(username, password):

messagebox.showinfo("Вход выполнен успешно", "Вход выполнен успешно.")

clear\_entries()

isLogIn = True

root.destroy()

else:

messagebox.showerror("Ошибка входа", "Ошибка входа. Пожалуйста, проверьте ваше имя пользователя и пароль.")

else:

messagebox.showwarning("Ошибка входа", "Пожалуйста, заполните оба поля.")

def clear\_entries():

username\_entry.delete(0, tk.END)

password\_entry.delete(0, tk.END)

def user\_exists(username):

if check\_file\_exist():

with open("users.txt", "r") as file:

for line in file:

if username == line.strip().split(':')[0]:

return True

return False

def check\_password(username, password):

if check\_file\_exist():

with open("users.txt", "r") as file:

for line in file:

if line.strip().split(':')[0] == username:

if line.strip().split(':')[1] == password:

return True

return False

def check\_file\_exist():

if not os.path.isfile("users.txt"):

choice = messagebox.askyesno(

"Ошибка!", " Отсутствует 'users.txt' файл. Хотите создать его?")

if choice:

with open("users.txt", "w"):

pass

else:

exit()

return True

def save\_user(username, password):

with open("users.txt", "a") as file:

file.write(f"{username}:{password}\n")

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

isLogIn = False

root = tk.Tk()

root.title("Авторизация")

root.geometry("450x250")

root.resizable(width=False, height=False)

root['bg'] = 'black'

main\_label = tk.Label(root, text='Авторизация', font='Arial 15 bold', bg='black', fg='white')

main\_label.pack()

username\_label = tk.Label(root, text="имя пользователя", font='Arial 11 bold', bg='black', fg='white', padx=10,

pady=8)

username\_label.pack()

username\_entry = tk.Entry(root, bg='black', fg='Lime', font='Arial 12 bold')

username\_entry.pack()

password\_label = tk.Label(root, text="пароль", font='Arial 11 bold', bg='black', fg='white', padx=10, pady=8)

password\_label.pack()

password\_entry = tk.Entry(root, bg='black', fg='Lime', font='Arial 12 bold', show='\*')

password\_entry.pack()

register\_button = tk.Button(root, text="Зарегистрироваться", font='Arial 11 bold', bg='black', fg='white',

command=register\_user)

register\_button.pack(anchor=tk.CENTER, pady=10)

login\_button = tk.Button(root, text="Войти", font='Arial 11 bold', bg='black', fg='white', command=login\_user)

login\_button.pack(anchor=tk.CENTER)

root.mainloop()

if (isLogIn):

app = MakIek()

app.mainloop()