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 Канадские шашки

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 22/2402 10 ТЗ-01

Листов 46

Руководитель разработки:

Шишкин Вадим

Викторинович

« » 2023 г.

Исполнитель:

студент гр. ИСТбд-21

Никонова Дарья Александровна

« » 2023 г.

2023

Содержание

Аннотация……...…………………………………………………………3

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

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

Руководство пользователя……………………………….…………..15

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

**АННОТАЦИЯ**

Тема курсового проекта: Компьютерная логическая игра «Канадские шашки»

Исполнитель: студент гр. ИСТбд-21 Никонова Дарья Александровна

Руководитель разработки: Шишкин Вадим Викторинович

Работа состоит из технического задания, пояснительной записки, руководства программиста и текста программы.

В техническом задании описаны общие правила компьютерной логической игры «Канадские шашки», условия выигрыша, проигрыша и ничьей, начальные позиции фигур. Приведены основания для разработки, функциональное назначение, основные требования к функциональным характеристикам, надежности, информационной и программной совместимости, хранению, транспортировке, программной документации.

В пояснительной записке указываются задачи, математические методы, архитектура и алгоритмы, тестирование и источники, использованные при разработке.

В руководстве программиста приводятся назначение и функции, выполняемые приложением, условия использования, характеристики приложения, особенности реализации приложения, обращение к программе, сообщения, выдаваемые по результатам контроля корректности ввода/вывода

Текст программы представляет собой полный код программы, реализующий компьютерную логическую игру «Канадские шашки».

Ключевые слова: компьютерная игра, шашки, канадские шашки, python.

Всего листов 46

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 Канадские шашки

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 22/2402 10 ТЗ-03

Листов 5

Исполнитель:

студент гр. ИСТбд-21

Никонова Дарья Александровна

« » 2023 г.

2023

**Введение**

Разрабатываемое приложение наименования «Канадские шашки» реализует игру канадские шашки. Канадские шашки – вариант игры в “Шашки” на доске 12х12 с 30 шашками. Игра предназначена для двух игроков. Игроки поочередно перемещают шашки на игровом поле. Первый игрок использует шашки белого цвета, а второй игрок - черного. Цель – побить все шашки соперника или заблокировать ходы его шашек.

**1. Основания для разработки**

Основание 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**

**Функциональное назначение**

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

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

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

Приложение должно соответствовать следующим правилам игры.

**Поле и игроки.** Игра ведется между двумя соперниками (пользователь-компьютер) шашками разного цвета (белые и чёрные) на квадратном поле, размером 12х12 клеток. У каждого игрока в начальной позиции находится 30 шашек, расположенных на темных клетках первых пяти горизонтальных рядов с каждой стороны поля.

**Типы ходов.**

Существует три типа ходов:

1. Перемещение простой шашки по диагонали вперед на одну клетку.
2. "Перепрыгивание" шашкой через занятую шашкой клетку, если за ней есть свободная клетка для посадки. При этом сбивается шашка соперника. Если после перемещения есть возможность продолжить "перепрыгивание", игрок может совершить еще один ход и так далее. Бить обязательно. При возможности нескольких вариантов взятия обязательно бить максимально возможное количество шашек, если остаются еще варианты, то можно выбрать любой из них. Запрещается дважды брать одну и ту же шашку, при этом можно пересекать дважды одно и то же пустое поле.
3. Ход дамкой. При достижении любого поля последней горизонтали, простая шашка превращается в дамку. Дамка ходит по диагонали на любое свободное поле как вперёд, так и назад. При этом также возможно побить шашку соперника. Если простая шашка в процессе взятия достигает дамочного поля и может бить дальше в роли простой шашки, то она этим ходом продолжает бой и остаётся простой. Иначе она превращается в дамку и останавливается. Право боя по правилам дамки она приобретает лишь со следующего хода.

**Порядок ходов.** Ходы в игре происходят поочередно. Первым ходит игрок с белыми шашками - пользователь, а затем ходы передаются игроку с черными - компьютеру. Во время хода игрок может переместить только одну свою шашку.

**Цель игры.**

Цель игры заключается в побитии всех шашек соперника или блокировке ходов его шашек.

**Завершение игры.**

Игра завершается, когда:

1. Один из игроков побьет все шашки противника – игрок победил;
2. Игрок не может сделать следующий ход, так как его шашки заперты – игрок проиграл;
3. Оба игрока не могут сделать ход, так как их шашки заперты - ничья;
4. Если один из игроков сдастся – игрок проиграл;

2.2.2 Требования к составу функций приложения

В приложении должны быть реализованы в графическом режиме следующие основные функции:

- регистрация/авторизация пользователя;

- отрисовка игрового поля;

- взаимодействие с пользователем ;

- интерактивные прием, проверка правильности и отрисовка хода пользователя;

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода компьютера;

- информирование пользователя об окончании игры и победителе.

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

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

**2.3 Требования к надёжности**

Приложение должно быть стабильным и работоспособным, не вызывать сбоев или ошибок. В случае сбоя или ошибки, приложение должно быть способно восстановиться без потери данных.

**2.4 Требования к информационной и программной совместимости**

Операционная система: Windows 11 Домашняя.

Версия языка программирования: Python 3.10.6.

Среда разработки: VS code - 1.83.1.

При создании программы используются следующие библиотеки:

tkinter, os, random, time, copy.

**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

Проект будет храниться в репозитории на сайте github.com по ссылке https://github.com/danikonova/CourseProject

2.6. 3 Сроки хранения

Срок хранения – до июля 2023 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 Канадские шашки**

**Пояснительная записка**

Р.02069337. 22/2402 10 ПЗ-01

Листов 6

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Исполнитель:

студент гр. ИСТбд-21

Никонова Дарья Александровна

« » 2023 г.

2023

**Введение**

Программа представляет из себя однопользовательское десктопное приложение по игре в канадские шашки с графическим интерфейсом в среде Windows.

В приложении реализованы в графическом режиме следующие функции:

* регистрация/авторизация пользователя;
* отрисовка игрового поля;
* взаимодействие с пользователем ;
* интерактивные прием, проверка правильности и отрисовка хода пользователя;
* проверка окончания игры;
* вычисление, проверка правильности и отрисовка хода компьютера;
* информирование пользователя об окончании игры и победителе.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

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

**1.2 Математические методы**

Модель игрового поля (матрица расположения шашек):

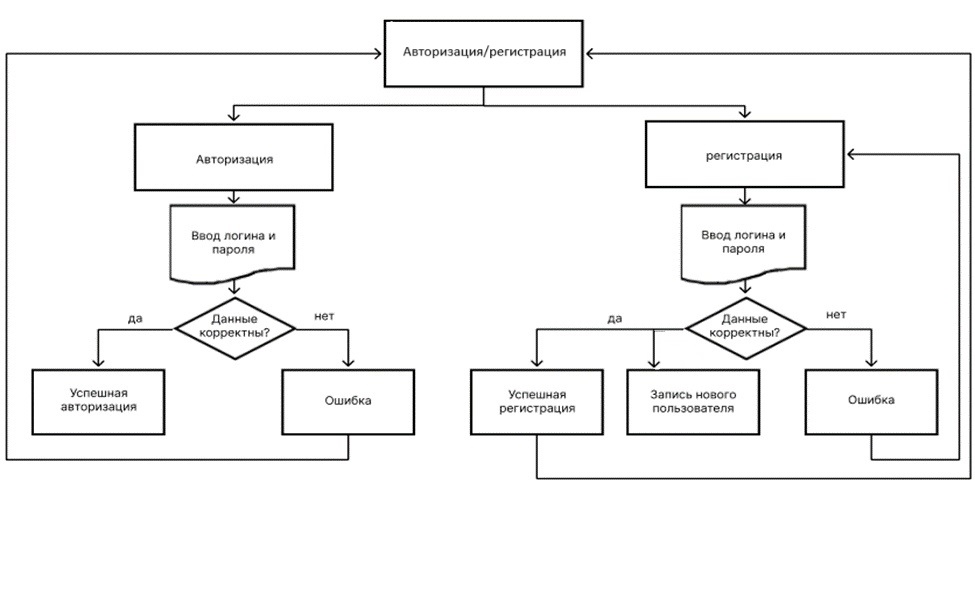
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 |
| 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 |
| 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

**1.3 Архитектура и алгоритмы**

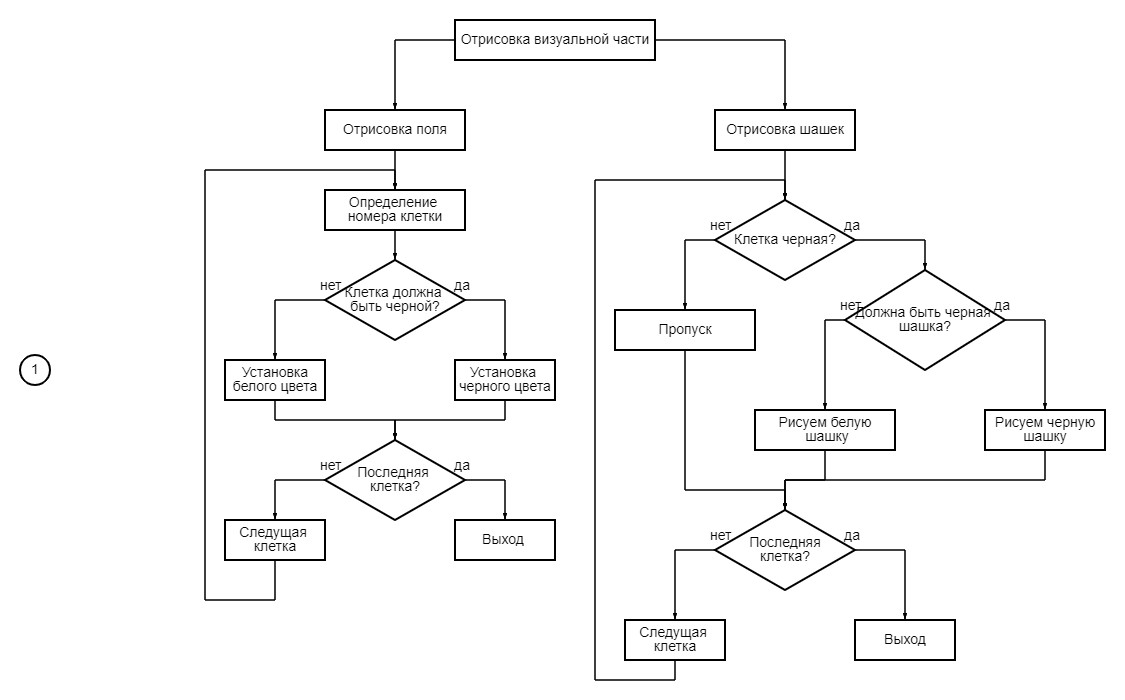
1.3.1. Архитектура



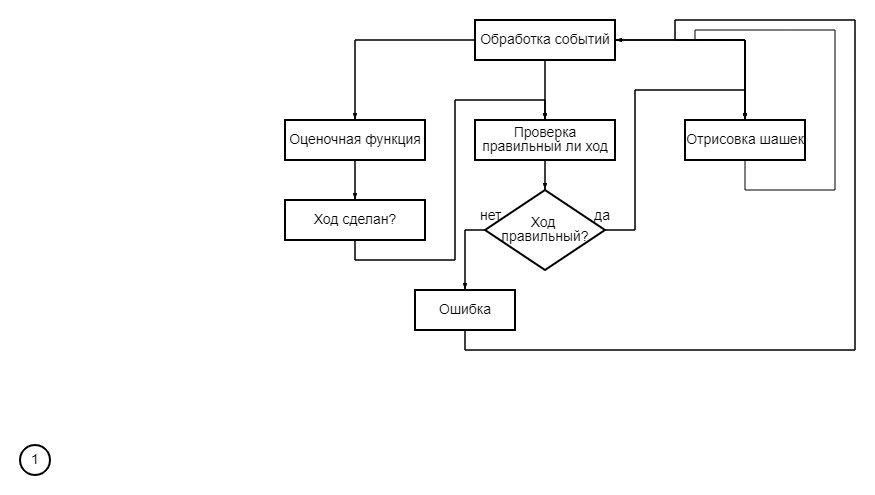
1.3.2. Алгоритм авторизации/регистрации



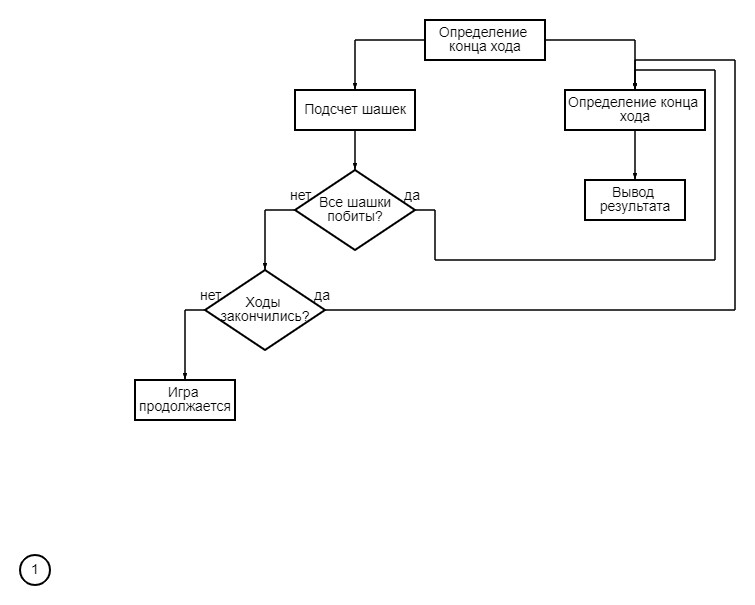
1.3.2 Алгоритм отрисовки



1.3.2 Алгоритм обработки событий



1.3.2 Алгоритм определение конца хода



**1.4 Тестирование**

1.4.1 Описание отчета о тестировании

В данном отчете представлены результаты тестирования программы на основе статического тестирования документации и программного кода. 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 ошибок. По результатам тестирования следует исправление выявленных ошибок.

1.4.3 Методика тестирования

Тестирование проводилось с использованием следующих методов:

* Статическое тестирование: анализ и проверка кода без его запуска, выявление ошибок в технической документации.
* Ручное тестирование: запуск пользовательских сценариев программы с различными входными данными и проверка корректности полученных результатов.

1.4.4 Проведенные тесты

В ходе тестирования были проведены следующие тесты:

1) Статическое тестирование

Количество обнаруженных и исправленных ошибок в документации: 2.

Количество обнаруженных и исправленных ошибок в программном коде: 5.

2) Ручное тестирование.

Проведены следующие тест-кейсы:

* Попытка авторизоваться при отсутствии файла с базы пользователей.
* Попытка зарегистрироваться с незаполненным полем «Логин».
* Попытка зарегистрироваться с логином user1 и паролем 123456.
* Попытка зарегистрироваться с логином user2 и незаполненным полем «Пароль».
* Попытка зарегистрироваться с логином user2 и паролем password.
* Попытка авторизоваться с логином user1 и произвольным паролем.
* Попытка авторизоваться с логином user1 и паролем 123456.
* Непосредственная игра в шашки с компьютером.

1.4.5 Вывод

На основе проведенных тестов сделаны следующие выводы:

Программа успешно прошла все тесты и работает корректно.

Обнаружены и исправлены следующие дефекты: некорректное шифрование паролей, переход к игре без авторизации.

**2. Источники, использованные при разработке**

1. Канадские шашки. – URL: [https://ru.wikipedia.org](https://ru.wikipedia.org/)
2. Руководство по Tkinter. – URL: <https://metanit.com/python/tkinter/>
3. Шишкин, В.В. Разработка логических компьютерных игр с

графическим интерфейсом в среде питон / В.В. Шишкин, Д.С.

Афонин. - Ульяновск: УлГТУ, 2023. – 89 с. – URL: chrome-

extension://efaidnbmnnnibpcajpcglclefindmkaj/http://lib.ulstu.ru/venec/di

sk/2023/112.pdf. – Текст: электронный

1. The Python Standart Library. – URL: [https://](https://pygame.ru/blog)docs.python.org/3.11/library

**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 Канадские шашки**

**Руководство программиста**

Р.02069337. 22/2402 10 РП-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 7

Исполнитель:

студент гр. ИСТбд-21

Никонова Дарья Александровна

« » 2023 г.

2023

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Однопользовательское десктопное приложение по игре с графическим интерфейсом в среде Windows.

**1.2 Условия, необходимые для использования приложения**

Версия операционной системы: Windows 7, 8, 8.1, 10, 11

Язык программирования: Python

Среда разработки: PyCharm Community Edition 2023.1.1

Используемые библиотеки: графическая библиотека Tk

**2. Характеристики программы**

**2.1 Характеристики приложения**

Программа состоит из файла с исходным кодом на языке Python **CC.py**. В качестве ресурсов используются графические файлы **b.png, bq.png, w.png, wq.png**. В процессе работы создаётся текстовый файл для регистрации пользователей **registry.txt**. В том числе при сохранении паролей применяется шифрование.

При реализации графического интерфейса программа создаёт основное окно для авторизации/регистрации пользователя. При успешной авторизации создаётся новое окно с игровой доской и расставленными шашками в соответствии с правилами канадских шашек. Согласно техническому заданию, игрок играет белыми фигурами и ходит первым. Игровое взаимодействие с пользователем происходит с помощью компьютерной мыши. В программе реализовано меню с пунктами *Регистрация, Сменить пользователя, Играть, Выход.*

Смена режимов авторизации и регистрации пользователей происходит с помощью смены фреймов в базовом окне.

В отдельных процедурах **encrypt** и **decrypt** реализован алгоритм шифрования методом Цезаря. Для авторизации и регистрации пользователей используются классы **Login\_page** и **Sign\_up\_page**, в которых реализованы методы проверки правильности входных данных и сокрытия пароля.

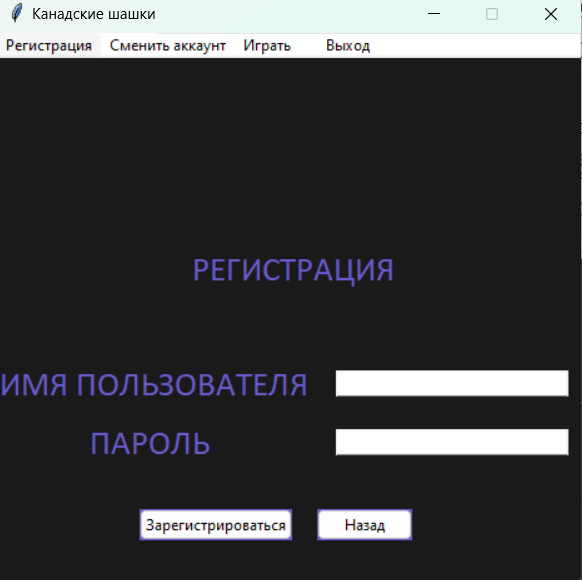
Непосредственно игровое взаимодействие происходит в отдельном окне, реализованном с помощью класса **Game\_page**, в котором реализовано

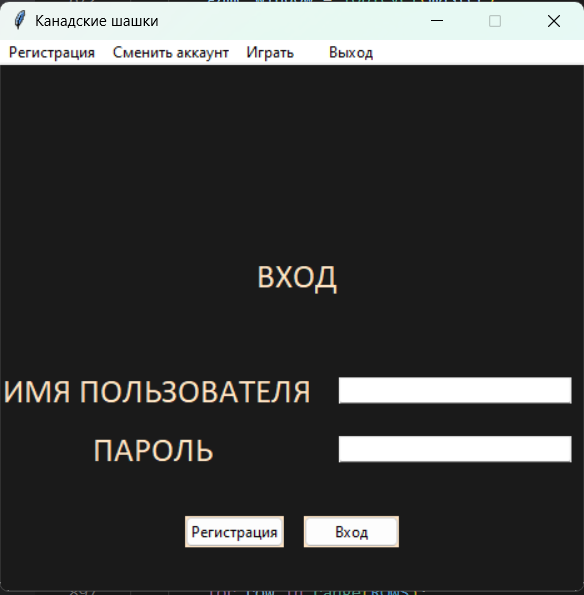
* взаимодействие с пользователем с помощью обработки событий (*event*) перемещения мыши и нажатия на левую кнопку мыши методами **mouse\_pos\_from** и **mouse\_pos\_to;**
* отрисовка игрового поля и шашек в методе **draw\_board;**
* логика игры в методах оценки возможности хода, обработки обязательных ходов, выбора оптимальной стратегии и оценки окончания игры: **comp\_moves\_list, player\_moves\_list, check\_comp\_move, check\_player\_move, check, check\_comp\_mandatory\_moves, check\_player\_mandatory\_moves, check\_comp\_moves, check\_player\_moves, check\_comp\_left\_moves, check\_player\_left\_moves, turn.**

Загрузка изображений шашек производится в отдельном методе **load\_images.**

**2. 2. Описание интерфейса программы**

1. Окно регистрации и авторизации

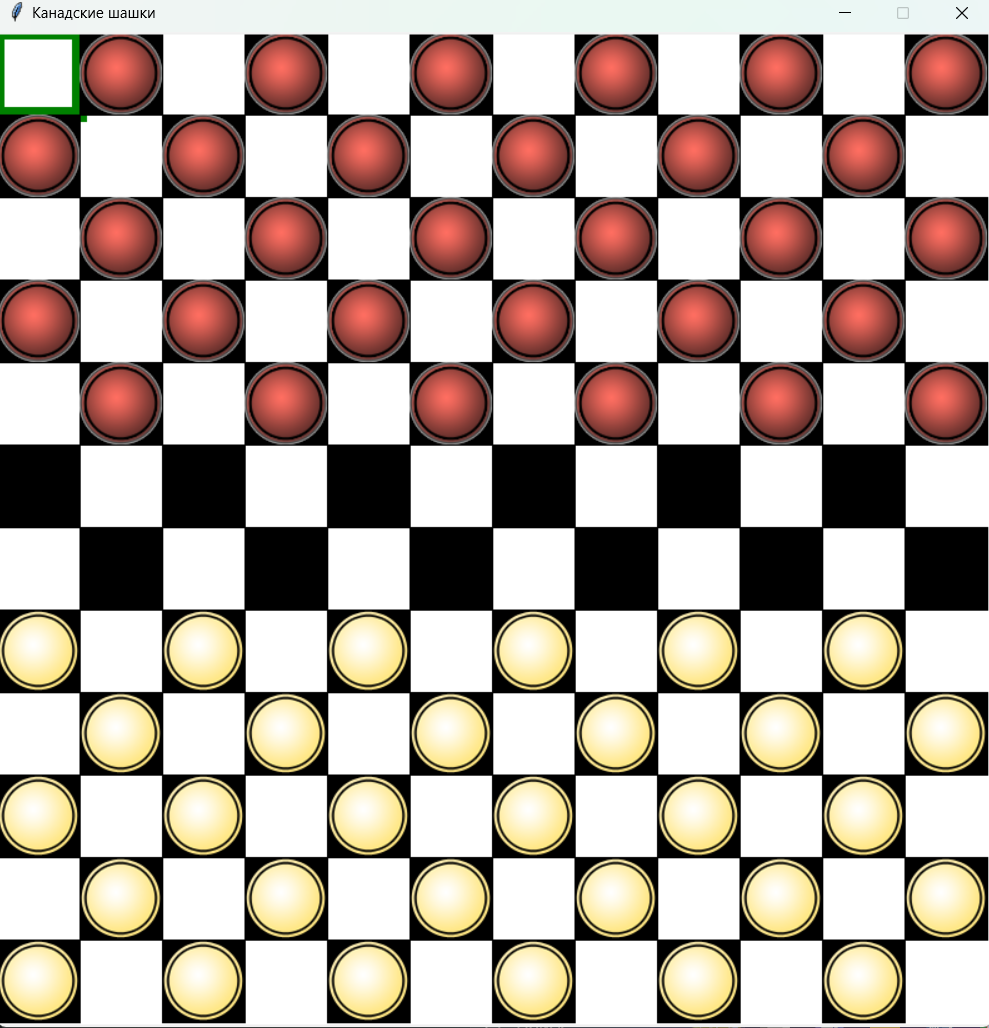




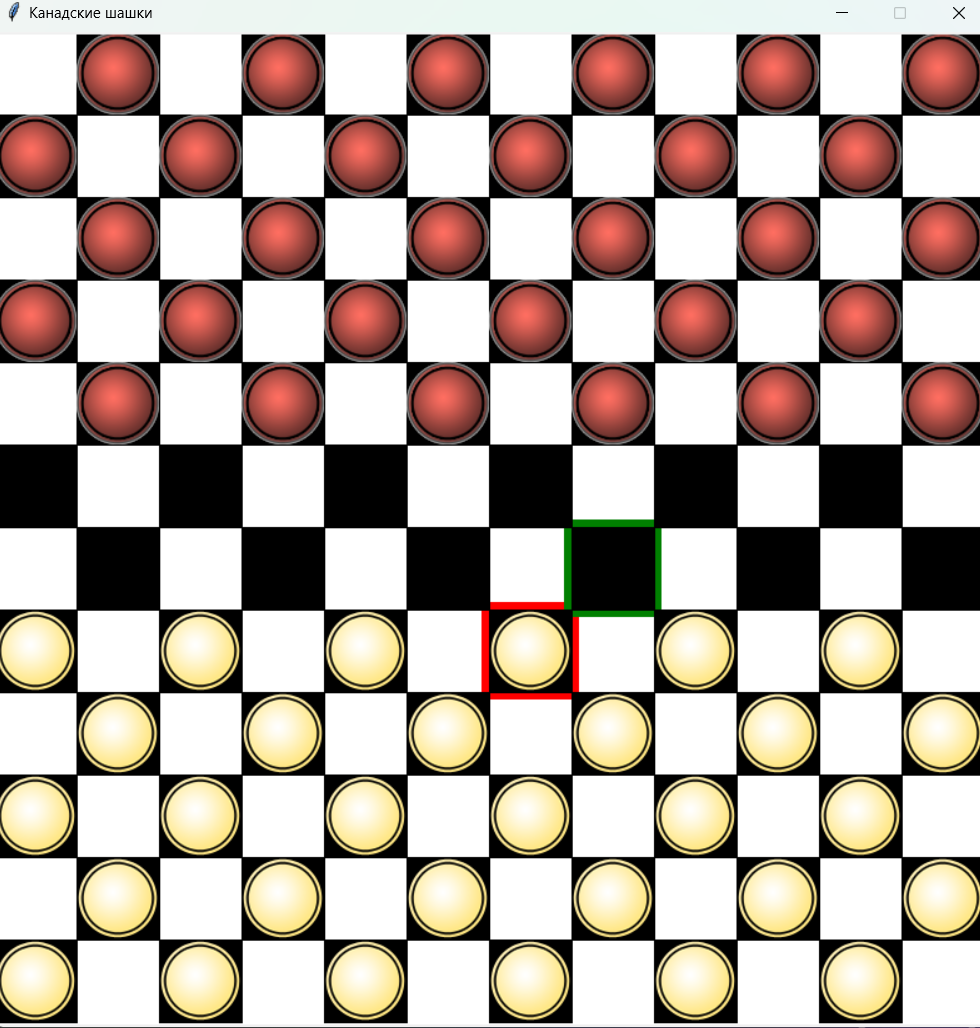
Для начала игры необходимо зарегистрироваться и авторизоваться Если введены не корректные данные, возникнет сообщение об ошибке. При введении корректных данных возникнет сообщение об успешной регистрации. При вводе корректных данных появится новое окно с сообщением об успешной авторизации:

2. Игра

После авторизации в новом окне запускается игра: происходит отрисовка доски и начальная расстановка шашек.



При наведении указателя мыши на клетку игрового поля, клетка подсвечивается зелёной рамкой. При нажатии на шашку клетка с шашкой подсвечивается красным и появляется возможность выбрать клетку для хода.



Если ход выбранной шашкой в заданную клетку возможен, то шашка передвигается и ход передаётся компьютеру. Выбранная ранее шашка «выключается». Если на пути хода была шашка, то она убирается с поля.

При съедении шашки, если есть ещё один ход, при котором можно съесть, ход продолжается.

При достижении шашки другого края поля, она превращается в дамку.

При «съедении» всех шашек одного из игроков или невозможности выполнить ход игра заканчивается. После этого подсчитываются очки и выводится результат партии.

**2.4. Средства контроля ввода/вывода:**

* Проверка на пустое поле
* Проверка на авторизацию

**2.5 Особенности реализации приложения**

Массив поля:

scene = [[0,3,0,3,0,3,0,3,0,3,0,3],  
 [3,0,3,0,3,0,3,0,3,0,3,0],  
 [0,3,0,3,0,3,0,3,0,3,0,3],  
 [3,0,3,0,3,0,3,0,3,0,3,0],  
 [0,3,0,3,0,3,0,3,0,3,0,3],  
 [0,0,0,0,0,0,0,0,0,0,0,0],  
 [0,0,0,0,0,0,0,0,0,0,0,0],

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

**3. Обращение к программе**

Tkinter – это кросс-платформенная событийно-ориентированная графическая библиотека Python, предназначенная для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя (GUI).

Модуль tkinter.messagebox, предоставляющий доступ к готовым диалоговым окнам.

**4. Сообщения**

1) *«Ошибка ввода. Введите действительное имя пользователя»* в случае отсутствия имени пользователя при регистрации.

2) *«Ошибка. Имя пользователя или пароль введены неверно»* в случае отсутствия указанной учётной записи или неверного ввода пароля.

3) *«Аккаунт был успешно добавлен. Пожалуйста, войдите в систему»* в случае успешной регистрации нового пользователя.

4) *«Давайте играть! Приветствую»* в случае успешной авторизации пользователя.

5) *«Игра завершена. Вы победили! Нажми <Да>, чтобы начать заново»* в случае окончании игры при условии выигрыша пользователя.

6) *«Игра завершена. Вы проиграли! Нажми <Да>, чтобы начать заново»* в случае окончании игры при условии проигрыша пользователя.

7) *«Игра завершена. Ходов больше нет. Нажми <Да>, чтобы начать заново»* в случае окончании игры при условии ничейной ситуации на доске.

**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 Канадские шашки**

**Текст программы**

Р.02069337. 22/2402 10 ТП-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 25

Исполнитель:

студент гр. ИСТбд-21

Никонова Дарья Александровна

« » 2023 г.

2023

import os

from tkinter import \*

import random

import time

import copy

from tkinter import messagebox, ttk

# ---------------------------------------- Глобально ---------------------------------

fa = None

file\_name = 'registry.txt'

bg\_color = 'gray10'

fg\_color = 'bisque'

menu\_fg = "snow"

def load\_images():  # загружаем изображения шашек

global pieces

img1 = PhotoImage(file="w.png")

img2 = PhotoImage(file="wq.png")

img3 = PhotoImage(file="b.png")

img4 = PhotoImage(file="bq.png")

pieces = [0, img1, img2, img3, img4]

# ------------------------------------------------ Шифрование ----------------------

def cesar(txt):

code = 0

for ch in txt:

code += ord(ch)

code += 128

return code

def encrypt(txt, code):

result = ""

for c in txt:

result += chr(ord(c) + code)

return result

def decrypt(txt, code):

result = ""

for c in txt:

result += chr(ord(c) - code)

return result

current\_user = None

# ---------------------------------------- Игра -----------------------------------------

WIDTH = 792

HEIGHT = 792

ROWS = 12

COLUMNS = 12

mx = 66

my = 66

# ------------------------------------------------ GUI ----------------------------------

class SampleApp(Tk):

    def \_\_init\_\_(self):

        Tk.\_\_init\_\_(self)

        self.title('Канадские шашки')

        # self.geometry(str(WIDTH) + 'x' + str(HEIGHT))

        self.resizable(0, 0)

        self.configure(bg=bg\_color)

        global user, password

        user = StringVar()

        password = StringVar()

        user.set('a')

        menubar = MenuBar(self)

        self.config(menu=menubar)

        self.\_frame = None

        self.switch\_frame(Login\_page)

    def switch\_frame(self, frame\_class):

        global new\_frame

        new\_frame = frame\_class(self)

        if self.\_frame is not None:

            self.\_frame.destroy()

        self.\_frame = new\_frame

        self.\_frame.pack(anchor='center')

    def switch\_window(self, window\_class):

        global new\_window

        new\_window = window\_class(self)

        # self.withdraw()

# ------------------------------------------------ Меню -------------------------------

class MenuBar(Menu):

    def \_\_init\_\_(self, parent):

        Menu.\_\_init\_\_(self, parent)

        self.add\_cascade(label="Регистрация", command=lambda: parent.switch\_frame(Sign\_up\_page))

        self.add\_cascade(label="Сменить аккаунт", command=lambda: parent.switch\_frame(Login\_page))

        self.add\_cascade(label="Играть", command=lambda: parent.switch\_window(Game\_page))

        self.add\_separator()

        self.add\_cascade(label="Выход", command=lambda: parent.destroy())

# ---------------------------------------------- Логин ---------------------------------

class Login\_page(Frame, Menu):

    def \_\_init\_\_(self, master):

        Frame.\_\_init\_\_(self, master)

        self.config(bg=bg\_color)

        # self.pack(expand=True)

        global fa

        fa = Login\_page

        style1 = ttk.Style()

        style1.configure('TLabel', background=bg\_color, foreground=fg\_color, font='Calibri 20')

        global user, password, current\_user

        user = StringVar()

        password = StringVar()

        frame1 = Frame(self, bg=bg\_color)

        frame1.grid(row=0, column=0)

        ttk.Label(frame1, text='ВХОД', style='login.TLabel').grid(row=0, column=1, pady=(150, 5), padx=(8, 0))

        frame3 = Frame(self, bg=bg\_color)

        frame3.grid(row=1, column=0)

        ttk.Label(frame3, text='ИМЯ ПОЛЬЗОВАТЕЛЯ', style='TLabel').grid(row=1, column=0, pady=(50, 10))

        ttk.Label(frame3, text='ПАРОЛЬ ', style='TLabel').grid(row=2, column=0, )

        ttk.Entry(frame3, textvariable=user, width=30).grid(row=1, column=2, pady=(50, 10), padx=(20, 10))

        p = ttk.Entry(frame3, textvariable=password, width=30)

        p.grid(row=2, column=2, padx=(12, 2))

        p.config(show='\*')

        self.x = 1

        style = ttk.Style()

        style.configure('a.TButton', background=fg\_color, borderwidth=35)

        frame2 = Frame(self, bg=bg\_color)

        frame2.grid(row=2, column=0, pady=35)

        ttk.Button(frame2, text='Регистрация', command=lambda: master.switch\_frame(Sign\_up\_page),

                   style='a.TButton').pack(side=LEFT, padx=8)

        ttk.Button(frame2, text='Вход', command=lambda: self.enter(master), style='a.TButton').pack(side=RIGHT, padx=8)

    def enter(self, master):

        global current\_user

        if os.path.isfile(file\_name):

            with open(file\_name, 'r', encoding='utf-16') as f:

                data = f.readlines()

                flag = False

                for line in data:

                    if line[0:line.find(' ')] == user.get():

                        p = decrypt(line[line.find(' ') + 1:len(line) - 1], cesar(user.get()))

                        if p == password.get():

                            current\_user = user.get()

                            flag = True

                        break

            if flag:

                messagebox.showinfo("Давайте играть!", f"Приветствую, {current\_user}!")

                master.switch\_window(Game\_page)

            else:

                messagebox.showerror("Ошибка", "Имя пользователя или пароль введены неверно")

                user.set('')

                password.set('')

        else:

            messagebox.showerror("Ошибка", "Имя пользователя или пароль введены неверно")

            user.set('')

            password.set('')

class Sign\_up\_page(Frame):

    def \_\_init\_\_(self, master):

        Frame.\_\_init\_\_(self, master)

        Frame.configure(self, bg=bg\_color)

        global fa

        fa = Sign\_up\_page

        global newuser, newpassword

        newuser = StringVar()

        newpassword = StringVar()

        frame1 = Frame(self, bg=bg\_color)

        frame1.grid(row=0, column=0)

        # frame = Frame(self, bg=bg\_color).grid(row=0, column=2)

        style = ttk.Style()

        style.configure('TButton', background='slateblue', borderwidth=30)

        Style1 = ttk.Style()

        Style1.configure('TLabel', background=bg\_color, foreground='slateblue', font='Calibri 20')

        ttk.Label(frame1, text='РЕГИСТРАЦИЯ', style='login.TLabel').grid(row=0, column=1, pady=(150, 5), padx=(8, 0))

        frame3 = Frame(self, bg=bg\_color)

        frame3.grid(row=1, column=0)

        ttk.Label(frame3, text='ИМЯ ПОЛЬЗОВАТЕЛЯ', style='TLabel').grid(row=1, column=0, pady=(50, 10))

        ttk.Label(frame3, text='ПАРОЛЬ ', style='TLabel').grid(row=2, column=0)

        ttk.Entry(frame3, textvariable=newuser, width=30).grid(row=1, column=2, pady=(50, 10), padx=(20, 10))

        p = ttk.Entry(frame3, textvariable=newpassword, width=30)

        p.grid(row=2, column=2, padx=(20, 10))

        p.config(show="")

        self.x = 2

        frame2 = Frame(self, bg=bg\_color)

        frame2.grid(row=2, column=0, pady=35)

        ttk.Button(frame2,

                   text='Зарегистрироваться',

                   command=lambda: self.sign\_up(master),

                   style='TButton').grid(row=0, column=0)

        (ttk.Button(frame2, text='Назад',

                   command=lambda: master.switch\_frame(Login\_page),

                   style='TButton').grid(row=0, column=1, padx=20))

    def sign\_up(self, master):

        new\_user = newuser.get()

        if new\_user == '':

            messagebox.showerror('Ошибка ввода', 'Введите действительное имя пользователя')

        else:

            if newpassword.get() == '':

                messagebox.showerror('Ошибка ввода', 'Поле ввода пароля не может быть пустым.')

            else:

                with open(file\_name, 'r', encoding='utf-16') as f:

                    existing\_users = [line.split()[0] for line in f]

                if new\_user in existing\_users:

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

                else:

                    with open(file\_name, 'a', encoding='utf-16') as f:

                        res = new\_user + ' ' + encrypt(newpassword.get(), cesar(new\_user)) + '\n'

                        f.write(res)

                    messagebox.showinfo('Успешно', 'Аккаунт был успешно добавлен\nПожалуйста, войдите в систему')

                    master.switch\_frame(Login\_page)

# --------------------------------------- Игра ------------------------------------------

comp\_moves\_f\_list = ()  # конечный список ходов компьютера

deep\_mind = 2  # количество предсказываемых компьютером ходов

k\_res = 0

o\_res = 0

pos1\_x = -1  # клетка не задана

def on\_close(window):

    # window.deiconify()

    window.destroy()

class Game\_page():

    moveable = True  # определение хода игрока(да)

    def \_\_init\_\_(self, master):

        game\_window = Toplevel(master)

        # Frame.\_\_init\_\_(self, master)

        # Frame.configure(self, bg=bg\_color)

        game\_window.title("Канадские шашки")

        game\_window.resizable(0, 0)

        game\_window.wm\_attributes("-topmost", 1)

        game\_window.protocol("WM\_DELETE\_WINDOW", lambda window=game\_window: on\_close(window))

        # global fa

        # fa = Game\_page

        global board

        board = Canvas(game\_window, width=WIDTH, height=HEIGHT, bg="white")

        board.pack()

        load\_images()  # здесь загружаем изображения шашек

        self.new\_game()  # начинаем новую игру

        self.draw\_board(-1, -1, -1, -1)  # рисуем игровое поле

        board.bind("<Motion>", self.mouse\_pos\_from)  # движение мышки по полю

        board.bind("<Button-1>", self.mouse\_pos\_to)  # нажатие левой кнопки

    def new\_game(self):  # начинаем новую игру

        global scene

        scene = [[0 for \_ in range(ROWS)] for \_ in range(COLUMNS)]

        for row in range(ROWS):

            for col in range(COLUMNS):

                if row < (ROWS - 2) / 2 and (row + col) % 2 == 1:

                    scene[row][col] = 3

                if row >= (ROWS + 2) / 2 and (row + col) % 2 == 1:

                    scene[row][col] = 1

    def draw\_board(self, x\_pos\_1, y\_pos\_1, x\_pos\_2, y\_pos\_2):  # рисуем игровое поле

        global pieces

        global scene

        global red\_frame, green\_frame

        k = WIDTH / ROWS

        x = 0

        board.delete('all')

        red\_frame = board.create\_rectangle(-5, -5, -5, -5, outline="red", width=12)

        green\_frame = board.create\_rectangle(-5, -5, -5, -5, outline="green", width=12)

        while x < WIDTH:  # рисуем доску

            y = k

            while y < HEIGHT:

                board.create\_rectangle(x, y, x + k, y + k, fill="black")

                y += 2 \* k

            x += 2 \* k

        x = k

        while x < WIDTH:  # рисуем доску

            y = 0

            while y < HEIGHT:

                board.create\_rectangle(x, y, x + k, y + k, fill="black")

                y += 2 \* k

            x += 2 \* k

        for y in range(ROWS):  # рисуем стоячие шашки

            for x in range(COLUMNS):

                z = scene[y][x]

                if z:

                    if (x\_pos\_1, y\_pos\_1) != (x, y):  # стоячие шашки?

                        board.create\_image(x \* k, y \* k, anchor=NW, image=pieces[z])

        # рисуем активную шашку

        z = scene[y\_pos\_1][x\_pos\_1]

        if z:  # ???

            board.create\_image(x\_pos\_1 \* k, y\_pos\_1 \* k, anchor=NW, image=pieces[z], tag='ani')

        # вычисление коэф. для анимации

        kx = 1 if x\_pos\_1 < x\_pos\_2 else -1

        ky = 1 if y\_pos\_1 < y\_pos\_2 else -1

        for i in range(abs(x\_pos\_1 - x\_pos\_2)):  # анимация перемещения шашки

            for j in range(33):

                board.move('ani', 0.03 \* k \* kx, 0.03 \* k \* ky)

                board.update()  # обновление

                time.sleep(0.01)

    def message(self, s):

        z = 'Игра завершена'

        if s == 1:

            i = messagebox.askyesno(title=z, message='Вы победили!\nНажми "Да" что бы начать заново.', icon='info')

        if s == 2:

            i = messagebox.askyesno(title=z, message='Вы проиграли!\nНажми "Да" что бы начать заново.', icon='info')

        if s == 3:

            i = messagebox.askyesno(title=z, message='Ходов больше нет.\nНажми "Да" что бы начать заново.', icon='info')

        self.new\_game()

        self.draw\_board(-1, -1, -1, -1)  # рисуем игровое поле

        self.moveable = True  # ход игрока доступен

    def mouse\_pos\_from(self, event):  # выбор клетки для хода 1

        x, y = event.x // mx, event.y // my  # вычисляем координаты клетки

        board.coords(green\_frame, x \* mx, y \* my, x \* mx + mx, y \* my + my)  # рамка в выбранной клетке

    def mouse\_pos\_to(self, event):  # выбор клетки для хода 2

        global pos1\_x, pos1\_y, pos2\_x, pos2\_y

        x, y = event.x // mx, event.y // my  # вычисляем координаты клетки

        if scene[y][x] == 1 or scene[y][x] == 2:  # проверяем шашку игрока в выбранной клетке

            board.coords(red\_frame, x \* mx, y \* my, x \* mx + mx, y \* my + my)  # рамка в выбранной клетке

            pos1\_x, pos1\_y = x, y

        else:

            if pos1\_x != -1:  # клетка выбрана

                pos2\_x, pos2\_y = x, y

                if self.moveable:  # ход игрока?

                    self.player\_move()

                    if not self.moveable:

                        time.sleep(0.5)

                        self.comp\_move()  # передаём ход компьютеру

                pos1\_x = -1  # клетка не выбрана

                board.coords(red\_frame, -5, -5, -5, -5)  # рамка вне поля

    def comp\_move(self):  # !!!

        global comp\_moves\_f\_list

        self.check\_comp\_move(1, (), [])

        if comp\_moves\_f\_list:  # проверяем наличие доступных ходов

            count\_moves = len(comp\_moves\_f\_list)  # количество ходов

            t\_move = random.randint(0, count\_moves - 1)  # случайный ход

            len\_move = len(comp\_moves\_f\_list[t\_move])  # длина хода

            for i in range(len\_move - 1):

                # выполняем ход

                self.turn(1, comp\_moves\_f\_list[t\_move][i][0], comp\_moves\_f\_list[t\_move][i][1],

                     comp\_moves\_f\_list[t\_move][1 + i][0], comp\_moves\_f\_list[t\_move][1 + i][1])

            comp\_moves\_f\_list = []  # очищаем список ходов

            self.moveable = True  # ход игрока доступен

        # определяем победителя

        comp\_scores, player\_scores = self.check()

        if not player\_scores:

            self.message(2)

        elif not comp\_scores:

            self.message(1)

        elif self.moveable and not self.player\_moves\_list():

            self.message(3)

        elif not self.moveable and not self.comp\_moves\_list():

            self.message(3)

    def comp\_moves\_list(self):  # составляем список ходов компьютера

        moves\_list = self.check\_comp\_mandatory\_moves([])  # здесь проверяем обязательные ходы

        if not moves\_list:

            moves\_list = self.check\_left\_comp\_moves([])  # здесь проверяем оставшиеся ходы

        return moves\_list

    def check\_comp\_move(self, tur, n\_list, moves\_list):  # !!!

        global scene

        global comp\_moves\_f\_list

        global best\_res, k\_res, o\_res

        if not moves\_list:  # если список ходов пустой...

            moves\_list = self.comp\_moves\_list()  # заполняем

        if moves\_list:

            k\_scene = copy.deepcopy(scene)  # копируем поле

            for ((pos1\_x, pos1\_y), (pos2\_x, pos2\_y)) in moves\_list:  # проходим все ходы по списку

                t\_list = self.turn(0, pos1\_x, pos1\_y, pos2\_x, pos2\_y)

                if t\_list:  # если существует ещё ход

                    self.check\_comp\_move(tur, (n\_list + ((pos1\_x, pos1\_y),)), t\_list)

                else:

                    self.check\_player\_move(tur, [])

                    if tur == 1:

                        t\_res = o\_res / k\_res

                        if not (comp\_moves\_f\_list):  # записыаем если пустой

                            comp\_moves\_f\_list = (n\_list + ((pos1\_x, pos1\_y), (pos2\_x, pos2\_y)),)

                            best\_res = t\_res  # сохряняем наилучший результат

                        else:

                            if t\_res == best\_res:

                                comp\_moves\_f\_list = comp\_moves\_f\_list + (n\_list + ((pos1\_x, pos1\_y), (pos2\_x, pos2\_y)),)

                            if t\_res > best\_res:

                                comp\_moves\_f\_list = ()

                                comp\_moves\_f\_list = (n\_list + ((pos1\_x, pos1\_y), (pos2\_x, pos2\_y)),)

                                best\_res = t\_res  # сохряняем наилучший результат

                        o\_res = 0

                        k\_res = 0

                scene = copy.deepcopy(k\_scene)  # возвращаем поле

        else:  # ???

            comp\_scores, player\_scores = self.check()  # подсчёт результата хода

            o\_res += (comp\_scores - player\_scores)

            k\_res += 1

    def player\_moves\_list(self):  # составляем список ходов игрока

        moves\_list = self.check\_player\_mandatory\_moves([])  # здесь проверяем обязательные ходы

        if not moves\_list:

            moves\_list = self.check\_left\_player\_moves([])  # здесь проверяем оставшиеся ходы

        return moves\_list

    def check\_player\_move(self, tur, moves\_list):

        global scene, k\_res, o\_res

        global deep\_mind

        if not moves\_list:

            moves\_list = self.player\_moves\_list()

        if moves\_list:  # проверяем наличие доступных ходов

            k\_scene = copy.deepcopy(scene)  # копируем поле

            for ((pos1\_x, pos1\_y), (pos2\_x, pos2\_y)) in moves\_list:

                t\_list = self.turn(0, pos1\_x, pos1\_y, pos2\_x, pos2\_y)

                if t\_list:  # если существует ещё ход

                    self.check\_player\_move(tur, t\_list)

                else:

                    if tur < deep\_mind:

                        self.check\_comp\_move(tur + 1, (), [])

                    else:

                        comp\_scores, player\_scores = self.check()  # подсчёт результата хода

                        o\_res += (comp\_scores - player\_scores)

                        k\_res += 1

                scene = copy.deepcopy(k\_scene)  # возвращаем поле

        else:  # доступных ходов нет

            comp\_scores, player\_scores = self.check()  # подсчёт результата хода

            o\_res += (comp\_scores - player\_scores)

            k\_res += 1

    def check(self):  # подсчёт шашек на поле

        global scene

        score\_player = 0

        score\_comp = 0

        for i in range(ROWS):

            for j in scene[i]:

                if j == 1: score\_player += 1

                if j == 2: score\_player += 3

                if j == 3: score\_comp += 1

                if j == 4: score\_comp += 3

        return score\_comp, score\_player

    def player\_move(self):

        global pos1\_x, pos1\_y, pos2\_x, pos2\_y

        self.moveable = False  # считаем ход игрока выполненным

        moves\_list = self.player\_moves\_list()

        if moves\_list:

            if ((pos1\_x, pos1\_y), (pos2\_x, pos2\_y)) in moves\_list:  # проверяем ход на соответствие правилам игры

                t\_list = self.turn(1, pos1\_x, pos1\_y, pos2\_x, pos2\_y)  # если всё хорошо, делаем ход

                if t\_list:  # если есть ещё ход той же шашкой

                    self.moveable = True  # считаем ход игрока невыполненным

            else:

                self.moveable = True  # считаем ход игрока невыполненным

        board.update()  # !!!обновление

    def turn(self, flag, pos1\_x, pos1\_y, pos2\_x, pos2\_y):

        global scene

        if flag: self.draw\_board(pos1\_x, pos1\_y, pos2\_x, pos2\_y)  # рисуем игровое поле

        # превращение

        if pos2\_y == 0 and scene[pos1\_y][pos1\_x] == 1:

            scene[pos1\_y][pos1\_x] = 2

        # превращение

        if pos2\_y == ROWS - 1 and scene[pos1\_y][pos1\_x] == 3:

            scene[pos1\_y][pos1\_x] = 4

        # делаем ход

        scene[pos2\_y][pos2\_x] = scene[pos1\_y][pos1\_x]

        scene[pos1\_y][pos1\_x] = 0

        # рубим шашку игрока

        kx = ky = 1

        if pos1\_x < pos2\_x: kx = -1

        if pos1\_y < pos2\_y: ky = -1

        x\_pos, y\_pos = pos2\_x, pos2\_y

        while (pos1\_x != x\_pos) or (pos1\_y != y\_pos):

            x\_pos += kx

            y\_pos += ky

            if scene[y\_pos][x\_pos] != 0:

                scene[y\_pos][x\_pos] = 0

                if flag: self.draw\_board(-1, -1, -1, -1)  # рисуем игровое поле

                # проверяем ход той же шашкой...

                if scene[pos2\_y][pos2\_x] == 3 or scene[pos2\_y][pos2\_x] == 4:  # ...компьютера

                    return self.check\_comp\_moves([], pos2\_x, pos2\_y)  # возвращаем список доступных ходов

                elif scene[pos2\_y][pos2\_x] == 1 or scene[pos2\_y][pos2\_x] == 2:  # ...игрока

                    return self.check\_player\_moves([], pos2\_x, pos2\_y)  # возвращаем список доступных ходов

        if flag: self.draw\_board(pos1\_x, pos1\_y, pos2\_x, pos2\_y)  # рисуем игровое поле

    def check\_comp\_mandatory\_moves(self, moves\_list):  # проверка наличия обязательных ходов

        for y in range(COLUMNS):  # сканируем всё поле

            for x in range(ROWS):

                moves\_list = self.check\_comp\_moves(moves\_list, x, y)

        return moves\_list

    def check\_comp\_moves(self, move\_list, x, y):

        if scene[y][x] == 3:  # шашка

            for ix, iy in (-1, -1), (-1, 1), (1, -1), (1, 1):

                if 0 <= y + iy + iy < COLUMNS and 0 <= x + ix + ix < ROWS:

                    if scene[y + iy][x + ix] == 1 or scene[y + iy][x + ix] == 2:

                        if scene[y + iy + iy][x + ix + ix] == 0:

                            move\_list.append(((x, y), (x + ix + ix, y + iy + iy)))  # запись хода в конец списка

        if scene[y][x] == 4:  # шашка с короной

            for ix, iy in (-1, -1), (-1, 1), (1, -1), (1, 1):

                isCorrect = 0  # определение правильности хода

                for i in range(1, COLUMNS):

                    if 0 <= y + iy \* i < COLUMNS and 0 <= x + ix \* i < ROWS:

                        if isCorrect == 1:

                            move\_list.append(((x, y), (x + ix \* i, y + iy \* i)))  # запись хода в конец списка

                        if scene[y + iy \* i][x + ix \* i] == 1 or scene[y + iy \* i][x + ix \* i] == 2:

                            isCorrect += 1

                        if scene[y + iy \* i][x + ix \* i] == 3 or scene[y + iy \* i][x + ix \* i] == 4 or isCorrect == 2:

                            if isCorrect > 0: move\_list.pop()  # удаление хода из списка

                            break

        return move\_list

    def check\_left\_comp\_moves(self, moves\_list):  # проверка наличия остальных ходов

        for y in range(COLUMNS):  # сканируем всё поле

            for x in range(ROWS):

                if scene[y][x] == 3:  # шашка

                    for ix, iy in (-1, 1), (1, 1):

                        if 0 <= y + iy < COLUMNS and 0 <= x + ix < ROWS:

                            if scene[y + iy][x + ix] == 0:

                                moves\_list.append(((x, y), (x + ix, y + iy)))  # запись хода в конец списка

                            if scene[y + iy][x + ix] == 1 or scene[y + iy][x + ix] == 2:

                                if 0 <= y + iy \* 2 < COLUMNS and 0 <= x + ix \* 2 < ROWS:

                                    if scene[y + iy \* 2][x + ix \* 2] == 0:

                                        moves\_list.append(((x, y),

                                                           (x + ix \* 2, y + iy \* 2)))  # запись хода в конец списка

                if scene[y][x] == 4:  # шашка с короной

                    for ix, iy in (-1, -1), (-1, 1), (1, -1), (1, 1):

                        isCorrect = 0  # определение правильности хода

                        for i in range(1, COLUMNS):

                            if 0 <= y + iy \* i < COLUMNS and 0 <= x + ix \* i < ROWS:

                                if scene[y + iy \* i][x + ix \* i] == 0:

                                    moves\_list.append(((x, y), (x + ix \* i, y + iy \* i)))  # запись хода в конец списка

                                if scene[y + iy \* i][x + ix \* i] == 1 or scene[y + iy \* i][x + ix \* i] == 2:

                                    isCorrect += 1

                                if scene[y + iy \* i][x + ix \* i] == 3 or scene[y + iy \* i][

                                    x + ix \* i] == 4 or isCorrect == 2:

                                    break

        return moves\_list

    def check\_player\_mandatory\_moves(self, moves\_list):  # проверка наличия обязательных ходов

        moves\_list = []  # список ходов

        for y in range(COLUMNS):  # сканируем всё поле

            for x in range(ROWS):

                moves\_list = self.check\_player\_moves(moves\_list, x, y)

        return moves\_list

    def check\_player\_moves(self, moves\_list, x, y):

        if scene[y][x] == 1:  # шашка

            for ix, iy in (-1, -1), (-1, 1), (1, -1), (1, 1):

                if 0 <= y + iy + iy < COLUMNS and 0 <= x + ix + ix < ROWS:

                    if scene[y + iy][x + ix] == 3 or scene[y + iy][x + ix] == 4:

                        if scene[y + iy + iy][x + ix + ix] == 0:

                            moves\_list.append(((x, y), (x + ix + ix, y + iy + iy)))  # запись хода в конец списка

        if scene[y][x] == 2:  # шашка с короной

            for ix, iy in (-1, -1), (-1, 1), (1, -1), (1, 1):

                isCorrect = 0  # определение правильности хода

                for i in range(1, COLUMNS):

                    if 0 <= y + iy \* i < COLUMNS and 0 <= x + ix \* i < ROWS:

                        if isCorrect == 1:

                            moves\_list.append(((x, y), (x + ix \* i, y + iy \* i)))  # запись хода в конец списка

                        if scene[y + iy \* i][x + ix \* i] == 3 or scene[y + iy \* i][x + ix \* i] == 4:

                            isCorrect += 1

                        if scene[y + iy \* i][x + ix \* i] == 1 or scene[y + iy \* i][x + ix \* i] == 2 or isCorrect == 2:

                            if isCorrect > 0: moves\_list.pop()  # удаление хода из списка

                            break

        return moves\_list

    def check\_left\_player\_moves(self, moves\_list):  # проверка наличия остальных ходов

        for y in range(ROWS):  # сканируем всё поле

            for x in range(COLUMNS):

                if scene[y][x] == 1:  # шашка

                    for ix, iy in (-1, -1), (1, -1):

                        if 0 <= y + iy < COLUMNS and 0 <= x + ix < COLUMNS:

                            if scene[y + iy][x + ix] == 0:

                                moves\_list.append(((x, y), (x + ix, y + iy)))  # запись хода в конец списка

                            if scene[y + iy][x + ix] == 3 or scene[y + iy][x + ix] == 4:

                                if 0 <= y + iy \* 2 < COLUMNS and 0 <= x + ix \* 2 < COLUMNS:

                                    if scene[y + iy \* 2][x + ix \* 2] == 0:

                                        moves\_list.append(((x, y),

                                                           (x + ix \* 2, y + iy \* 2)))  # запись хода в конец списка

                if scene[y][x] == 2:  # шашка с короной

                    for ix, iy in (-1, -1), (-1, 1), (1, -1), (1, 1):

                        isCorrect = 0  # определение правильности хода

                        for i in range(1, COLUMNS):

                            if 0 <= y + iy \* i < COLUMNS and 0 <= x + ix \* i < ROWS:

                                if scene[y + iy \* i][x + ix \* i] == 0:

                                    moves\_list.append(((x, y), (x + ix \* i, y + iy \* i)))  # запись хода в конец списка

                                if scene[y + iy \* i][x + ix \* i] == 3 or scene[y + iy \* i][x + ix \* i] == 4:

                                    isCorrect += 1

                                if scene[y + iy \* i][x + ix \* i] == 1 or scene[y + iy \* i][

                                    x + ix \* i] == 2 or isCorrect == 2:

                                    break

        return moves\_list

# --------------------------------------- MAIN ----------------------------------------

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

    app = SampleApp()

    app.mainloop()