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

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

по дисциплине «Алгоритмы и структуры данных»

# Тема Компьютерная логическая игра «80-клеточные русские шашки»

# Р.02069337. 22/2378-23

Листов 46

Руководитель разработки:

Доцент Шишкин Вадим Викторинович

**« » 2023 г.**

Исполнитель:

студент гр. ИСТбд-22

*Кабанов Георгий Викторович*

« » 202 г.

2023

Содержание

Аннотация……...…………………………………………………………3

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

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

Руководство пользователя……………………………….…………...17

Текст программы…..…………………………………………………...24

**Аннотация**

Данный документ представляет собой пояснительную записку на курсовую работу на тему «Компьютерная логическая игра 80-клеточные русские шашки». Документ содержит следующие разделы: техническое задание, пояснительная записка и руководство программиста, код программы; в нем излагается постановка задачи и описание реализуемой программы, ее назначение. Документ может быть использован в качестве инструкции для применения рассматриваемого программного средства.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# на курсовую работу

# по дисциплине «Алгоритмы и структуры данных»

# Тема: Компьютерная логическая игра «80-клеточные русские шашки»

# Р.02069337. 22/2378-8

# Листов 4

**Исполнитель**:

студент гр. ИСТбд-22

*Кабанов Георгий Викторович*

« » 2023 г.

**2023**

**Введение**

Курсовая работа представляет собой однопользовательское десктопное приложение, реализующее игру в 80-клеточные русские шашки.

Краткие правила игры

В 80-клеточных русских шашках работают следующие правила:

* Доска имеет размер 10 на 8 клеток.
* У обоих сторон есть по 15 шашек.
* Шашки, попадая на последнюю противоположную горизонталь, становятся дамками.

Функциональные возможности

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

1. **Основания для разработки**

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

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

**2.1. Функциональное назначение**

Требуется разработать однопользовательское десктопное приложение по игре в 80-клеточные русские шашки с графическим интерфейсом в среде Windows.

**2.2 Требования к функциональным характеристикам**

Приложение должно соответствовать следующим правилам игры:

* Игра ведётся между двумя соперниками (пользователь-компьютер) шашками разного цвета на прямоугольном поле размером 10х8 клеток. В начальной позиции у каждого игрока 15 шашек, расставленных на чёрных клетках доски.
* Игроки совершают ходы поочерёдно. Первый ход делает пользователь. В процессе хода игрок может переместить только одну свою шашку. Простая шашка ходит по диагонали вперёд на одну клетку. Дамка ходит по диагонали на любое свободное поле как вперёд, так и назад. Взятие обязательно. Побитые шашки и дамки снимаются только после завершения хода. Простая шашка, находящаяся рядом с шашкой соперника, за которой имеется свободное поле, переносится через эту шашку на это свободное поле. Если есть возможность продолжить взятие других шашек соперника, то это взятие продолжается, пока бьющая шашка не достигнет положения, из которого бой невозможен. Взятие простой шашкой производится как вперёд, так и назад. Дамка бьёт по диагонали, как вперёд, так и назад, и становится на любое свободное поле после побитой шашки. Аналогично, дамка может бить несколько фигур соперника и должна бить до тех пор, пока это возможно.

Партия считается закончившейся вничью в следующих случаях:

* Если один из участников предлагает ничью, а другой её принимает;
* При невозможности выигрыша ни одного из соперников;
* Если в течение 15 ходов игроки делали ходы только дамками, не передвигая простых шашек и не производя взятия.

Цель игры – лишить противника всех его шашек..

2.2.1 Требования к структуре приложения

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

2.2.2 Требования к составу функций приложения

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

- регистрация/авторизация пользователя;

- отрисовка игрового поля;

- взаимодействие с пользователем;

- интерактивные приём, проверка правильности и отрисовка хода пользователя;

- проверка окончания игры;

- вычисление, проверка правильности и отрисовка хода компьютера;

- информирование пользователя об окончании игры и победителе.

2.2.2 Требования к организации информационного обеспечения, входных и выходных данных

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

**2.3 Требования к надёжности**

Программа должна нормально функционировать при бесперебойной работе ЭВМ. При возникновении сбоя в работе аппаратуры, восстановление нормальной работы программы должно производиться после: перезагрузки операционной системы; запуска исполняемого файла программы; повторного выполнения действий, потерянных до последнего сохранения информации в файл на диске. Уровень надёжности программы должен соответствовать технологии программирования, предусматривающей: инспекцию исходных текстов программы; автономное тестирование модулей (методов) программы; тестирование сопряжении модулей (методов) программы; комплексное тестирование программы.

**2.4 Требования к информационной и программной совместимости**

Операционная система: Windows 11

Используемые библиотеки: tkinter, os, random, time, copy

Язык: Python 3.11

Среда разработки: PyCharm Community Edition 2023.2.1

**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

Хранится в репозиторие личного кабинета на сайте github.com

2.6 3 Сроки хранения

Срок хранения – до июля 2024 года.

**3. Требования к программной документации**

Определяются заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

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

**Тема** Компьютерная логическая игра «80-клеточные русские шашки»

**Пояснительная записка**

# Р.02069337. 22/2378-23

Листов 8

**Исполнитель**:

студент гр. ИСТбд-22

*Кабанов Георгий Викторович*

« » 2023 г.

2023

**Введение**

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализованной игры. Приводится описание и обоснование выбранного подхода, краткое описание реализованного приложения.

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу.

* 1. **Математические методы**

1. Описание состояния игры:

* + Положение фигур на доске с шашками. Необходимо использование двумерного массива размером 10x8, где каждый элемент представляет ячейку доски и содержит информацию о наличии или отсутствии фигуры, стоимость фигуры.
  + Состояние игры: текущий игрок, наличие взятия на проходе, возможность хода выбранной шашки, отслеживание таких ситуаций, как: взятие шашки и превращение из шашки в дамку.

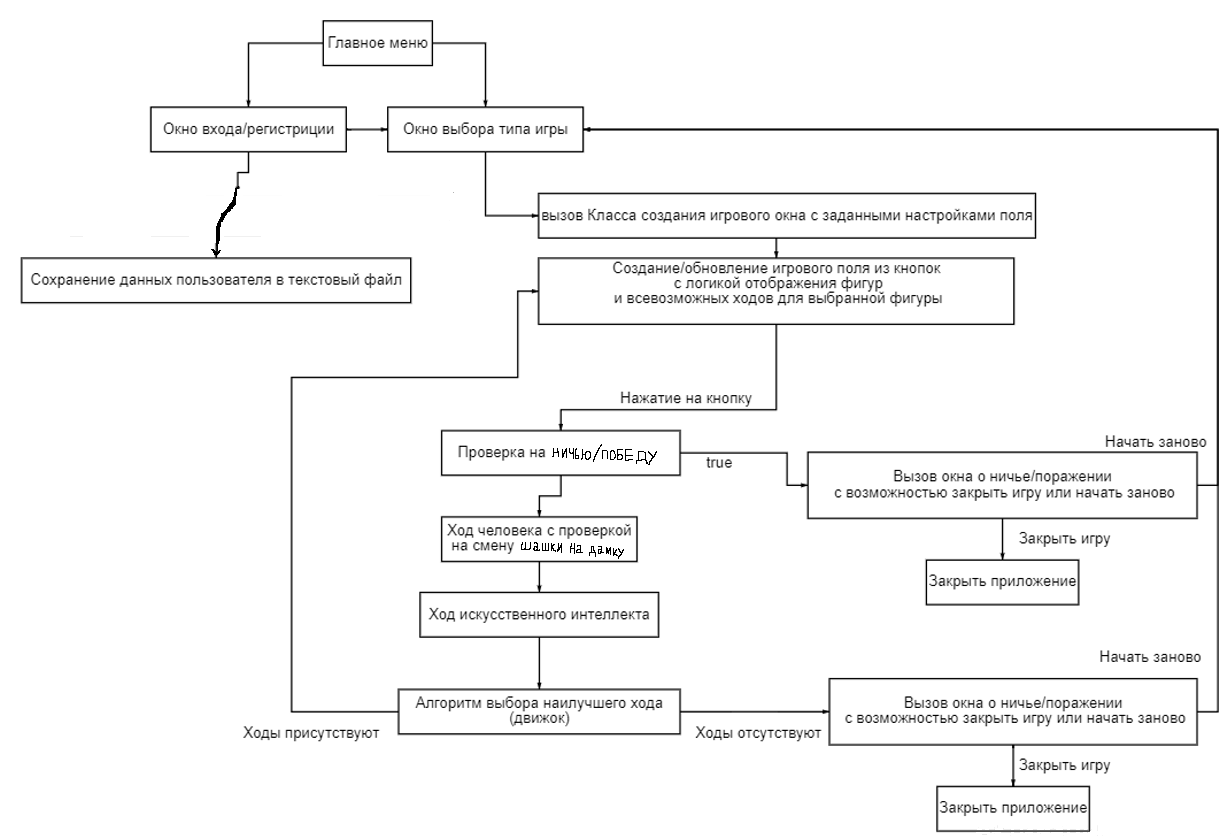
1. Правила игры:
   * Правила перемещения шашек, а так же ограничения при взятии шашек противника.
   * Ограничения на действия: нельзя ходить на занятую клетку, если есть возможность взять шашку, то это делается обязательно, нельзя ходить за пределы доски.
   * Обязательные условия окончания игры, такие как ничья и взятие всех пешек.
2. Алгоритмы:
   * Создание списка возможных ходов для шашек и дамок.
   * Проверка возможности хода с учетом правил и ограничений.
   * Определение возможных ходов для противника, оценка выгодности позиции, определение наилучшего хода с помощью алгоритма минимакс с альфа-бета отсечением.

Выбор такой модели обоснован тем, что русские шашки являются стратегической настольной игрой, в которой игроки принимают решения на основе анализа возможных ходов и прогнозирования дальнейших ходов соперника. Математическая модель позволяет представить все состояния игры, правила и алгоритмы в виде логических и математических выражений, что упрощает анализ и программную реализацию игры.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

Основные структуры данных(классы и основные методы в классе):



1.3.2. Алгоритм Минимакс с альфа-бета отсечением

Алгоритм минимакс с альфа-бета отсечением является классическим алгоритмом для поиска оптимального хода в играх, основанных на двух игроках с полной информацией, таких как шахматы.

Процедура Minimax:

* + Вход: текущее состояние доски, глубина поиска, игрок, α (минимальное значение), β (максимальное значение)
  + Выход: возвращается оценка лучшего хода

1. Если текущее состояние доски является конечным (например, победа, ничья или поражение) или достигнута максимальная глубина, вернуть оценку этого состояния.
2. Если текущий игрок - максимизирующий игрок:
   * Установить лучшую оценку на очень низкое значение.
   * Для каждого возможного хода из текущего состояния доски:
     + Сделать ход.
     + Вызвать рекурсивно Minimax для следующего состояния, уменьшая глубину на 1.
     + Обновить лучшую оценку, если найден лучший ход.
     + Обновить значение α с помощью лучшей оценки.
     + Проверить, если можно применить альфа-бета отсечение и прервать цикл, если это возможно.
     + Отменить ход.
3. Если текущий игрок - минимизирующий игрок:
   * Установить лучшую оценку на очень высокое значение.
   * Для каждого возможного хода из текущего состояния доски:
     + Сделать ход.
     + Вызвать рекурсивно Minimax для следующего состояния, уменьшая глубину на 1.
     + Обновить лучшую оценку, если найден худший ход.
     + Обновить значение β с помощью лучшей оценки.
     + Проверить, если можно применить альфа-бета отсечение и прервать цикл, если это возможно.
     + Отменить ход.
4. Вернуть лучшую оценку.

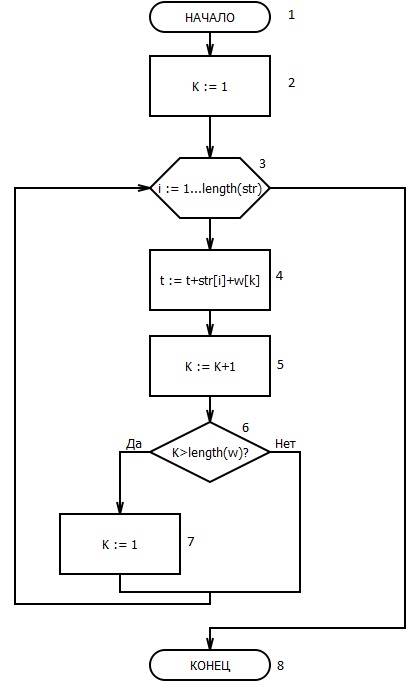
1.3.2 Алгоритм шифрования(дешифрования) Цезаря

Алгоритм шифрования Цезаря - это простой метод шифрования, в котором каждая буква в открытом тексте заменяется на букву сфинеца через определенное смещение. Вот алгоритм шифрования Цезаря по пунктам:

Задать открытый текст, который требуется зашифровать.

1. Задать значение смещения, которое будет использоваться для замены каждой буквы в тексте.
2. Создать пустую строку для хранения зашифрованного текста.
3. Для каждой буквы в открытом тексте выполнить следующие действия:
   * Определить позицию буквы в алфавите (например, "A" будет иметь позицию 0, "B" - 1 и т.д.).
   * Применить смещение к позиции, используя операцию модуля (%), чтобы вернуться к началу алфавита, если позиция смещается за его пределы.
   * Получить новую букву, используя новую позицию и алфавит для поиска соответствующей буквы.
   * Добавить новую букву в зашифрованный текст.
4. Вернуть зашифрованный текст.

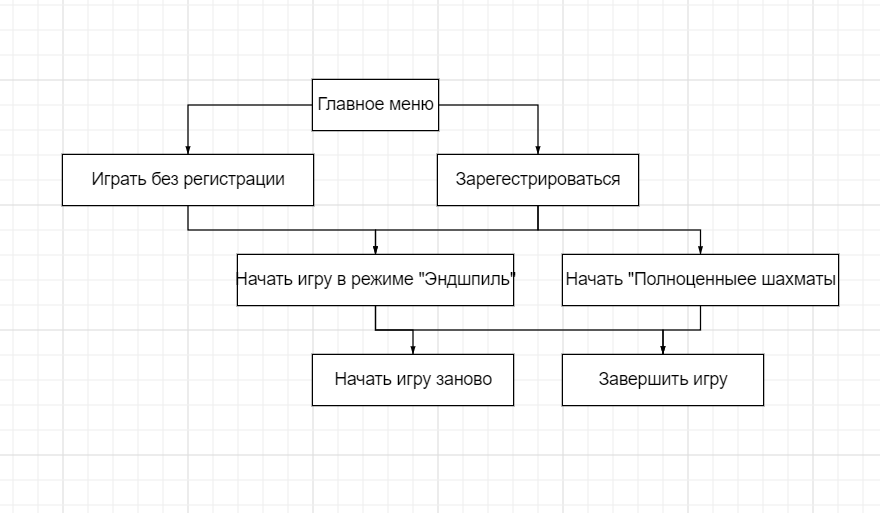
Алгоритм дешифрования аналогичен, за исключением того, что смещение происходит «в другую сторону»



Количество рассмотренных алгоритмов должно полностью соответствовать техническому заданию. Для алгоритмов шифрования, дешифрования, проверки доступности хода, хода и оценочного добавить блок- схему. В схемах и комментариях не должны использоваться конструкции языка программирования, конкретные названия объектов программы и т.п. (пояснительная записка создаётся до реализации).

**1.4 Тестирование**

Интеллектуальная карта приложения



1.4.1 Описание отчета о тестировании

Отчёт служит для: определения соответствия исходного продукта техническому заданию, обнаружения дефектов в исходном приложении, исторической фиксации информации о разработке приложения, анализа для дальнейшего улучшения приложения.

1.4.2 Цель тестирования

Целью тестирования является проверка соответствия ПО предъявляемым требованиям, а также выявление возможных багов. По результатам тестирования следует исправление выявленных багов.

1.4.3 Методика тестирования

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

4.1) Статическое тестирование

Количество обнаруженных и исправленных ошибок в документации: 3.

Количество обнаруженных и исправленных ошибок в программном

коде: 2.

4.2) Ручное тестирование

ТК1 Регистрация/вход с верным вводом данных

Шаги:

1. Запустить главное меню
2. Нажать на кнопку «Регистрация»
3. Ввести верные логин и пароль

Ожидание:

Переход в меню выбора режима игры

Фактический результат:

Переход в меню выбора режима игры

ТК2 Регистрация/вход с неверным вводом данных

Шаги:

1. Запустить главное меню
2. Нажать на кнопку «Регистрация»
3. Ввести неверные логин и пароль

Ожидание:

Вывод сообщения об ошибке

Фактический результат:

Вывод сообщения об ошибке

ТК3 Запуск игры

Шаги:

1. Совершил ход по правилам в игровом окне
2. Совершил ход не по правилам в игровом окне
3. Совершил ход не дождавшись хода соперника(искусственного интеллекта

Ожидание:

Игра продолжается в нормальном режиме. В случае шага 3 искусственный интеллект реагирует на ход оппонента

Фактический результат:

Игра продолжается в нормальном режиме. В случае шага 3 искусственный интеллект реагирует на ход оппонента

1.4.5 Выводы

На основе проведенных тестов сделаны следующие выводы:

– Программа успешно прошла все тесты и работает корректно.

– Обнаружены и исправлены следующие дефекты: Отсутствие хода соперника при «быстром» ходе игрока, отображение всевозможных ходов при выборе игроком пешки.

**2. Источники, использованные при разработке**

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ [Текст] - 3-е изд. - М.: Вильямс, 2004. - 1345 с.

2. "Metanit.com" [Электронный ресурс]. - Режим доступа: https://metanit.com. - Дата доступа: 20.12.2023

3. "ART HAB" [Электронный ресурс]. - Режим доступа: https://www.articleshub.net/2023/04/kak-sdelat-shashki-na-python.html. - Дата доступа: 21.12.2023

4. Dzen.ru [Электронный ресурс]. - Режим доступа: https://dzen.ru/a/YwU5ybOuaELgpTxb. - Дата доступа: 20.12.2023

5. Шишкин В.В., Афонин Д.С. Разработка логических компьютерных игр с графическим интерфейсов в среде Питон [Текст] - УлГТУ, 2023. – 88 с.

**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 «80-точные русские шашки»

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

**Руководство программиста**

Р.02069337. 22/2366-8

Листов 9

Исполнитель:

студент гр. ИСТбд-22

*Кабанов Георгий Викторович*

« » 2023 г.

2023

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Формулируется назначение приложения. Даётся свод правил реализуемой игры, общая характеристика конкретных функциональных возможностей, которое предоставляет приложение.

Приложение предназначено для развлечения, развития интеллекта, совершенствования способности игры в шахматы.

Правила игры:

* + Игра проводится на доске размера 10x8;
  + Игра начинается с указания стороны, за которую играет игрок;
  + У игрока есть 15 шашек;
  + Ходы выполняются по очереди;
  + Игрок может передвигать только свои шашки;
  + Шашка может двигаться только на одну клетку по диагонали вперёд;
  + Дамка может двигаться вперёд и назад на любое количество клеток по диагонали;
  + Если шашка противника оказалась под ударом – бить обязательно;
  + Игрок оставшийся без шашек проигрывает.

В приложении реализованы: окно регистрации, модуль, который отрисовывает игру и проверяет на проигрыш\выигрыш\ничью, логика искусственного интеллекта, меню для создания новой партии или выхода из приложения

**1.2 Условия, необходимые для использования приложения**

1. Операционная система: Приложение должно быть доступно для Windows, macOS и Linux.

2. Платформа: Приложение должно быть написано на Python и совместимо с версиями Python 3 и выше.

3. Инструментальная среда: Для использования приложения необходимо установить Python и настроить его на компьютере пользователя.

4. Библиотеки: Приложение должно использовать библиотеки, такие как tkinter для создания графического интерфейса пользователя, а также стандартные библиотеки языка Python версии 3. и выше

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк: 570

Количество структур данных: 3(структура для хранения состояния шахматной доски и соответствующими методами поиска допустимых ходов, проверки доски на мат/пат/блокаду, добавления хода и другие, структура для хранения состояния доски, которая может прочитаться алгоритмом отрисовки интерфейса, структура данных для хранения логина и пароля, а также шифрования пароля)

Количество алгоритмов: 4(алгоритм для отрисовки игрового поля и возможных ходов, алгоритм поиска наилучшего хода для ИИ, алгоритм проверки на проигрыш/победу/ничью, алгоритм обработки хода игрока)

Описание используемых библиотек.

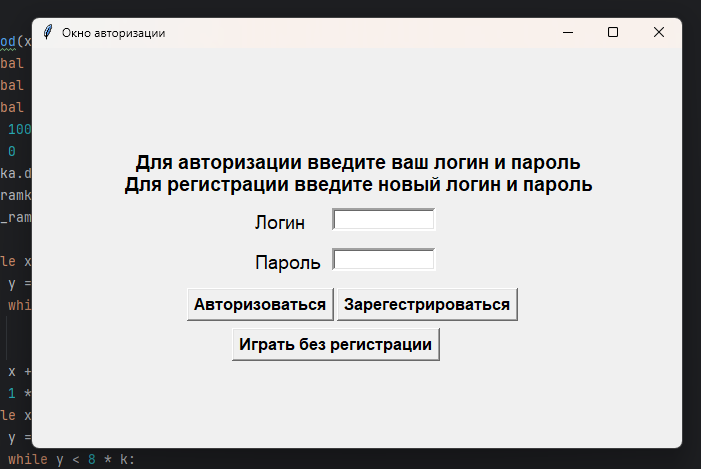
1. Библиотека tkinter: это стандартная библиотека для создания графического интерфейса пользователя (GUI) в Python. Она предоставляет различные виджеты и методы для создания окон, кнопок, полей ввода, меню и других элементов интерфейса. Tkinter обеспечивает простой способ взаимодействия с пользователем и создания приятного пользовательского опыта.

3. Библиотека os: это стандартная библиотека Python, предназначенная для работы с операционной системой, файлами и путями. Она предоставляет методы для управления файлами и директориями, проверки существования файлов, выполнения команд в командной строке, получения информации о системе и других операций, связанных с операционной системой. Библиотека os является мощным инструментом для автоматизации задач взаимодействия с файловой системой и операционной системой.

4. Библиотека random: это стандартная библиотека Python, которая предоставляет функции для генерации случайных чисел и выбора случайных элементов из последовательностей (списков, кортежей и т. д.). Она может быть использована для создания случайных данных, случайной генерации игровых элементов, тестирования и других случайных операций. Библиотека random предоставляет различные методы для генерации случайных чисел с разными распределениями и диапазонами значений.

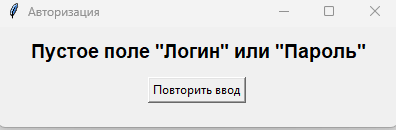
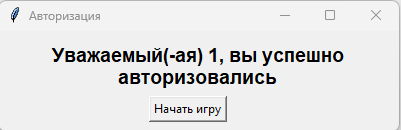
Описание работы приложения:

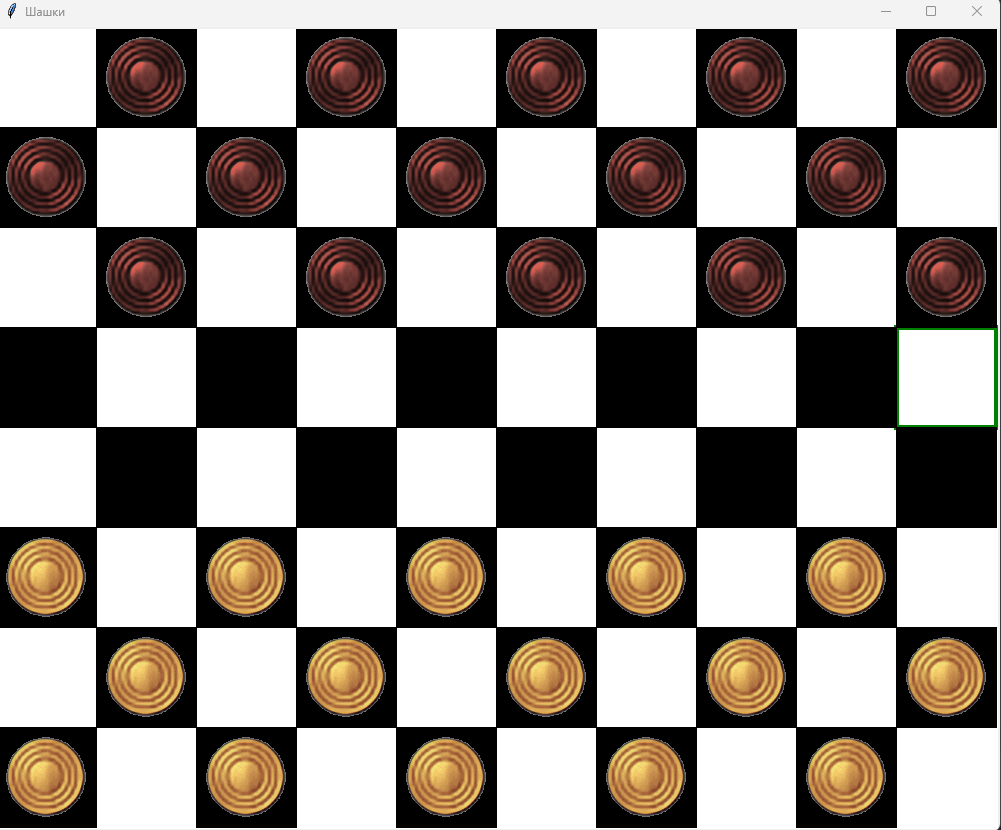
1.Изначально пользователь видит главное меню приложения в котором можно выбрать хочет ли он играть без регистрации или зарегистрироваться



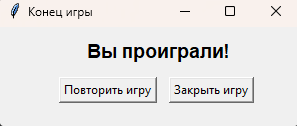
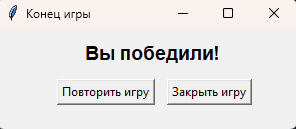
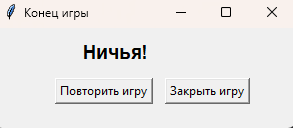
2.При нажатии на кнопку «Зарегистрироваться» открывается окно регистрации, при нажатии на кнопку «Играть без регистрации» открывается игра

3.В окне регистрации обрабатываются случаи правильного и неправильного ввода данных



 4. Открытие игры.

5. После окончания игры(три случая) открывается соответствующее меню в котором можно закрыть приложение или начать игру заново

**3. Обращение к программе**

init(self, board=ch.Board, depth\_lvl=1, root=tk.Tk): Это метод конструктора класса. Он инициализирует различные атрибуты, такие как шахматная доска, уровень глубины и корневой объект Tkinter.

draw\_board(self, root, c\_pc="", a1="normal"): Этот метод используется для рисования шахматной доски в графическом интерфейсе. Он принимает корневой объект Tkinter в качестве параметра и необязательно принимает текущую фигуру и состояние a1. Он создает кнопки для каждого квадрата на шахматной доске и присваивает им соответствующий текст и цвет фона в зависимости от текущего состояния игры.

Функция `register\_login()`:

- Уничтожает главное окно `root1`.

- Проверяет, существует ли файл "usersааа1.txt". Если да, создает экземпляр класса `RegistrationWindow` с этим файлом в качестве аргумента. Если нет, создает файл "usersааа1.txt" и затем создает экземпляр класса `RegistrationWindow` с этим файлом в качестве аргумента.

- Запускает метод `run()` экземпляра класса `RegistrationWindow` для отображения окна регистрации.

- Выводит в консоль результат метода `accesf()` экземпляра класса `RegistrationWindow`.

- Если результат метода `accesf()` равен `True`, вызывает функцию `hard()`.

Функция `play\_without\_reg()`:

- Уничтожает главное окно `root1`.

- Вызывает функцию `hard()`.

Методы, описанные в коде:

Методы из окна регистрации.

1. Метод `shifr(self, message1)`: Этот метод выполняет шифрование сообщения, принимая на вход строку `message1`. Внутри метода определены переменные `alfavit\_EU` (содержит алфавит в верхнем регистре), `smeshenie` (сдвиг для шифрования), `itog` (переменная для сохранения зашифрованного сообщения). Метод преобразует входную строку в верхний регистр, затем проходит по каждому символу и находит его индекс в `alfavit\_EU`. Затем происходит сдвиг индекса на значение `smeshenie` и добавление соответствующего символа в `itog`. Если символ не найден в алфавите, он остается без изменений. Метод выводит зашифрованное сообщение и возвращает его.

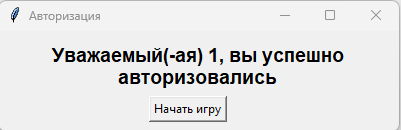
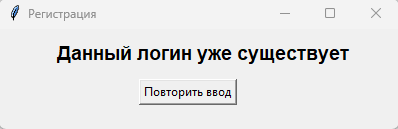
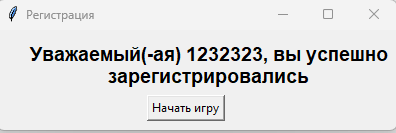
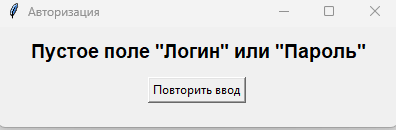
2. Метод `register(self)`: Этот метод выполняет регистрацию пользователя. Он получает логин (`username`) и пароль (`password1`) из соответствующих полей, а затем вызывает метод `shifr` для шифрования пароля. Если логин или пароль не указаны, метод выводит ошибку. Затем метод проверяет, есть ли уже такой пользователь в файле. Если пользователь уже существует, метод выводит ошибку. В противном случае, метод добавляет нового пользователя в файл и выводит сообщение об успешной регистрации.

3. Метод `run(self)`: Этот метод запускает основной цикл графического интерфейса пользователя (GUI).

4. Метод `accesf(self)`: Этот метод возвращает значение переменной `acces`, которая указывает на успешность авторизации пользователя.

**4. Сообщения**

Сообщения выдаваемые по результатам контроля корректности ввода/вывода.



Полный код программы:

from tkinter import \*

import os

import random

import time

import copy

def regis():

global txt, txtl, txtp, login, password, lab, b

def dismiss(win\_t):

win\_t.grab\_release()

win\_t.destroy()

def close\_all\_windows():

root.destroy()

s\_l = login.get()

s\_p = password.get()

if len(s\_l) == 0 or len(s\_p) == 0:

win = Toplevel(root, relief=SUNKEN)

win.geometry("400x100+730+420")

win.title("Регистрация")

win.minsize(width=400, height=100)

win.maxsize(width=400, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Пустое поле "Логин" или "Пароль"', font=("Brass Mono", 14, 'bold')).place(x=30, y=10)

close\_button = Button(win, text="Повторить ввод", command=lambda: dismiss(win))

close\_button.place(x=150, y=50)

win.grab\_set()

else:

file = open("1\_p.txt", "r+")

a = file.read().split()

m=0

for b in range(0, len(a), 2):

if a[b] == s\_l:

win = Toplevel(root, relief=SUNKEN)

win.geometry("400x100+730+420")

win.title("Регистрация")

win.minsize(width=400, height=100)

win.maxsize(width=400, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Данный логин уже существует', font=("Brass Mono", 14, 'bold')).place(x=55, y=10)

close\_button = Button(win, text="Повторить ввод", command=lambda: dismiss(win))

close\_button.place(x=140, y=50)

win.grab\_set()

m+=1

break

if m==0:

file.seek(0, os.SEEK\_END)

file.write(s\_l + ' ' + s\_p + ' ')

win\_r = Toplevel(root, relief=SUNKEN)

win\_r.geometry("400x100+730+420")

win\_r.title("Регистрация")

win\_r.minsize(width=400, height=100)

win\_r.maxsize(width=400, height=100)

win\_r.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win\_r))

Label(win\_r, text="Уважаемый(-ая) " + s\_l + ", вы успешно" + "\n" "зарегистрировались", font=("Arial", 14, 'bold')).place(x=30, y=10)

close\_button = Button(win\_r, text="Начать игру", command=close\_all\_windows)

close\_button.place(x=150, y=65)

win\_r.grab\_set()

file.close()

def avtoris():

global txt, txtl, txtp, login, password, lab, b

def close\_all\_windows():

root.destroy()

def dismiss(win\_t):

win\_t.grab\_release()

win\_t.destroy()

s\_l = login.get()

s\_p = password.get()

if len(s\_l) == 0 or len(s\_p) == 0:

win = Toplevel(root, relief=SUNKEN)

win.geometry("400x100+730+420")

win.title("Авторизация")

win.minsize(width=400, height=100)

win.maxsize(width=400, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Пустое поле "Логин" или "Пароль"', font=("Brass Mono", 14, 'bold')).place(x=30, y=10)

close\_button = Button(win, text="Повторить ввод", command=lambda: dismiss(win))

close\_button.place(x=150, y=50)

win.grab\_set()

else:

file = open("1\_p.txt", "r+")

a = file.read().split()

n=0

m=0

if len(a) == 0:

win = Toplevel(root, relief=SUNKEN)

win.geometry("400x100+730+420")

win.title("Авторизация")

win.minsize(width=500, height=100)

win.maxsize(width=500, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='В базе данных нет пользователей.' '\n' 'Поэтому вы можете только зарегестрироваться.', font=("Brass Mono", 14, 'bold')).place(x=10, y=10)

close\_button = Button(win, text="Повторить ввод", command=lambda: dismiss(win))

close\_button.place(x=190, y=70)

win.grab\_set()

else:

for b in range(0, len(a), 2):

if a[b] != s\_l:

n+=1

if n==len(a)/2:

win = Toplevel(root, relief=SUNKEN)

win.geometry("400x100+730+420")

win.title("Авторизация")

win.minsize(width=400, height=100)

win.maxsize(width=400, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Данного логина не существует', font=("Brass Mono", 14, 'bold')).place(x=30, y=10)

close\_button = Button(win, text="Повторить ввод", command=lambda: dismiss(win))

close\_button.place(x=150, y=50)

win.grab\_set()

m+=1

break

if m==0:

for j in range(0, len(a), 2):

n=0

if a[j] == s\_l and a[j + 1] == s\_p:

win\_r = Toplevel(root, relief=SUNKEN)

win\_r.geometry("400x100+730+420")

win\_r.title("Авторизация")

win\_r.minsize(width=400, height=100)

win\_r.maxsize(width=400, height=100)

win\_r.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win\_r))

Label(win\_r, text="Уважаемый(-ая) " + s\_l + ", вы успешно" + "\n" "авторизовались", font=("Arial", 14, 'bold')).place(x=50, y=10)

close\_button = Button(win\_r, text="Начать игру", command=close\_all\_windows)

close\_button.place(x=150, y=65)

win\_r.grab\_set()

n+=1

break

if n == 0:

for j in range(0, len(a), 2):

if a[j] != s\_l or a[j + 1] != s\_p:

win = Toplevel(root, relief=SUNKEN)

win.geometry("400x100+730+420")

win.title("Авторизация")

win.minsize(width=440, height=100)

win.maxsize(width=440, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Вы ввели неправильный логин или пароль', font=("Brass Mono", 14, 'bold')).place(x=10, y=10)

close\_button = Button(win, text="Повторить ввод", command=lambda: dismiss(win))

close\_button.place(x=160, y=50)

win.grab\_set()

break

file.close()

if not os.path.exists('1\_p.txt'):

def create\_file(filename):

with open('1\_p.txt', 'w'):

pass

create\_file("1\_p.txt")

def r():

root.destroy()

root = Tk()

root.geometry("650x400+500+50")

root.title('Окно авторизации')

s\_l, s\_p = 'name', 'pas'

txt = Label(root, text="Для авторизации введите ваш логин и пароль\nДля регистрации введите новый логин и пароль", font=("Arial", 14, 'bold'))

txtl = Label(root, text="Логин", font=("Brass Mono", 14))

login = Entry(root, width=16, bd=3, textvariable=s\_l)

txtp = Label(root, text="Пароль", font=("Brass Mono", 14))

password = Entry(root, width=16, bd=3, textvariable=s\_p)

close\_but = Button(root, text="Авторизоваться", font=("Arial", 12, 'bold'), command=avtoris)

close\_but1 = Button(root, text="Зарегестрироваться", font=("Arial", 12, 'bold'), command=regis)

close\_but2 = Button(root, text="Играть без регистрации", font=("Arial", 12, 'bold'), command=r)

txt.place(x=90, y=100)

txtl.place(x=220, y=160)

login.place(x=300, y=160)

txtp.place(x=220, y=200)

password.place(x=300, y=200)

close\_but.place(x=155, y=240)

close\_but1.place(x=305, y=240)

close\_but2.place(x=200, y=280)

mainloop()

def dismiss(win\_t):

win\_t.grab\_release()

win\_t.destroy()

def close\_all\_windows():

gl\_okno.destroy()

gl\_okno = Tk() # создаём окно

gl\_okno.title('Шашки') # заголовок окна

doska = Canvas(gl\_okno, width=1000, height=800, bg='#FFFFFF')

doska.pack()

n2\_spisok = ()# конечный список ходов компьютера

ur = 2 # количество предсказываемых компьютером ходов

k\_rez = 0 # !!!

o\_rez = 0

poz1\_x = -1 # клетка не задана

f\_hi = True # определение хода игрока(да)

def izobrazheniya\_peshek(): # загружаем изображения пешек

global peshki

i1 = PhotoImage(file="res\\1b.gif")

i2 = PhotoImage(file="res\\1bk.gif")

i3 = PhotoImage(file="res\\1h.gif")

i4 = PhotoImage(file="res\\1hk.gif")

peshki = [0, i1, i2, i3, i4]

def novaya\_igra(): # начинаем новую игру

global pole

pole = [[0, 3, 0, 3, 0, 3, 0, 3, 0, 3],

[3, 0, 3, 0, 3, 0, 3, 0, 3, 0],

[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],

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

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

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

def vivod(x\_poz\_1, y\_poz\_1, x\_poz\_2, y\_poz\_2): # рисуем игровое поле

global peshki

global pole

global kr\_ramka, zel\_ramka

k = 100

x = 0

doska.delete('all')

kr\_ramka = doska.create\_rectangle(-5, -5, -5, -5, outline="red", width=5)

zel\_ramka = doska.create\_rectangle(-5, -5, -5, -5, outline="green", width=5)

while x < 10 \* k: # рисуем доску

y = 1 \* k

while y < 8 \* k:

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

y += 2 \* k

x += 2 \* k

x = 1 \* k

while x < 10 \* k: # рисуем доску

y = 0

while y < 8 \* k:

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

y += 2 \* k

x += 2 \* k

for y in range(8): # рисуем стоячие пешки

for x in range(10):

z = pole[y][x]

if z:

if (x\_poz\_1, y\_poz\_1) != (x, y): # стоячие пешки?

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

# рисуем активную пешку

z = pole[y\_poz\_1][x\_poz\_1]

if z: # ???

doska.create\_image(x\_poz\_1 \* k, y\_poz\_1 \* k, anchor=NW, image=peshki[z], tag='ani')

# вычисление коэф. для анимации

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

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

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

for ii in range(33):

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

doska.update() # обновление

time.sleep(0.01)

def soobsenie(s):

global f\_hi, i

z = 'Игра завершена'

if s == 1:

win = Tk()

win.geometry("400x100+730+420")

win.title("Конец игры")

win.minsize(width=300, height=100)

win.maxsize(width=300, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Вы победили!', font=("Brass Mono", 14, 'bold')).place(x=85, y=10)

close\_button1 = Button(win, text="Повторить игру", command=lambda: s())

close\_button3 = Button(win, text="Закрыть игру", command=lambda: s1())

close\_button1.place(x=60, y=50)

close\_button3.place(x=170, y=50)

win.grab\_set()

if s == 2:

win = Tk()

win.geometry("400x100+730+420")

win.title("Конец игры")

win.minsize(width=300, height=100)

win.maxsize(width=300, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Вы проиграли!', font=("Brass Mono", 14, 'bold')).place(x=85, y=10)

close\_button1 = Button(win, text="Повторить игру", command=lambda: s())

close\_button3 = Button(win, text="Закрыть игру", command=lambda: s1())

close\_button1.place(x=60, y=50)

close\_button3.place(x=170, y=50)

win.grab\_set()

if s == 3:

win = Tk()

win.geometry("400x100+730+420")

win.title("Конец игры")

win.minsize(width=300, height=100)

win.maxsize(width=300, height=100)

win.protocol("WM\_DELETE\_WINDOW", lambda: dismiss(win))

Label(win, text='Ничья!', font=("Brass Mono", 14, 'bold')).place(x=85, y=10)

close\_button1 = Button(win, text="Повторить игру", command=lambda: s())

close\_button3 = Button(win, text="Закрыть игру", command=lambda: s1())

close\_button1.place(x=60, y=50)

close\_button3.place(x=170, y=50)

win.grab\_set()

def s1():

win.destroy()

close\_all\_windows()

def s():

win.destroy()

novaya\_igra()

vivod(-1, -1, -1, -1) # рисуем игровое поле

f\_hi = True # ход игрока доступен

def pozici\_1(event): # выбор клетки для хода 1

x, y = (event.x) // 100, (event.y) // 100 # вычисляем координаты клетки

doska.coords(zel\_ramka, x \* 100, y \* 100, x \* 100 + 100, y \* 100 + 100) # рамка в выбранной клетке

def pozici\_2(event): # выбор клетки для хода 2

global poz1\_x, poz1\_y, poz2\_x, poz2\_y

global f\_hi

x, y = (event.x) // 100, (event.y) // 100 # вычисляем координаты клетки

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

doska.coords(kr\_ramka, x \* 100, y \* 100, x \* 100 + 100, y \* 100 + 100) # рамка в выбранной клетке

poz1\_x, poz1\_y = x, y

else:

if poz1\_x != -1: # клетка выбрана

poz2\_x, poz2\_y = x, y

if f\_hi: # ход игрока?

hod\_igroka()

if not (f\_hi):

time.sleep(0.5)

hod\_kompjutera() # передаём ход компьютеру

# gl\_okno.after(500,hod\_kompjutera(0))#!!!#передаём ход компьютеру

poz1\_x = -1 # клетка не выбрана

doska.coords(kr\_ramka, -5, -5, -5, -5) # рамка вне поля

def hod\_kompjutera(): # !!!

global f\_hi

global n2\_spisok

proverka\_hk(1, (), [])

if n2\_spisok: # проверяем наличие доступных ходов

kh = len(n2\_spisok) # количество ходов

th = random.randint(0, kh - 1) # случайный ход

dh = len(n2\_spisok[th]) # длина хода

for h in n2\_spisok: # !!!для отладки!!!

h = h # !!!для отладки!!!

for i in range(dh - 1):

# выполняем ход

spisok = hod(1, n2\_spisok[th][i][0], n2\_spisok[th][i][1], n2\_spisok[th][1 + i][0], n2\_spisok[th][1 + i][1])

n2\_spisok = [] # очищаем список ходов

f\_hi = True # ход игрока доступен

# определяем победителя

s\_k, s\_i = skan()

if not (s\_i):

soobsenie(2)

elif not (s\_k):

soobsenie(1)

elif f\_hi and not (spisok\_hi()):

soobsenie(3)

elif not (f\_hi) and not (spisok\_hk()):

soobsenie(3)

def spisok\_hk(): # составляем список ходов компьютера

spisok = prosmotr\_hodov\_k1([]) # здесь проверяем обязательные ходы

if not (spisok):

spisok = prosmotr\_hodov\_k2([]) # здесь проверяем оставшиеся ходы

return spisok

def proverka\_hk(tur, n\_spisok, spisok): # !!!

global pole

global n2\_spisok

global l\_rez, k\_rez, o\_rez

if not (spisok): # если список ходов пустой...

spisok = spisok\_hk() # заполняем

if spisok:

k\_pole = copy.deepcopy(pole) # копируем поле

for ((poz1\_x, poz1\_y), (poz2\_x, poz2\_y)) in spisok: # проходим все ходы по списку

t\_spisok = hod(0, poz1\_x, poz1\_y, poz2\_x, poz2\_y)

if t\_spisok: # если существует ещё ход

proverka\_hk(tur, (n\_spisok + ((poz1\_x, poz1\_y),)), t\_spisok)

else:

proverka\_hi(tur, [])

if tur == 1:

t\_rez = o\_rez / k\_rez

if not (n2\_spisok): # записыаем если пустой

n2\_spisok = (n\_spisok + ((poz1\_x, poz1\_y), (poz2\_x, poz2\_y)),)

l\_rez = t\_rez # сохряняем наилучший результат

else:

if t\_rez == l\_rez:

n2\_spisok = n2\_spisok + (n\_spisok + ((poz1\_x, poz1\_y), (poz2\_x, poz2\_y)),)

if t\_rez > l\_rez:

n2\_spisok = ()

n2\_spisok = (n\_spisok + ((poz1\_x, poz1\_y), (poz2\_x, poz2\_y)),)

l\_rez = t\_rez # сохряняем наилучший результат

o\_rez = 0

k\_rez = 0

pole = copy.deepcopy(k\_pole) # возвращаем поле

else: # ???

s\_k, s\_i = skan() # подсчёт результата хода

o\_rez += (s\_k - s\_i)

k\_rez += 1

def spisok\_hi(): # составляем список ходов игрока

spisok = prosmotr\_hodov\_i1([]) # здесь проверяем обязательные ходы

if not (spisok):

spisok = prosmotr\_hodov\_i2([]) # здесь проверяем оставшиеся ходы

return spisok

def proverka\_hi(tur, spisok):

global pole, k\_rez, o\_rez

global ur

if not (spisok):

spisok = spisok\_hi()

if spisok: # проверяем наличие доступных ходов

k\_pole = copy.deepcopy(pole) # копируем поле

for ((poz1\_x, poz1\_y), (poz2\_x, poz2\_y)) in spisok:

t\_spisok = hod(0, poz1\_x, poz1\_y, poz2\_x, poz2\_y)

if t\_spisok: # если существует ещё ход

proverka\_hi(tur, t\_spisok)

else:

if tur < ur:

proverka\_hk(tur + 1, (), [])

else:

s\_k, s\_i = skan() # подсчёт результата хода

o\_rez += (s\_k - s\_i)

k\_rez += 1

pole = copy.deepcopy(k\_pole) # возвращаем поле

else: # доступных ходов нет

s\_k, s\_i = skan() # подсчёт результата хода

o\_rez += (s\_k - s\_i)

k\_rez += 1

def skan(): # подсчёт пешек на поле

global pole

s\_i = 0

s\_k = 0

for i in range(8):

for ii in pole[i]:

if ii == 1: s\_i += 1

if ii == 2: s\_i += 3

if ii == 3: s\_k += 1

if ii == 4: s\_k += 3

return s\_k, s\_i

def hod\_igroka():

global poz1\_x, poz1\_y, poz2\_x, poz2\_y

global f\_hi

f\_hi = False # считаем ход игрока выполненным

spisok = spisok\_hi()

if spisok:

if ((poz1\_x, poz1\_y), (poz2\_x, poz2\_y)) in spisok: # проверяем ход на соответствие правилам игры

t\_spisok = hod(1, poz1\_x, poz1\_y, poz2\_x, poz2\_y) # если всё хорошо, делаем ход

if t\_spisok: # если есть ещё ход той же пешкой

f\_hi = True # считаем ход игрока невыполненным

else:

f\_hi = True # считаем ход игрока невыполненным

doska.update() # !!!обновление

def hod(f, poz1\_x, poz1\_y, poz2\_x, poz2\_y):

global pole

if f: vivod(poz1\_x, poz1\_y, poz2\_x, poz2\_y) # рисуем игровое поле

# превращение

if poz2\_y == 0 and pole[poz1\_y][poz1\_x] == 1:

pole[poz1\_y][poz1\_x] = 2

# превращение

if poz2\_y == 7 and pole[poz1\_y][poz1\_x] == 3:

pole[poz1\_y][poz1\_x] = 4

# делаем ход

pole[poz2\_y][poz2\_x] = pole[poz1\_y][poz1\_x]

pole[poz1\_y][poz1\_x] = 0

# рубим пешку игрока

kx = ky = 1

if poz1\_x < poz2\_x: kx = -1

if poz1\_y < poz2\_y: ky = -1

x\_poz, y\_poz = poz2\_x, poz2\_y

while (poz1\_x != x\_poz) or (poz1\_y != y\_poz):

x\_poz += kx

y\_poz += ky

if pole[y\_poz][x\_poz] != 0:

pole[y\_poz][x\_poz] = 0

if f: vivod(-1, -1, -1, -1) # рисуем игровое поле

# проверяем ход той же пешкой...

if pole[poz2\_y][poz2\_x] == 3 or pole[poz2\_y][poz2\_x] == 4: # ...компьютера

return prosmotr\_hodov\_k1p([], poz2\_x, poz2\_y) # возвращаем список доступных ходов

elif pole[poz2\_y][poz2\_x] == 1 or pole[poz2\_y][poz2\_x] == 2: # ...игрока

return prosmotr\_hodov\_i1p([], poz2\_x, poz2\_y) # возвращаем список доступных ходов

if f: vivod(poz1\_x, poz1\_y, poz2\_x, poz2\_y) # рисуем игровое поле

def prosmotr\_hodov\_k1(spisok): # проверка наличия обязательных ходов

for y in range(8): # сканируем всё поле

for x in range(10):

spisok = prosmotr\_hodov\_k1p(spisok, x, y)

return spisok

def prosmotr\_hodov\_k1p(spisok, x, y):

if pole[y][x] == 3: # пешка

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

if 0 <= y + iy + iy <= 7 and 0 <= x + ix + ix <= 9:

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

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

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

if pole[y][x] == 4: # пешка с короной

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

osh = 0 # определение правильности хода

for i in range(1, 10):

if 0 <= y + iy \* i <= 7 and 0 <= x + ix \* i <= 9:

if osh == 1:

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

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

osh += 1

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

if osh > 0: spisok.pop() # удаление хода из списка

break

return spisok

def prosmotr\_hodov\_k2(spisok): # проверка наличия остальных ходов

for y in range(8): # сканируем всё поле

for x in range(10):

if pole[y][x] == 3: # пешка

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

if 0 <= y + iy <= 7 and 0 <= x + ix <= 9:

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

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

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

if 0 <= y + iy \* 2 <= 7 and 0 <= x + ix \* 2 <= 9:

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

spisok.append(((x, y), (

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

if pole[y][x] == 4: # пешка с короной

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

osh = 0 # определение правильности хода

for i in range(1, 10):

if 0 <= y + iy \* i <= 7 and 0 <= x + ix \* i <= 9:

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

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

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

osh += 1

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

break

return spisok

def prosmotr\_hodov\_i1(spisok): # проверка наличия обязательных ходов

spisok = [] # список ходов

for y in range(8): # сканируем всё поле

for x in range(10):

spisok = prosmotr\_hodov\_i1p(spisok, x, y)

return spisok

def prosmotr\_hodov\_i1p(spisok, x, y):

if pole[y][x] == 1: # пешка

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

if 0 <= y + iy + iy <= 7 and 0 <= x + ix + ix <= 9:

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

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

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

if pole[y][x] == 2: # пешка с короной

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

osh = 0 # определение правильности хода

for i in range(1, 10):

if 0 <= y + iy \* i <= 7 and 0 <= x + ix \* i <= 9:

if osh == 1:

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

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

osh += 1

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

if osh > 0: spisok.pop() # удаление хода из списка

break

return spisok

def prosmotr\_hodov\_i2(spisok): # проверка наличия остальных ходов

for y in range(8): # сканируем всё поле

for x in range(10):

if pole[y][x] == 1: # пешка

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

if 0 <= y + iy <= 7 and 0 <= x + ix <= 9:

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

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

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

if 0 <= y + iy \* 2 <= 7 and 0 <= x + ix \* 2 <= 7:

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

spisok.append(((x, y), (

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

if pole[y][x] == 2: # пешка с короной

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

osh = 0 # определение правильности хода

for i in range(1, 10):

if 0 <= y + iy \* i <= 7 and 0 <= x + ix \* i <= 9:

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

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

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

osh += 1

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

break

return spisok

izobrazheniya\_peshek() # здесь загружаем изображения пешек

novaya\_igra() # начинаем новую игру

vivod(-1, -1, -1, -1) # рисуем игровое поле

doska.bind("<Motion>", pozici\_1) # движение мышки по полю

doska.bind("<Button-1>", pozici\_2) # нажатие левой кнопки

mainloop()