МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема Компьютерная логическая игра «Евразийские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 23/736 08 ТЗ-01

Листов 54

Руководитель разработки:

Шишкин Вадим

Викторинович

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-21

Вражкин Никита Александрович  
« » 2024 г.

2024

Содержание

Аннотация……...…………………………………………………………3

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

Пояснительная записка...……………………………………………....10

Руководство программиста……………………………….…………..18

Тестовая документация………………………………………………..28

Листинг программы……………………………………………………..32

**АННОТАЦИЯ**

Тема курсового проекта: Компьютерная логическая игра «Евразийские шашки».

Исполнитель: студент гр. ИСТбд-21 Вражкин Никита Александрович.

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 «Евразийские шашки», условия выигрыша, проигрыша, 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 «Евразийские шашки».

Ключевые слова: компьютерная игра, шашки, python.

Всего листов 54.

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 «Евразийские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 23/736 08 ТЗ-02

Листов 6

Исполнитель:

студент гр. ИСТбд-21

Вражкин Никита Александрович   
« » 2024 г.

2024

**Введение**

Разрабатываемое приложение наименования «Евразийские шашки» реализует игру Евразийские шашки. Евразийские шашки – вариант игры в "Шашки" на доске 8х8 с 10 шашками у каждого из игроков. Игра предназначена для двух игроков. Игроки поочередно перемещают шашки на игровом поле. Первый игрок использует шашки белого цвета, а второй игрок - черного. Цель – переместить все свои шашки на противоположную сторону доски в исходное расположение соперника.

**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 халму с графическим интерфейсом в среде Windows.

**2.2 Требования к функциональным характеристикам**

**2.2.1 Требования к структуре приложения**

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

**Правила игры**  
Приложение должно соответствовать следующим правилам Евразийских шашек.

**Поле и игроки**

Игра ведётся между двумя соперниками (пользователь — компьютер или два пользователя) шашками разного цвета (белые и чёрные) на квадратном поле размером 8×8 клеток.

* Игра ведётся только на чёрных клетках доски.
* В начальной позиции у каждого игрока есть 8 шашек, расположенных на первых двух рядах чёрных клеток с каждой стороны.

**Типы ходов**

В Евразийских шашках существует два типа ходов:

1. **Обычные перемещения**
   * Обычные шашки ходят только вперёд по диагонали на одну свободную чёрную клетку.
   * Дамки могут перемещаться на любое количество клеток по горизонтали, вертикали или диагонали, но только на чёрных полях.
2. **Взятие (бой)**
   * Взятие осуществляется вертикально или горизонтально по чёрным клеткам.
   * Для взятия шашка или дамка "перепрыгивает" через шашку или дамку противника, если за ними есть свободное поле.
   * Взятие может выполняться как вперёд, так и назад.
   * Если после взятия возникает возможность продолжить бой, игрок обязан выполнить серию "прыжков" до исчерпания всех доступных вариантов.
   * При наличии нескольких вариантов взятия игрок обязан выбрать путь, который позволяет захватить максимальное количество шашек противника.

**Порядок ходов**

Ходы в игре выполняются поочерёдно:

* Первый ход делает игрок с белыми шашками.
* В течение своего хода игрок может переместить только одну шашку или дамку.

**Цель игры**

Цель игры — заблокировать или уничтожить все шашки противника, либо добиться ситуации, в которой противник не сможет выполнить ход.

**Завершение игры**

Игра завершается в одном из следующих случаев:

1. Все шашки одного из игроков уничтожены.
2. Один из игроков лишён возможности выполнить ход (блокировка).
3. Игрок объявляет сдачу.

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 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. Пояснительные информационные сообщения для пользователя должны выводиться внизу игрового поля по ходу игры, либо во всплывающих сообщениях.

**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 Pro 20H2.

Версия языка программирования: Python 3.9.

Среда разработки: PyCharm Community Edition 2022.3.

При создании программы используются следующие библиотеки:

tkinter, Sys.

**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/sslleeeeppyy/evr_shash>  
2.6. 3 Сроки хранения

Срок хранения – до июля 2027 года.

**3. Требования к программной документации**

Программная документация должна содержать:

* Техническое задание;
* Руководство программиста;
* Пояснительная записка;
* Тестовая документация.

**4. Стадии и этапы разработки**

1. Составление технического задания.
2. Разработка архитектуры проекта.
3. Написание кода программы:
   * + - Реализация движка игры.
       - Создание бота.
       - Разработка пользовательского интерфейса.
4. Тестирование и отладка.
5. Оформление документации.
6. Защита проекта.

**5. Порядок контроля и приёмки**

Контроль осуществляется путем тестирования всех частей приложения. Приемка работы включает повторное тестирование и защиту готового проекта.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

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

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

**Пояснительная записка**

Р.02069337. 23/736 08 ПЗ-01

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Листов 8

Исполнитель:

студент гр. ИСТбд-21

Вражкин Никита Александрович   
«\_\_\_ »\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

2024

**Введение**

Программа представляет из себя однопользовательское десктопное приложение по игре в Евразийские 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 приемы, 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**

1.2.1 Модель игрового поля (матрица начального расположения шашек):

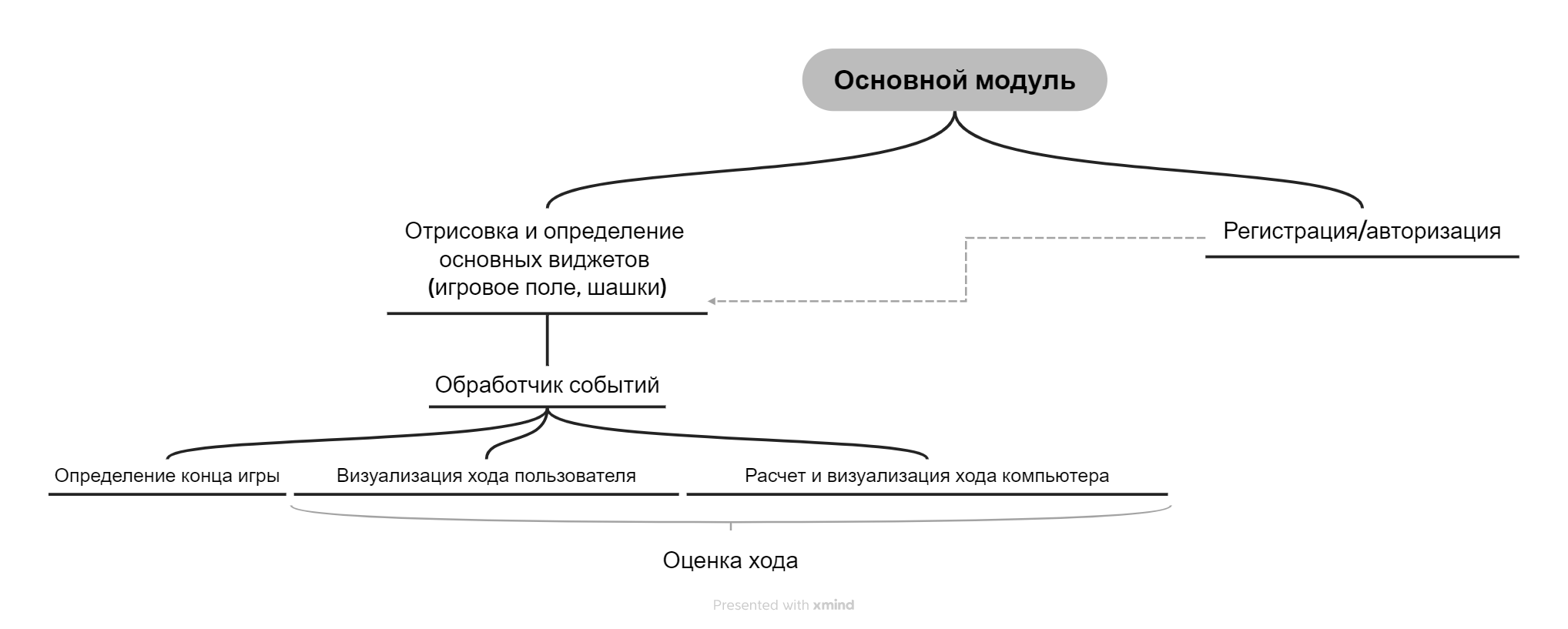
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | -1 | 0 | -1 | 0 | -1 | 0 | -1 |
| -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 |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

1.2.2 Весовая модель поля ( матрица весов, для описания бота):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |

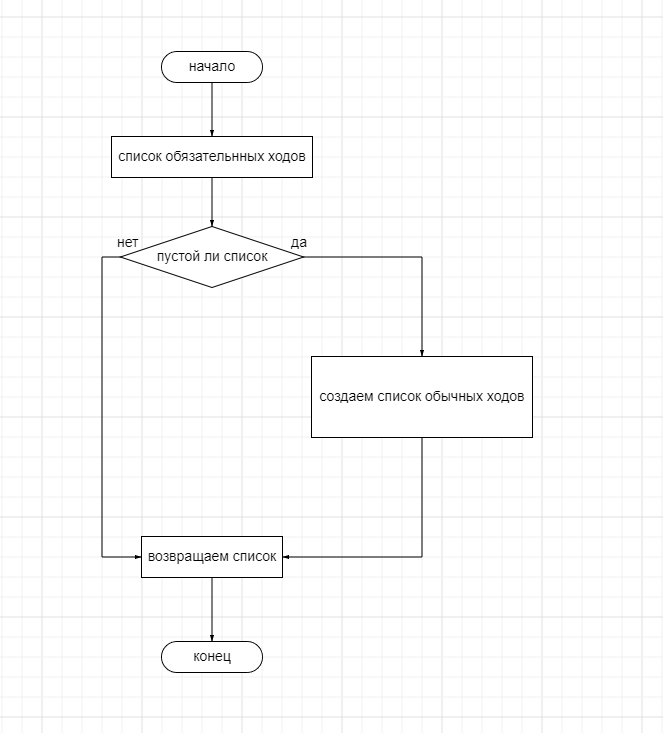
**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура.



При запуске игры создается окно приложения, загружаются графические ресурсы и запускается система авторизации/регистрации пользователей. После успешной авторизации начинается игровой процесс, который включает в себя несколько ключевых элементов: отрисовку игровой доски, отображение истории ходов игроков и обработку пользовательского ввода. Игровая логика построена на обработке событий, где каждое действие игрока проходит проверку на возможность выполнения хода. Важной частью структуры является бот, реализованный через алгоритм минимакс, который управляет ходами белых шашек. Система хранения данных обеспечивает сохранение информации о пользователях в зашифрованном виде.

1.3.2. Алгоритм авторизации/регистрации

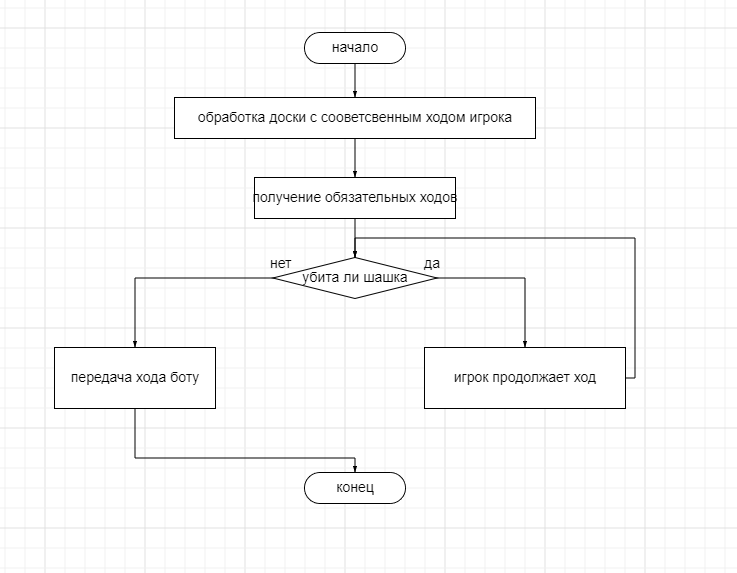


Алгоритм реализует систему авторизации и регистрации пользователей, обеспечивающий персонализацию игрового процесса. Пользователю необходимо ввести свои логин и пароль. При корректном пароле алгоритм проверяет, зарегистрирован ли пользователь, путем поиска его учетных данных в файле users.txt. Если пользователь уже зарегистрирован и ввел правильные данные, происходит успешный вход в игру. Если пользователь не найден, алгоритм информирует об этом. При регистрации введенные данные шифруются и сохраняются в таком виде в файле пользовательских данных.

1.3.3 Алгоритм визуализации игрового поля.

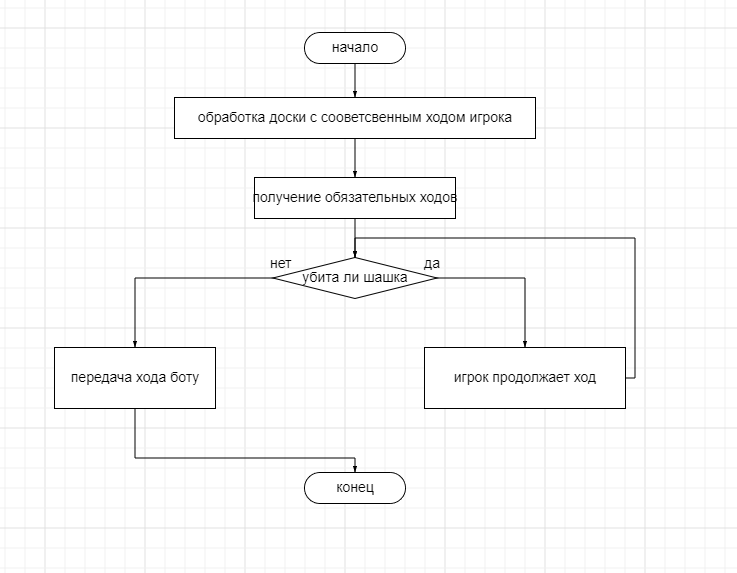
Данный алгоритм отвечает за отрисовку игрового поля. Сначала происходит заполнение экрана цветом, затем накладывается фоновое изображение доски с учетом отступов. После этого создается сетка игрового поля, отрисовка шашек, отображение истории хода каждого игрока.

1.3.4 Алгоритм определение возможности хода.



Алгоритм является основным компонентом, определяющим всю игровую механику, анализирует текущее положение шашки на доске и определяет все допустимые варианты перемещения согласно правилам игры, обеспечивает корректную работу как пользовательского интерфейса, так и бота.

1.3.5 Алгоритм отрисовки хода.

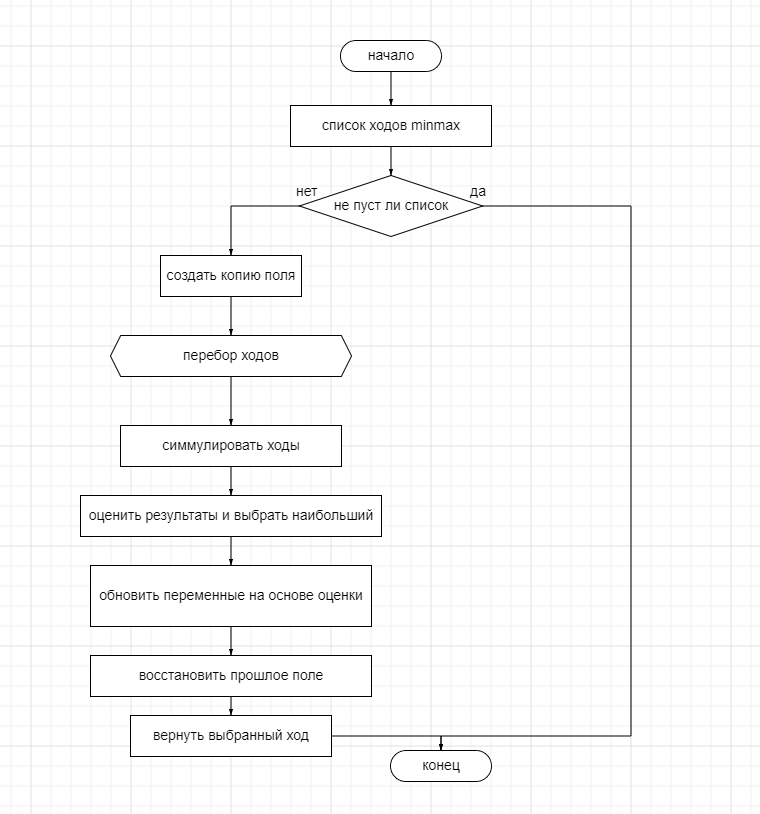


Алгоритм хода шашки осуществляется благодаря «подсветке» возможных ходов, при выборе шашки. После определения конечной позиции происходит анимация перемещения шашки, в конкретную точку.

1.3.6 Алгоритм определения победителя.

Суть алгоритма заключается в проверке достижения шашками захвата всех вражеских шашек. Алгоритм определения победителя является ключевым компонентом игры, обеспечивающим корректное завершение партии и правильную работу бота при оценке позиций и принятии решений.

1.3.7 Алгоритм Минимакс.



**Начало:**

Процесс начинается с вызова функции Minimax.

**Проверка на пустоту списка ходов:**

На этом этапе проверяется, есть ли доступные ходы. Если их нет (например, игра завершена), алгоритм возвращается к предыдущему шагу или завершает выполнение.

Создание копии игрового поля:

Создаётся копия текущего состояния игры. Это необходимо, чтобы не изменять реальное игровое поле во время симуляций.

**Перебор возможных ходов:**

Алгоритм рассматривает все доступные ходы (например, в случае крестиков-ноликов это может быть выбор пустой клетки на поле).

**Симуляция хода:**

Для каждого возможного хода выполняется его симуляция. Алгоритм применяет ход на копию игрового поля, чтобы оценить результат.

**Оценка результата и выбор наилучшего:**

После симуляции оценивается "ценность" результата. Эта оценка зависит от того, выигрывает ли игрок, делает ли ничью или проигрывает. Алгоритм выбирает лучший ход на основе максимальной (или минимальной, если ход соперника) оценки.

**Обновление переменных на основе оценки:**

Значение оценки записывается в переменные, чтобы хранить информацию о лучшем ходе.

**Восстановление игрового поля:**

После симуляции и оценки ходов копия игрового поля возвращается в исходное состояние, чтобы обработать следующий возможный ход.

**Возврат лучшего хода:**

После анализа всех возможных вариантов возвращается ход, который даёт игроку наилучший результат.

**2. Источники, использованные при разработке**

1. 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/disk/2023/112.pdf. – Текст: электронный.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

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

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

**Руководство программиста**

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337. 23/736 08 РП-01

Листов 10

Исполнитель:

студент гр. ИСТбд-21

Вражкин Никита Александрович   
« \_\_» \_\_\_\_\_\_\_\_\_\_\_ 2024 г.

2024

**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 Pygame.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Программа состоит из нескольких файлов с исходным кодом на языке Python. В процессе работы создаётся текстовый файл для регистрации пользователей **users.txt**.

При реализации графического интерфейса программа создаёт основное окно для авторизации/регистрации пользователя. При успешной авторизации пользователь переходит в окно с игровой доской и расставленными шашками в соответствии с правилами Евразийских Шашек. Согласно техническому заданию, игрок играет белыми фигурами и ходит первым. Игровое взаимодействие с пользователем происходит с помощью компьютерной мыши. В программе реализовано несколько окон: *Авторизация, Регистрация, Евразийские шашки.*

Структура классов:

* + Game: основной класс, управляющий игровым процессом.
  + Board: отвечает за логику игрового поля.
  + Checker: реализует функционал шашек.
  + LoginScreen: управляет системой авторизации.

Взаимодействие с пользователем:

* + Обработка событий мыши реализована в методе on\_cell\_click класса CheckersBoard, который отслеживает клики на клетках доски.
  + Выбор шашки и определение целевой позиции происходит через методы get\_available\_moves и move\_checker класса CheckersBoard.

Визуализация игры:

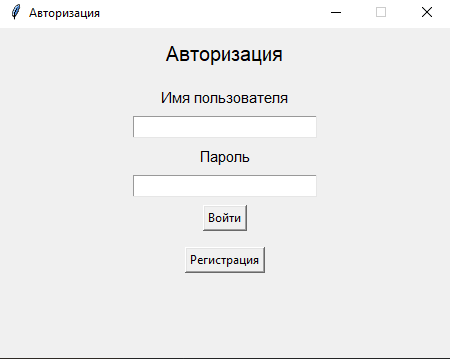
* Отрисовка игрового поля и всех элементов происходит в методе create\_gui класса CheckersBoard.
* Анимация перемещения шашек не реализована явно, но перемещение отображается через обновление графических элементов в методе move\_checker.
* Отображение интерфейса выполняется через методы create\_gui и draw\_starting\_position.
* Загрузка графических ресурсов (изображения шашек, доски) происходит при инициализации игры в методе create\_gui.

Игровая логика:

* Определение возможных ходов реализовано в методе get\_available\_moves класса CheckersBoard.
* Проверка победных условий выполняется методом is\_game\_over.
* Искусственный интеллект для игры за белые шашки реализован через случайный выбор хода в методе auto\_turn класса CheckersBoard.
* Обработка ходов и проверка их корректности выполняется в методе on\_cell\_click.

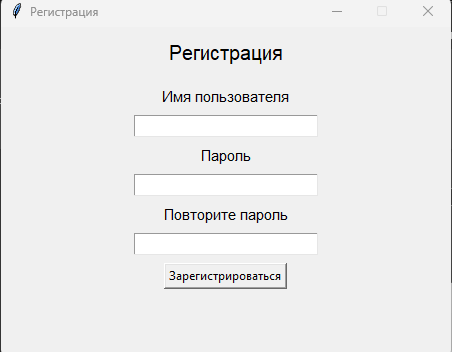
**2. 2. Описание интерфейса программы**

1. Окно авторизации



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

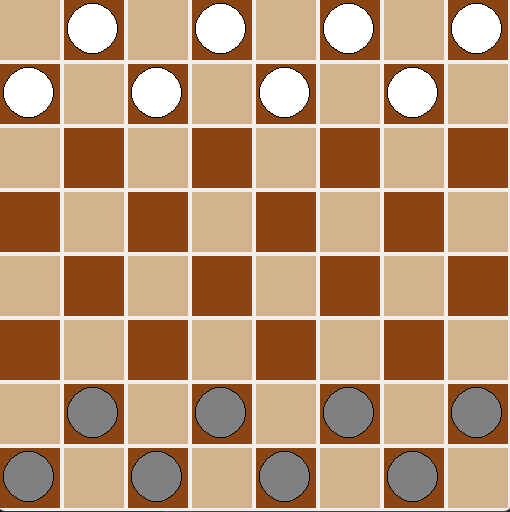
2. Окно регистрации



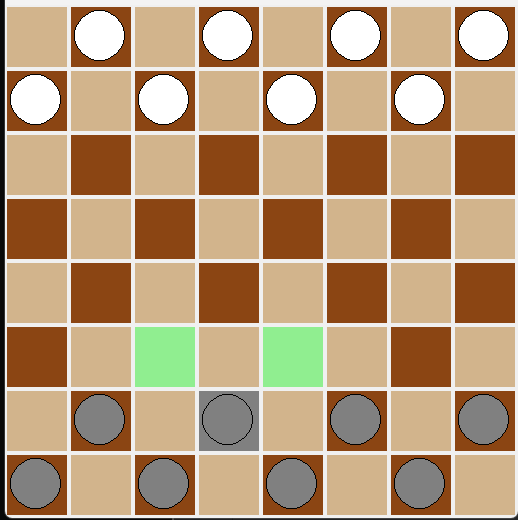
Для начала игры необходимо зарегистрироваться.

**2.3. Игра**

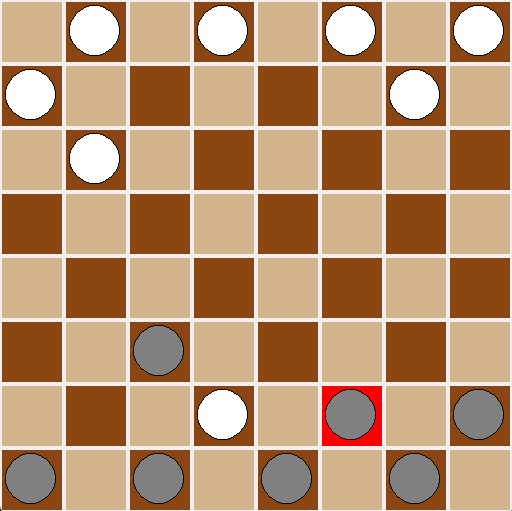
После авторизации в новом окне запускается игра: происходит отрисовка доски и начальная расстановка шашек.



При наведении мыши на клетку игрового поля, клетка подсвечивается синей рамкой, также подсвечиваются шашки, куда возможно сделать ход.



Если ход выбранной шашкой в заданную клетку возможен, то шашка передвигается и ход передаётся компьютеру. При возможности осуществить множественное «перепрыгивание» ходы также подсвечиваются, шашки через которые перепрыгнули исчезают.



При уничтожении всех шашек противника игра заканчивается с уведомлением о победе.

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

1. **Проверка авторизации:**

В классе EnterWindow метод clicked\_auth() контролирует корректность введённых данных. Он проверяет существование пользователя в базе данных и правильность введённых учётных данных. При неверных данных выводится сообщение об ошибке.

1. **Проверка шифрования:**

В классе Coder метод encode() обеспечивает корректное шифрование паролей пользователей с использованием алгоритма Цезаря со сдвигом в 5 позиций.

1. **Проверка победителя:**

В классе CheckersBoard метод is\_game\_over() отслеживает условия окончания игры, такие как отсутствие возможных ходов или всех шашек одного цвета. При достижении условий победы игра завершается с соответствующим сообщением.

1. **Проверка возможности хода:**

В классе CheckersBoard метод find\_possibilities\_to\_attack() проверяет допустимость прыжка, включая проверку занятости целевой клетки и наличие шашки для перепрыгивания.

1. **Проверка доступных ходов:**

Метод get\_available\_moves() в классе CheckersBoard определяет все возможные ходы для выбранной шашки, включая простые перемещения и прыжки через другие шашки.

1. **Проверка хода компьютера:**

Метод auto\_turn() в классе CheckersBoard использует случайный выбор хода для бота, оценивая возможные ходы.

1. **Проверка границ:**

Метод get\_available\_moves() следит, чтобы все ходы оставались в пределах игрового поля 8x8 клеток.

**3. Обращение к программе**

В текущем проекте используется библиотека tkinter для создания графического интерфейса. Основные компоненты включают:

Графический интерфейс:

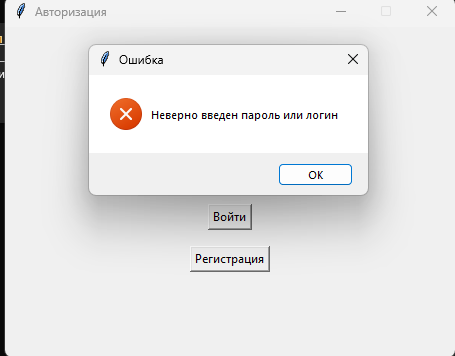
* tkinter.Tk для создания главного окна приложения.
* tkinter.Canvas для отображения игрового поля и шашек.
* tkinter.Label и tkinter.Entry для ввода и отображения текстовой информации.

Обработка событий:

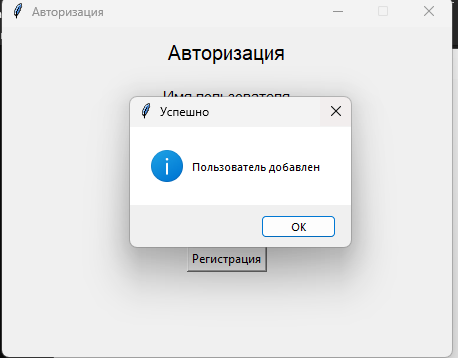
* Методы, такие как on\_cell\_click() в классе CheckersBoard, обрабатывают пользовательский ввод (клики мыши).

**4. Сообщения**

1) *«Неверно введен пароль или логин»* в случае ввода некорректного пароля или логина.



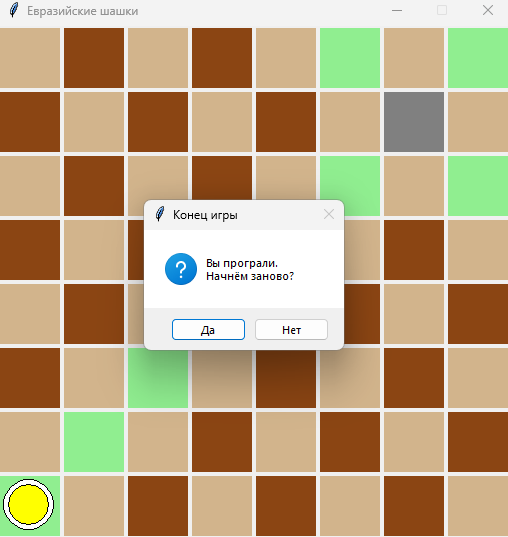
2) *«Пользователь добавлен»* в случае регистрации нового аккаунта



3) *«Вы победили!»* в случае окончании игры при условии проигрыша пользователя.



4) *«Вы проиграли!»* в случае окончании игры при условии выигрыша пользователя.

****

**Тестовая документация**

Для корректности приложения необходимо провести функциональное тестирование компонентов приложения.

Необходимые тест-кейсы:

1. Тесты авторизации и регистрации:
   * Регистрация нового пользователя.

Ожидаемый результат: Программа создает нового пользователя и сохраняет данные в файл users.txt.

Реальный результат: Программа успешно создает пользователя

* + Авторизация существующего пользователя.

Ожидаемый результат: Программа проверяет учетные данные и предоставляет доступ к игре.

Реальный результат: Программа корректно авторизует пользователя и запускает игру.

1. Тесты игрового поля:
   * Корректность начальной расстановки шашек.

Ожидаемый результат: Программа размещает 8 белых шашек сверху и 8 черных снизу в шахматном порядке.

Реальный результат: Программа корректно расставляет все шашки на начальные позиции.

* + Проверка размеров доски.

Ожидаемый результат: Программа создает доску 8x8 клеток с размером клетки 68 пикселей.

Реальный результат: Программа отображает доску с корректными размерами.

* + Проверка корректности отображения белых и черных шашек.

Ожидаемый результат: Программа отображает белые и черные шашки с соответствующими изображениями.

Реальный результат: Программа корректно отображает шашки при наличии файлов изображений.

* + Проверка подсветки выбранной шашки.

Ожидаемый результат: Программа подсвечивает выбранную шашку синим цветом.

Реальный результат: Программа обводит контур клетки, в которой находится выбранная шашка.

* + Проверка отображения возможных ходов.

Ожидаемый результат: Программа подсвечивает доступные ходы темно-синим цветом.

Реальный результат: Программа корректно отображает все возможные ходы.

1. Тесты механики движения:
   * Простой ход на одну клетку.

Ожидаемый результат: Программа позволяет перемещать шашку на соседнюю клетку в любом направлении.

Реальный результат: Программа корректно обрабатывает простые ходы.

* + Прыжок через одну шашку.

Ожидаемый результат: Программа позволяет перепрыгивать через соседнюю шашку на свободное поле.

Реальный результат: Программа корректно обрабатывает прыжки через шашки.

* + Множественные прыжки за один ход.

Ожидаемый результат: Программа позволяет совершать несколько последовательных прыжков за один ход.

Реальный результат: Программа корректно обрабатывает серию прыжков как через свои, так и через шашки противника.

* + Попытка недопустимого хода.

Ожидаемый результат: Программа блокирует недопустимый ход и показывает сообщение об ошибке.

Реальный результат: Программа показывает сообщение "Такой ход сейчас не доступен!".

* + Проверка очередности ходов.

Ожидаемый результат: Программа обеспечивает строгое чередование ходов белых и черных.

Реальный результат: Программа корректно контролирует очередность ходов.

1. Тесты бота:
   * Проверка выбора оптимального хода.

Ожидаемый результат: Программа выбирает ход с наилучшей оценкой по матрице весов.

Реальный результат: Программа успешно выбирает оптимальные ходы.

* + Проверка работы алгоритма минимакс.

Ожидаемый результат: Программа анализирует ходы на указанную глубину с учетом ответов противника.

Реальный результат: Программа корректно применяет алгоритм минимакс.

* + Проверка оценочной функции.

Ожидаемый результат: Программа правильно оценивает позиции по матрице весов.

Реальный результат: Программа корректно использует оценочную функцию.

1. Тесты завершения игры:
   * Победа белых.

Ожидаемый результат: Программа определяет победу белых при отсутствии черных шашек на доске.

Реальный результат: Программа корректно определяет победу белых и отображает окно победителя.

* + Победа черных.

Ожидаемый результат: Программа определяет победу черных при отсутствии белых шашек на доске.

Реальный результат: Программа корректно определяет победу черных и отображает окно победителя.

* + Проверка корректного отображения диалогового окна победителя.

Ожидаемый результат: Программа отображает окно с указанием победителя, вопросом «Начнем игру заново?» и кнопками "Да", и "Нет”.

Реальный результат: Программа корректно отображает окно победителя.

1. Тесты интерфейса:
   * Отображение текущего игрока.

Ожидаемый результат: Программа показывает, чей сейчас ход (белые/черные).

Реальный результат: Программа корректно отображает текущего игрока.

* + Отображение обязательного хода.

Ожидаемый результат: Программа показывает, может ли игрок совершить обязательные ходы, если да, то подсвечивает такие ходы

Реальный результат: Программа корректно отображает обязательные ходы.

* + Отображение сообщений об ошибках.

Ожидаемый результат: Программа показывает сообщения об ошибках.

Реальный результат: Программа корректно отображает и скрывает сообщения об ошибках.

1. Тесты дополнительных файлов:
   * Загрузка ресурсов (изображения, звуки).

Ожидаемый результат: Программа загружает все необходимые изображения и звуковые файлы при запуске.

Реальный результат: Программа успешно загружает ресурсы при их наличии в директории.

**Листинг программы**

**board.py**

import random

import sys

import tkinter as tk

from checker import Checker

from tkinter import messagebox

def get\_cell\_color(row, col):

return "#D2B48C" if (row + col) % 2 == 0 else "#8B4513"

def is\_valid\_cell(row, col):

return 0 <= row < 8 and 0 <= col < 8

class CheckersBoard:

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

# Матрица для хранения шашек

self.cells = [[None for \_ in range(8)] for \_ in range(8)]

# Матрица для хранения графических элементов (канвасов)

self.canvas\_cells = [[None for \_ in range(8)] for \_ in range(8)]

# Предыдущая выделенная клетка

self.prev\_highlighted = None

# Главное окно приложения

self.master = master

# Список доступных ходов

self.highlighted\_moves = []

# Флаг окончания игры

self.isOver = False

# Флаг хода игрока

self.isPlayerTurn = True

# Цвет шашек игрока

self.playerColor = "#808080"

# Цвет шашек бота

self.autoColor = "white"

# Обязательные ходы

self.required\_highlighted = []

# Размещение шашки на доске

def place\_checker(self, checker):

if is\_valid\_cell(checker.row, checker.col):

self.cells[checker.row][checker.col] = checker

# Инициализация начальной позиции шашек

def draw\_starting\_position(self):

for row in range(8):

for col in range(8):

if get\_cell\_color(row, col) == "#8B4513":

if row < 2:

checker\_color = "white" # Цвет для верхних рядов

elif row > 5:

checker\_color = "#808080" # Изменено с "blue" на светло-черный (серый)

else:

checker\_color = None

if checker\_color:

checker = Checker(checker\_color, row, col)

self.place\_checker(checker)

self.canvas\_cells[row][col].create\_oval(5, 5, 55, 55, fill=checker\_color, tags="checker")

# Создание графического интерфейса

def create\_gui(self):

for row in range(8):

for col in range(8):

cell = tk.Canvas(self.master, width=60, height=60, bg=get\_cell\_color(row, col))

cell.grid(row=row, column=col)

cell.bind("<Button-1>", lambda event, r=row, c=col: self.on\_cell\_click(r, c, self.playerColor))

self.canvas\_cells[row][col] = cell

self.draw\_starting\_position()

# Отслеживание нажатий на канвас

def on\_cell\_click(self, row, col, color):

item\_id = self.canvas\_cells[row][col].find\_withtag("checker")

if item\_id:

# Получаем цвет

current\_color = self.canvas\_cells[row][col].itemcget(item\_id, "fill")

# Проверяем, что это наша шашка

if (self.isPlayerTurn and current\_color == self.playerColor) or \

(not self.isPlayerTurn and current\_color == self.autoColor):

# 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

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

color = self.cells[prev\_row][prev\_col].color

# Двигаем

self.move\_checker(prev\_row, prev\_col, row, col)

deleted = False

# Если далеко пошли, значит надо проверить, кого убили

if abs(prev\_row - row) > 2 or abs(prev\_col - col) > 2:

deleted = True

self.delete\_checker(prev\_row, prev\_col, row, col, self.autoColor if color == self.playerColor else self.playerColor)

# 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 == self.autoColor: # "white"

moves.append((row + 1, col - 1))

moves.append((row + 1, col + 1))

elif checker.color == self.playerColor: # "#808080"

moves.append((row - 1, col - 1))

moves.append((row - 1, col + 1))

# Теперь уберем из списка те ходы, которые выходят за границы доски

moves = [(r, c) for r, c in moves if 0 <= r < 8 and 0 <= c < 8]

# Уберем из списка ходы, которые уже заняты другими шашками

moves = [(r, c) for r, c in moves if self.cells[r][c] is None]

return moves

# Ходы королевы

def find\_queen\_moves(self, row, col, row\_dif, col\_dif):

moves = []

ch\_row = row + row\_dif

ch\_col = col + col\_dif

while 8 > ch\_col >= 0 and 8 > ch\_row >= 0:

if self.cells[ch\_row][ch\_col] is None and get\_cell\_color(ch\_row, ch\_col) == '#8B4513':

moves.append((ch\_row, ch\_col))

else:

break

ch\_row += row\_dif

ch\_col += col\_dif

return moves

# Проверка наличия возможных ходов для заданного цвета

def is\_there\_moves(self, color):

return len(self.get\_possible\_checkers(color)) > 0

# Подсчёт количества шашек заданного цвета

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 окончание игры

def is\_game\_over(self, color):

checkers\_count = self.count\_checkers\_by\_color(color)

is\_game\_over = False

is\_lose = False

if checkers\_count == 0:

is\_game\_over = True

if color == self.playerColor:

is\_lose = True

if not self.is\_there\_moves(self.autoColor) or not self.is\_there\_moves(self.playerColor):

is\_game\_over = True

if self.count\_checkers\_by\_color(self.playerColor) < self.count\_checkers\_by\_color(self.autoColor):

is\_lose = True

if is\_game\_over:

if is\_lose:

message = 'Вы програли.\nНачнём заново?'

else:

message = 'Вы победили!\nНачнём заново?'

if messagebox.askyesno('Конец игры', message):

self.restart()

else:

sys.exit(1)

return is\_game\_over

# Очистка подсветки доступных ходов

def clear\_highlighted\_moves(self):

for inner\_list in self.canvas\_cells:

for val in inner\_list:

if val["background"] != '#D2B48C': # Светло-коричневый цвет

val.config(bg='#8B4513') # Темно-коричневый цвет

# Очистка сохранённых ходов

def clear\_saved\_moves(self):

self.prev\_highlighted = []

self.required\_highlighted = []

self.highlighted\_moves = []

# Перемещение шашки

def move\_checker(self, prev\_row, prev\_col, row, col):

checker = self.cells[prev\_row][prev\_col]

self.cells[prev\_row][prev\_col] = None

checker.move(row, col)

self.cells[row][col] = checker

if (checker.color == self.playerColor and row == 0) or (checker.color == self.autoColor and row == 7):

checker.is\_queen = True

# Начало игры

def start\_game(self):

self.create\_gui()

# Поиск обязательных ходов для заданного цвета

def find\_necessary\_moves(self, color):

self.clear\_highlighted\_moves()

for row in range(8):

for col in range(8):

if self.cells[row][col] is None:

continue

if self.cells[row][col].color != color:

continue

if self.cells[row][col].is\_queen:

continue

if len(self.find\_possibilities\_to\_attack(row, col)) > 0:

self.required\_highlighted.append((row, col))

# Берем шашку и оцениваем, можно ли ходить

def find\_possibilities\_to\_attack(self, row, col):

color = self.cells[row][col].color

moves = []

if not self.cells[row][col].is\_queen:

if (col - 4 >= 0) and (self.cells[row][col - 2] is not None) and (

self.cells[row][col - 2].color != color) and (self.cells[row][col - 4] is None):

moves.append((row, col - 4))

if (col + 4 < 8) and (self.cells[row][col + 2] is not None) and (

self.cells[row][col + 2].color != color) and (self.cells[row][col + 4] is None):

moves.append((row, col + 4))

if (row - 4 >= 0) and (self.cells[row - 2][col] is not None) and (

self.cells[row - 2][col].color != color) and (self.cells[row - 4][col] is None):

moves.append((row - 4, col))

if (row + 4 < 8) and (self.cells[row + 2][col] is not None) and (

self.cells[row + 2][col].color != color) and (self.cells[row + 4][col] is None):

moves.append((row + 4, col))

else:

moves.extend(self.find\_queen\_attack(row, col, -1, 0, color))

moves.extend(self.find\_queen\_attack(row, col, 1, 0, color))

moves.extend(self.find\_queen\_attack(row, col, 0, -1, color))

moves.extend(self.find\_queen\_attack(row, col, 0, 1, color))

return moves

def find\_queen\_attack(self, row, col, row\_dif, col\_dif, color):

enemy\_has\_met = False

moves = []

ch\_row = row + row\_dif

ch\_col = col + col\_dif

while 8 > ch\_col >= 0 and 8 > ch\_row >= 0:

if self.cells[ch\_row][ch\_col] is None and get\_cell\_color(ch\_row, ch\_col) == 'black' and enemy\_has\_met:

moves.append((ch\_row, ch\_col))

if self.cells[ch\_row][ch\_col] is not None:

if enemy\_has\_met:

break

else:

if self.cells[ch\_row][ch\_col].color == color:

break

else:

enemy\_has\_met = True

ch\_row += row\_dif

ch\_col += col\_dif

return moves

def auto\_turn(self):

if len(self.required\_highlighted) == 0:

self.find\_necessary\_moves(self.autoColor)

if len(self.required\_highlighted) > 0:

move = random.choice(self.required\_highlighted)

prev\_row = move[0]

prev\_col = move[1]

self.on\_cell\_click(prev\_row, prev\_col, self.autoColor)

row, col = random.choice(self.highlighted\_moves)

else:

checker = random.choice(self.get\_possible\_checkers(self.autoColor))

prev\_row, prev\_col = checker.row, checker.col

self.on\_cell\_click(prev\_row, prev\_col, self.autoColor)

next\_move = random.choice(self.highlighted\_moves)

row, col = next\_move[0], next\_move[1]

self.on\_cell\_click(row, col, self.autoColor)

# Получение шашек, которые могут ходить

def get\_possible\_checkers(self, color):

checkers = []

for row in range(8):

for col in range(8):

moves = []

checker = self.cells[row][col]

if checker is None:

continue

if checker.color != color:

continue

row, col = checker.row, checker.col

# Шашки могут двигаться по диагонали

if checker.is\_queen:

moves.extend(self.find\_queen\_moves(row, col, -1, 1))

moves.extend(self.find\_queen\_moves(row, col, -1, -1))

moves.extend(self.find\_queen\_moves(row, col, 1, 1))

moves.extend(self.find\_queen\_moves(row, col, 1, -1))

else:

if self.autoColor == checker.color:

moves.append((row + 1, col - 1))

moves.append((row + 1, col + 1))

elif self.playerColor == checker.color:

moves.append((row - 1, col - 1))

moves.append((row - 1, col + 1))

# Убираем ходы, выходящие за границы доски

moves = [(r, c) for r, c in moves if 0 <= r < 8 and 0 <= c < 8]

# Убираем ходы, занятые 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

# Удаление шашки

def delete\_checker(self, prev\_row, prev\_col, row, col, enemy\_color):

if prev\_row > row:

low\_row = row

high\_row = prev\_row

else:

low\_row = prev\_row

high\_row = row

if prev\_col > col:

low\_col = col

high\_col = prev\_col

else:

low\_col = prev\_col

high\_col = col

for r in range(low\_row, high\_row + 1):

for c in range(low\_col, high\_col + 1):

checker = self.cells[r][c]

if checker is not None and checker.color == enemy\_color:

self.cells[r][c] = None

self.canvas\_cells[r][c].delete('checker')

self.canvas\_cells[r][c].delete('crown')

# Перезапуск игры

def restart(self):

self.cells = [[None for \_ in range(8)] for \_ in range(8)]

self.canvas\_cells = [[None for \_ in range(8)] for \_ in range(8)]

self.prev\_highlighted = None

self.highlighted\_moves = []

self.isOver = False

self.isPlayerTurn = True

self.required\_highlighted = []

self.start\_game()

**checker.py**

class Checker:

def \_\_init\_\_(self, color, row, col):

# Инициализация атрибутов шашки

self.color = color

self.row = row

self.col = col

self.is\_queen = False

def move(self, new\_row, new\_col):

# Перемещение шашки на новые координаты

self.row = new\_row

self.col = new\_col

**encode.py**

class Coder:

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

# Алфавит для шифрования

self.alfavit\_EU = 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890wertyuioqasdfghjklzxcvbnm-!"#$%&()\*,./:;?@[]^\_`{|}~+<=>'

# Смещение для шифрования

self.smeshenie = 5

self.itog = ''

def encode(self, message):

# Шифрование сообщения

self.itog = ''

for i in message:

mesto = self.alfavit\_EU.find(i) # Алгоритм для шифрования сообщения на английском

new\_mesto = mesto + self.smeshenie

if i in self.alfavit\_EU:

self.itog += self.alfavit\_EU[new\_mesto % len(self.alfavit\_EU)]

else:

self.itog += i

return self.itog

**enter\_window.py**

import tkinter as tk

from pass\_manager import Manager

from register\_window import RegisterWindow

from tkinter import messagebox

class EnterWindow(tk.Tk):

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

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

self.manager = Manager()

self.response = False

# Создание и настройка окна авторизации

self.title('Авторизация')

self.geometry('450x330')

self.resizable(False, False)

font\_header = ('Arial', 15)

font\_entry = ('Arial', 12)

label\_font = ('Arial', 11)

base\_padding = {'padx': 10, 'pady': 8}

header\_padding = {'padx': 10, 'pady': 12}

# Обработка нажатия кнопки "Войти"

def clicked\_auth():

username = username\_entry.get()

password = password\_entry.get()

if self.manager.get\_user(username, password):

self.response = True

self.destroy()

else:

messagebox.showerror('Ошибка', 'Неверно введен пароль или логин')

# Обработка нажатия кнопки "Регистрация"

def clicked\_register():

register\_window = RegisterWindow()

main\_label = tk.Label(self, text='Авторизация', font=font\_header, justify=tk.CENTER, \*\*header\_padding)

main\_label.pack()

# Метка для поля ввода имени

username\_label = tk.Label(self, text='Имя пользователя', font=label\_font, \*\*base\_padding)

username\_label.pack()

# Поле ввода имени

username\_entry = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)

username\_entry.pack()

# Метка для поля ввода пароля

password\_label = tk.Label(self, text='Пароль', font=label\_font, \*\*base\_padding)

password\_label.pack()

# Поле ввода пароля

password\_entry = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)

password\_entry.pack()

# Кнопка отправки формы

send\_btn1 = tk.Button(self, text='Войти', command=clicked\_auth)

send\_btn1.pack(\*\*base\_padding)

# Кнопка отправки регистрации

send\_btn2 = tk.Button(self, text='Регистрация', command=clicked\_register)

send\_btn2.pack(\*\*base\_padding)

**game.py**

import tkinter as tk

from board import CheckersBoard

class Game(tk.Tk):

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

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

# Создание и настройка главного окна игры

self.title('Евразийские шашки')

self.geometry('513x513')

self.resizable(False, False)

self.board = CheckersBoard(self)

self.start\_the\_game()

def mainloop(self, n=0):

super().mainloop()

self.start\_the\_game()

def start\_the\_game(self):

# Инициализация и запуск игрового процесса

self.board.start\_game()

**Main.py**

import sys

from enter\_window import EnterWindow

from game import Game

if \_\_name\_\_ == '\_\_main\_\_':

enter\_window = EnterWindow()

enter\_window.mainloop()

if not enter\_window.response:

sys.exit(1)

game\_window = Game()

game\_window.mainloop()

**pass\_manager.py**

from encode import Coder

import os

class Manager:

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

self.coder = Coder()

self.file\_name = 'users.txt'

self.init\_file()

def init\_file(self): # Инициализация файла, если этого не сделать програма вылетит м ошибкой, что файла нет

"""Создает файл пользователей"""

if not os.path.exists(self.file\_name):

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

pass

def add\_user(self, login: str, password: str) -> bool:

"""Добавляет пользователя в файл"""

with open(self.file\_name, 'r') as f:

users = f.read().splitlines() # Считываем всех пользователей из файла

for user in users:

args = user.split(':')

if login == args[

0]: # Если логин уже есть, парль не проверяем, шанс взлома увеличится(кто-то мб узнает пароль)

return False # Тут можно написать что угодно, будь то HTML статус(409 - conflict), либо просто фразу ошибки

result = self.coder.encode(password)

with open('users.txt', 'a') as f:

f.write(f'{login}:{result}\n') # Добавляем нового пользователя

return True

def get\_user(self, login: str, password: str) -> bool:

"""Проверяет логин и пароль пользователя"""

with open('users.txt', 'r') as f:

users = f.read().splitlines() # Считываем всех пользователей из файла

result = self.coder.encode(password)

for user in users:

args = user.split(':')

if login == args[0] and result == args[1]: # Если пользователь с таким логином и паролем существует

return True

return False

**register\_window.py**

import tkinter as tk

from pass\_manager import Manager

from tkinter import messagebox

class RegisterWindow(tk.Tk):

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

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

self.manager = Manager()

# Создание и настройка окна

self.title('Регистрация')

self.geometry('450x330')

self.resizable(False, False)

font\_header = ('Arial', 15)

font\_entry = ('Arial', 12)

label\_font = ('Arial', 11)

base\_padding = {'padx': 10, 'pady': 8}

header\_padding = {'padx': 10, 'pady': 12}

# обработчик нажатия на клавишу 'Регистрация'

def clicked\_register():

password1 = password\_entry\_1.get()

password2 = password\_entry\_2.get()

if len(password1) == 0:

return

if password2 != password1:

messagebox.showerror('Ошибка', 'Пароли не совпадают')

return

username = username\_entry.get()

if self.manager.add\_user(username, password1):

messagebox.showinfo('Успешно', 'Пользователь добавлен')

self.destroy()

else:

messagebox.showerror('Ошибка', 'Измените имя пользователя или пароль')

main\_label = tk.Label(self, text='Регистрация', font=font\_header, justify=tk.CENTER, \*\*header\_padding)

main\_label.pack()

# метка для поля ввода имени

username\_label = tk.Label(self, text='Имя пользователя', font=label\_font, \*\*base\_padding)

username\_label.pack()

# поле ввода имени

username\_entry = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)

username\_entry.pack()

# метка для поля ввода пароля

password\_label\_1 = tk.Label(self, text='Пароль', font=label\_font, \*\*base\_padding)

password\_label\_1.pack()

# поле ввода пароля

password\_entry\_1 = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)

password\_entry\_1.pack()

# метка для поля ввода пароля

password\_label\_2 = tk.Label(self, text='Повторите пароль', font=label\_font, \*\*base\_padding)

password\_label\_2.pack()

# поле ввода пароля

password\_entry\_2 = tk.Entry(self, bg='#fff', fg='#444', font=font\_entry)

password\_entry\_2.pack()

# кнопка отправки регистрации

send\_btn = tk.Button(self, text='Зарегистрироваться', command=clicked\_register)

send\_btn.pack(\*\*base\_padding)

# запускаем главный цикл окна

#self.window.mainloop()