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 «Русские 80-клеточные шашки»

Пояснительная записка

Р.02069337. 21/849-01 ПЗ-01

Листов 8

Руководитель разработки:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2024 г.

Исполнитель:

студент гр. ИСТбд-32

*Абляев Глеб Сергеевич*

« » 2024 г.

2024

Содержание

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

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

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

Руководство программиста…………………………………………………….21

Текст программы……………………………………………………………….27

Аннотация.

Данная курсовая работа посвящена разработке программы компьютерных русских 80-клеточных шашек. Цель работы состоит в создании программного приложения, способного играть в шашки на стандартной доске 10x8 клеток с использованием классических правил и стратегий игры. Для достижения этой цели были проведены исследования по анализу правил игры, разработаны алгоритмы для выбора оптимальных ходов, а также реализована графическая интерфейсная часть для взаимодействия пользователя с программой. В результате работы созданная программа позволяет пользователям играть в русские 80-клеточные шашки как против компьютера, так и против других игроков, обеспечивая при этом удобство и эффективность игрового процесса.

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 «Русские 80-клеточные шашки»

Р.02069337. 21/849-01 ТЗ-01

Листов 5

Исполнитель:

студент гр. ИСТбд-32

*Абляев Глеб Сергеевич*

« » 2024

2024

**Введение**

Игра “Русские 80-клеточные шашки”

80-Клеточные шашки ("Шашки на 80 квадратов" - 80-клеточные шашки), также известные как Советские шашки ("Советские шашки" - советские шашки), представляют собой современную разновидность [русских шашек](https://translated.turbopages.org/proxy_u/en-ru.ru.743e42dc-65f6f238-b07b2d34-74722d776562/https/checkers.fandom.com/wiki/Russkie_Shashki), которая была изобретена в России около 1930 года Николаем Спанцирети (1916-1991).

Функциональные возможности приложения: авторизация, регистрация, игра в киммерийские шашки.

**1. Основания для разработки**

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

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

**2.1. Функциональное назначение**

Приложение создано для игры в «Русские 80-клеточные шашки». Данное приложение должно иметь графический интерфейс. Необходим личный кабинет пользователя: авторизация, регистрация. Данные пользователя хранить в файле, БД не нужно.

**2.2 Требования к функциональным характеристикам**

2.2.1 Требования к структуре приложения

Требования к графическому и пользовательскому интерфейсам:

1. программа должна работать в графическом режиме;
2. в программе должна использоваться мышка для перемещения шашек;
3. программа должна иметь анимацию движения шашек.

В данной работе использовались библиотеки:

1. Pygame\_menu – для создания и отображения меню
2. Pygame – для создания шахматной доски, меню и работе со звуками
3. tkinter - для отрисовки доски и шашек.

2.2.2 Требования к составу функций приложения

Приложение включает в себя следующие классы:

1. Piece – представляет из себя класс пешки
2. Board – формирует игровую доску
3. Game – задает игру, правила игры, работает с доской

Приложение включает в себя следующие функции:

1. Регистрация / авторизация пользователя.
2. Проверка логина и пароля.
3. Шифрование логина и пароля.
4. Возможность зашифровать любой текст из личного кабинета, длиной от 1 до 16 символов.
5. Проверять правильность хода.
6. Выявлять победителя.
7. Начать игру заново.

Приложение включает в себя следующие алгоритмы:

1. Проверка регистрации пользователя.
2. Проверка на авторизированных пользователей.
3. Проверка корректности логина и пароля.
4. Шифрование логина и пароля.
5. Начало новой игры.
6. Генерация поля.
7. Проверка на победу.
8. Реализация ходов.

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

* Пользовательский интерфейс – графический.
* При регистрации, поля ввода не должны быть пустыми.
* Логин и пароль должны состоять из латинских букв и цифр в количестве от 1 до 16.
* Входные данные при регистрации должны шифроваться и записываться в текстовый файл.

**2.3 Требования к надежности**

Необходимо, чтобы входные данные можно было бы зашифровать с помощью Цезаря, то есть они должны быть либо в латинском алфавите, либо одной из цифр.

**2.4 Требования к информационной и программной совместимости**

1. ОС: Windows 10 21H2.
2. Среда разработки: PyCharm Community Edition Version: 2022.3.
3. Версия языка: Python 3.10.
4. Библиотеки: pygame 2.1.2, pygame-menu 4.3.6, Tkinter.

**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

Условия хранения для диска CD-R следующие:

температура – от 5°C до 20°C.

влажность – от 30% до 50%.

2.6 3 Сроки хранения

Срок хранения – до июля 2023 года.

**3. Требования к программной документации**

Определяются заданием на курсовую работу.

**4. Стадии и этапы разработки**

Определяются заданием на курсовую работу.

**5. Порядок контроля и приёмки**

Определяются заданием на курсовую работу.

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Измерительно-вычислительные комплексы»

**Курсовая работа**

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

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

**Пояснительная записка**

Р.02069337. 21/849-01 ПЗ-01

Листов 12

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2024 г.

**Исполнитель**:

студент гр. ИСТбд-32

*Абляев Глеб Сергеевич*

« » 2024 г.

**2024**

**Введение**

Реализована игра «Русские шахматы»

Шашки — это абстрактная стратегическая настольная игра для двух игроков, играющих на доске с клетками, состоящей из чередующихся светлых и темных квадратных ячеек. Целью игры является захват или блокировка всех шашек противника.

Игра шашки имеет древнее происхождение и широко распространена по всему миру. Она предоставляет игрокам возможность развивать стратегическое мышление, тактику и аналитические способности.

Шашки включают в себя не только базовые правила, но и сложные тактические методы и стратегии, такие как создание дамок, блокировка противника, проведение комбинаций и атака на ключевые позиции на игровом поле.

Программирование игры шашки не только демонстрирует понимание основных концепций игры и алгоритмов, но и представляет собой отличный способ практического применения знаний в области компьютерной графики, пользовательского интерфейса и разработки искусственного интеллекта.

1. **Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Целью проекта является разработка приложения для игры в шашки. Основная задача заключается в создании программного продукта, который позволит пользователям играть в шашки против компьютера или друг против друга. Техническое задание на разработку приложения подробно описано в соответствующем документе.

**1.2 Математические методы**

В данном проекте математические методы не применяются, поскольку основной акцент делается на разработку игровой логики, пользовательского интерфейса и искусственного интеллекта для компьютерного противника.

**1.3 Архитектура и алгоритмы**

1.3.1 Архитектура

В архитектуре приложения выделяются следующие основные компоненты:

Piece (Шашка): Представляет из себя пешку на игровом поле. Содержит информацию о своих координатах как относительно игровой матрицы, так и относительно окна. Также включает функции для пересчета координат и отрисовки шашки, а также функцию для выполнения хода (изменение координат).

Board (Доска): Содержит игровую матрицу (двумерный список), представляющую игровое поле. Включает функции для создания доски, отрисовки её на экране, выполнения хода шашки (изменения её координат в матрице), поиска пешек, которыми обязательно нужно ходить, поиска обязательных ходов, поиска необязательных ходов и определения победителя.

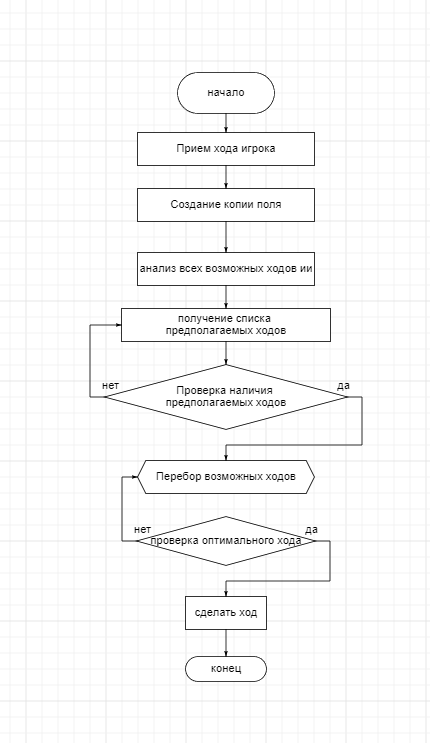
Game (Игра): Содержит информацию о текущем ходе, объект доски, доступные ходы и пользовательский интерфейс (GUI). Включает функции для определения победителя, выбора клетки для хода, смены очереди игрока и выполнения хода шашки.

Архитектура приложения предполагает четкое разделение ответственностей между компонентами, что позволяет обеспечить более гибкую и поддерживаемую структуру кода.

Алгоритм Минимакс

Алгоритм Минимакс - это классический алгоритм принятия решений в играх, основанный на принципе "минимум-максимум". Алгоритм широко применяется в задачах искусственного интеллекта для поиска оптимального хода в играх с нулевой суммой, таких как шашки, шахматы и другие.

Принцип работы алгоритма минимакс заключается в том, что он рекурсивно оценивает возможные ходы игроков на несколько ходов вперед, чтобы определить наилучший ход для текущего игрока и наихудший для его противника. Для этого алгоритм использует дерево возможных игровых состояний, где каждый узел представляет собой конкретное состояние игры, а рёбра - возможные ходы.



Алгоритм минимакс работает в двух режимах:

Максимизация: когда алгоритм выбирает ход для максимизации выигрыша текущего игрока.

Минимизация: когда алгоритм выбирает ход для минимизации потерь противника.

На каждом уровне рекурсии алгоритм переключается между этими двумя режимами, чтобы просчитать все возможные ходы и выбрать наилучший из них. По мере расширения дерева игровых состояний, алгоритм оценивает каждый возможный ход с помощью функции оценки (эвристики), которая оценивает выигрыш для текущего игрока.

1.3.4 Алгоритм создания доски

Этот алгоритм ответственен за создание игровой доски, её отрисовку и заполнение. Он создаёт двумерный массив (матрицу), представляющий собой игровое поле, и заполняет его начальным расположением шашек. Затем алгоритм отрисовывает доску в графическом интерфейсе пользователя (GUI), обеспечивая правильное отображение на экране.

1.3.5 Алгоритм поиска обязательных ходов

Этот алгоритм осуществляет поиск обязательных ходов для определенной шашки. Он анализирует текущее состояние игры и определяет, есть ли у какой-либо из шашек обязательные ходы, которые необходимо выполнить в текущем ходе. Для этого алгоритм проверяет возможные ходы каждой шашки и выявляет те из них, которые обязательны для выполнения.

1.3.6 Алгоритм поиска пешек с обязательными ходами

Этот алгоритм осуществляет поиск шашек, у которых есть обязательные ходы. Он проходит по всем пешкам на игровой доске и определяет, есть ли у них обязательные ходы. Если такие ходы обнаруживаются, шашки добавляются в список, который затем используется для выполнения обязательных ходов.

1.3.7 Алгоритм поиска всех пешек

Этот алгоритм осуществляет поиск всех шашек на игровой доске. Он проходит по всем клеткам игрового поля и определяет, находится ли на каждой из них шашка определенного игрока. Затем он составляет список всех найденных шашек для дальнейшего использования в других алгоритмах.

1.3.8 Алгоритм поиска ходов пешки

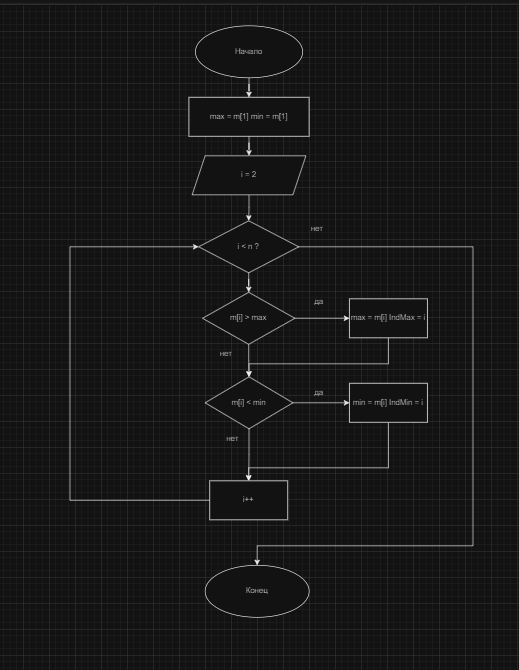
Этот алгоритм осуществляет поиск всех возможных ходов для определенной шашки, включая необязательные ходы, то есть те, которые не предполагают битья других шашек. Он анализирует текущее положение пешки на доске и определяет все возможные ходы в рамках правил игры.

1.3.9 Алгоритм выбора шашки/хода шашки

Этот алгоритм учитывает обязательные ходы и помогает игроку выбрать шашку, которой нужно сделать ход. Если у игрока есть возможность сделать обязательный ход, этот алгоритм помогает определить, какую шашку следует использовать для выполнения этого хода.

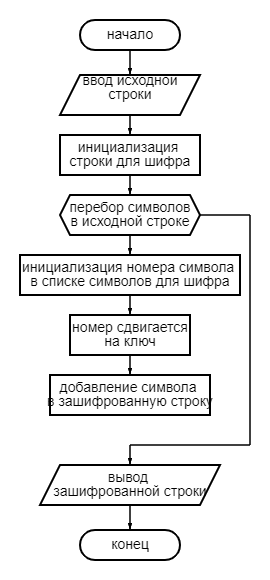
1.3.10 Алгоритм хода шашки

Этот алгоритм осуществляет перемещение пешки по игровой доске в соответствии с выбранным ходом. Он меняет положение шашки, удаляет битые шашки противника (если таковые есть) и переключает ход к следующему игроку.



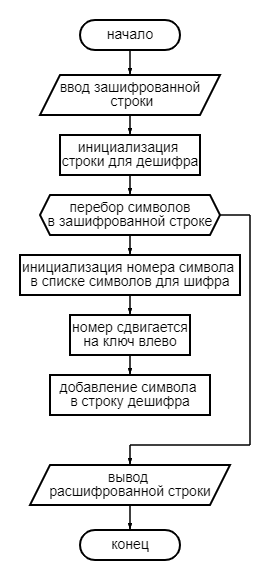
1.3.11 Алгоритм шифрования Цезаря

Алгоритм реализует шифрование пароля при регистрации.



1.3.11 Алгоритм дешифрования Цезаря

Алгоритм реализует дешифрование пароля при регистрации.



1.4 Тестирование

Для проверки реализованного приложения было проведено ручное тестирование. Исходный код был также проверен на наличие неиспользуемых переменных, ошибок, связанных с логикой игры, а также на ошибки форматирования введенных данных пользователем.

Для тестирования были выбраны два распространенных сценария, которые часто приводят к ошибкам:

Сценарий: Неверный ввод данных пользователем.

Дейрезультат: Пользователь получит сообщение "INCORRECT LOGIN OR PASSWORD."ствие пользователя: Ввод пустых данных.

Ожидаемый

Фактический результат: Пользователь получает сообщение "INCORRECT LOGIN OR PASSWORD."

Сценарий: Попытка авторизации несуществующего пользователя.

Действие пользователя: Попытка авторизации с несуществующим аккаунтом.

Ожидаемый результат: Пользователь получит сообщение "INCORRECT LOGIN OR PASSWORD."

Фактический результат: Пользователь получает сообщение "INCORRECT LOGIN OR PASSWORD."

**2. Источники, использованные при разработке**

1. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с. ( Дата обращения: 17.03.2024)
2. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с. ( Дата обращения: 13.03.2024)
3. Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с. ( Дата обращения: 15.03.2024)
4. Лучано Рамальо Python. К вершинам мастерства. – М.: ДМК Пресс, 2016. – 768 с. ( Дата обращения: 17.03.2024)

**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 «Русские 80-клеточные шашки»** 

**Руководство программиста**

Р.02069337. 21/849-01 РП-01

Листов 7

**Исполнитель**:

студент гр. ИСТбд-32

*Абляев Глеб Сергеевич*

« » 2024 г.

**2024**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Игра “Русские 80-клеточные шашки”

80-Клеточные шашки ("Шашки на 80 квадратов" - 80-клеточные шашки), также известные как Советские шашки ("Советские шашки" - советские шашки), представляют собой современную разновидность [русских шашек](https://translated.turbopages.org/proxy_u/en-ru.ru.743e42dc-65f6f238-b07b2d34-74722d776562/https/checkers.fandom.com/wiki/Russkie_Shashki), которая была изобретена в России около 1930 года Николаем Спанцирети (1916-1991).

Функциональные возможности приложения: авторизация, регистрация, игра в киммерийские шашки.

**1.2 Условия, необходимые для использования приложения**

1. ОС: Windows 10 21H2.
2. Среда разработки: PyCharm Community Edition Version: 2022.3.
3. Версия языка: Python 3.10.
4. Библиотеки: pygame 2.1.2, tkinter, checkers.

В качестве БД использовать файл.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк кода – 667.

Количество алгоритмов – 28.

Количество методов 24.

Запуск программы производится двойным щелчком мыши на файле «main.exe» или на его ярлыке.

При запуске приложения появляется окно регистрации/авторизации пользователя и выхода (Рис. 1). Логин и пароль должны состоять из набора латинских букв и цифр, с длиной символов от 1 до 16. В случае введения некорректных данных на окне появляются замечания.

Появляется окно регистрации, в котором требуется ввести пароль и имя пользователя.(Рис.1)

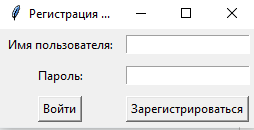


Рис.1

После по нажатию кнопки зарегистрироваться произойдет регистрация пользователя с записью данных в текстовый файл, также появится окно об успешной регистрации. (Рис.2)

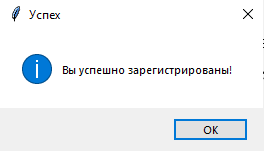


Рис.2

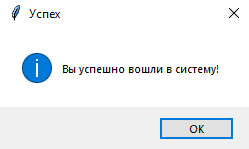
После этого пользователь вводит свои данные в окно еще раз и нажимает войти, появляется окно об успешном заходе в систему. (Рис.3) 

Рис.3

На рис. 4 представлен геймплей игры. Пользователь по нажатию на свои шашки может совершать ходы в соответствии с правилами игры.

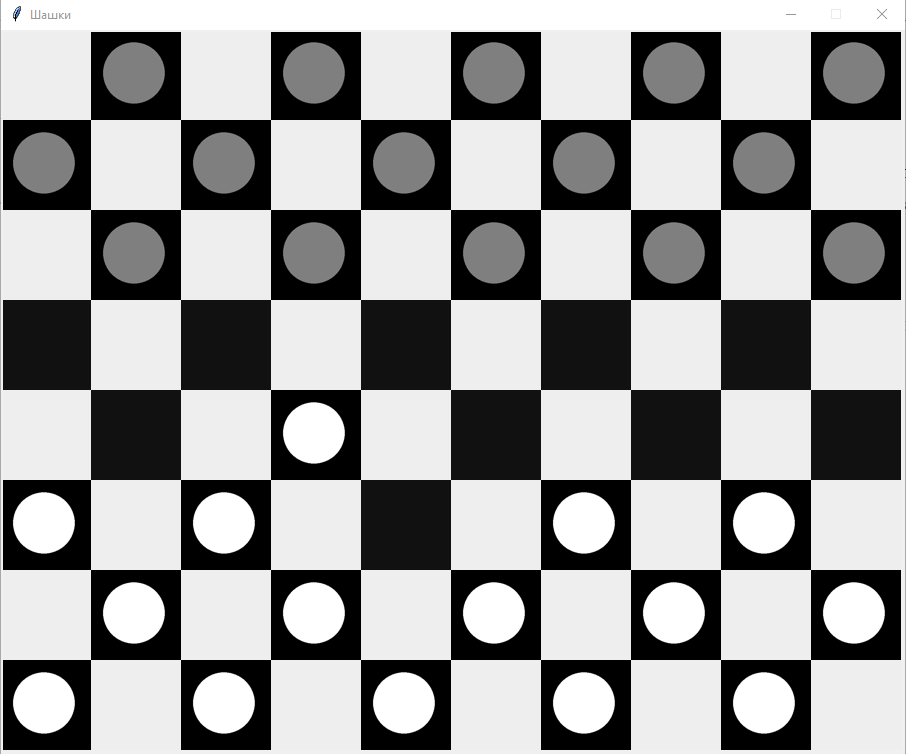


Рис.4

Результат игры можно увидеть на рис.5. По окончанию игры появляется окно с надписью о победе черных или белых.

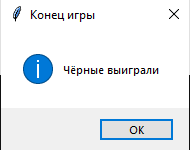


Рис.5

**2.2 Особенности реализации приложения**

В программе используются массивы, отвечающие за наименование координат игрового поля и для проверки победителя, правильности ходов, работы компьютера.

Приложение используются библиотеки:

* tkinter-использовалась для отрисовки шахматной доски и меню.

**3. Обращение к программе**

В классе Game:

\_\_init\_\_() : Конструктор класса, инициализирует игровое поле, изображения шашек, устанавливает очередь игрока. Если игрок играет за черных, совершает ход противника.

\_\_init\_images(self): Инициализирует изображения шашек, загружая их из файлов.

\_\_animate\_move(self, move: Move): Анимирует ход шашки на игровом поле.

\_\_draw(self): Очищает холст и перерисовывает игровое поле и шашки.

\_\_draw\_field\_grid(self): Рисует сетку игрового поля.

\_\_draw\_checkers(self): Рисует шашки на игровом поле.

\_\_handle\_player\_turn(self, move: Move): Обрабатывает ход игрока.

\_\_handle\_enemy\_turn(self): Обрабатывает ход противника.

\_\_check\_for\_game\_over(self): Проверяет, не завершилась ли игра, и выводит сообщение о победе одной из сторон.

Класс Checker:

\_\_init\_\_(): Конструктор класса. Принимает размеры игрового поля по оси X и по оси Y и инициализирует поле.

x\_size(self) -> int: Свойство, возвращающее размер игрового поля по оси X.

y\_size(self) -> int: Свойство, возвращающее размер игрового поля по оси Y.

size(self) -> int: Метод, возвращающий максимальный размер игрового поля измеренный по оси X или Y.

copy( ): Метод класса, создающий копию объекта игрового поля.

\_\_generate(self): Приватный метод, генерирующий начальную расстановку шашек на игровом поле.

type\_at( ) -> CheckerType: Метод, возвращающий тип шашки по указанным координатам.

at( ) -> Checker: Метод, возвращающий объект шашки по указанным координатам.

is\_within( ) -> bool: Метод, проверяющий, находится ли точка с указанными координатами в пределах игрового поля.

white\_checkers\_count(self) -> int: Метод, возвращающий количество белых шашек на поле.

black\_checkers\_count(self) -> int: Метод, возвращающий количество черных шашек на поле.

white\_score(self) -> int: Метод, возвращающий сумму очков белых шашек на поле.

black\_score(self) -> int: Метод, возвращающий сумму очков черных шашек на поле.

Библиотеки:

1. tkinter - использовалась для создания шахматной доски и меню.
2. checkers - для проверки ходов и перемещение шашек.

В виде БД обычные файлы txt.

**4. Сообщения**

При победе программа выводит окно с сообщениями: “черные выиграли” или “белые выиграли”

При некорректном вводе логина и пароля отображается замечание: “неправильное имя пользователя или пароль”

**Текст программы**

main.py :

from tkinter import Tk, Canvas, Button, Entry, Label, messagebox

import sqlite3

import hashlib

from checkers.game import Game

from checkers.point import Point

from checkers.enums import CheckerType, SideType

# Размер поля

X\_SIZE = 10

Y\_SIZE = 8

CELL\_SIZE = 90

USERS\_FILE = "users.txt"

# Возможные смещения ходов шашек

MOVE\_OFFSETS = [

Point(-1, -1),

Point( 1, -1),

Point(-1, 1),

Point( 1, 1)

]

# Массивы типов белых и чёрных шашек [Обычная пешка, дамка]

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

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

def caesar\_cipher(password):

shift = 1

result = ""

for char in password:

if char.isalpha():

if char.islower():

result += chr((ord(char) - 97 + shift) % 26 + 97)

elif char.isupper():

result += chr((ord(char) - 65 + shift) % 26 + 65)

else:

result += char

return result

def register():

username = username\_entry.get()

password = password\_entry.get()

if username and password:

hashed\_password = caesar\_cipher(password)

with open(USERS\_FILE, "a") as file:

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

messagebox.showinfo("Успех", "Вы успешно зарегистрированы!")

else:

messagebox.showerror("Ошибка", "Введите имя пользователя и пароль")

def authenticate(username, password):

with open(USERS\_FILE, "r") as file:

for line in file:

stored\_username, stored\_password = line.strip().split(":")

if username == stored\_username:

hashed\_password = caesar\_cipher(password)

return hashed\_password == stored\_password

return False

def run\_game():

main\_window.destroy()

# Создание окна

game\_window = Tk()

game\_window.title('Шашки')

game\_window.resizable(0, 0)

# Создание холста

main\_canvas = Canvas(game\_window, width=CELL\_SIZE \* X\_SIZE, height=CELL\_SIZE \* Y\_SIZE)

main\_canvas.pack()

game = Game(main\_canvas, X\_SIZE, Y\_SIZE)

main\_canvas.bind("<Button-1>", game.mouse\_down)

game\_window.mainloop()

def login():

username = username\_entry.get()

password = password\_entry.get()

if authenticate(username, password):

messagebox.showinfo("Успех", "Вы успешно вошли в систему!")

run\_game()

else:

messagebox.showerror("Ошибка", "Неправильное имя пользователя или пароль")

main\_window = Tk()

main\_window.title("Регистрация и вход")

label\_username = Label(main\_window, text="Имя пользователя:")

label\_username.grid(row=0, column=0, padx=5, pady=5)

username\_entry = Entry(main\_window)

username\_entry.grid(row=0, column=1, padx=5, pady=5)

label\_password = Label(main\_window, text="Пароль:")

label\_password.grid(row=1, column=0, padx=5, pady=5)

password\_entry = Entry(main\_window, show="\*")

password\_entry.grid(row=1, column=1, padx=5, pady=5)

button\_login = Button(main\_window, text="Войти", command=login)

button\_login.grid(row=2, column=0, padx=5, pady=5)

button\_register = Button(main\_window, text="Зарегистрироваться", command=register)

button\_register.grid(row=2, column=1, padx=5, pady=5)

main\_window.mainloop()

enums.py :

from enum import Enum, auto

class SideType(Enum):

WHITE = auto()

BLACK = auto()

def opposite(self) -> 'SideType':

if self == SideType.WHITE:

return SideType.BLACK

else:

return SideType.WHITE

def \_\_str\_\_(self) -> str:

return self.name

class CheckerType(Enum):

NONE = auto()

WHITE\_REGULAR = auto()

BLACK\_REGULAR = auto()

WHITE\_QUEEN = auto()

BLACK\_QUEEN = auto()

def \_\_str\_\_(self) -> str:

return self.name

field.py :

from checkers.enums import CheckerType

from checkers.point import Point

from functools import reduce

class Checker:

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

self.\_\_type = type

@property

def type(self):

return self.\_\_type

def change\_type(self, type: CheckerType):

self.\_\_type = type

class Field:

def \_\_init\_\_(self, x\_size: int, y\_size: int):

self.\_\_x\_size = x\_size

self.\_\_y\_size = y\_size

self.\_\_generate()

@property

def x\_size(self) -> int:

return self.\_\_x\_size

@property

def y\_size(self) -> int:

return self.\_\_y\_size

@property

def size(self) -> int:

return max(self.x\_size, self.y\_size)

@classmethod

def copy(cls, field\_instance):

field\_copy = cls(field\_instance.x\_size, field\_instance.y\_size)

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

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

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

return field\_copy

def \_\_generate(self):

self.\_\_checkers = [[Checker() for x in range(self.x\_size)] for y in range(self.y\_size)]

for y in range(self.y\_size):

for x in range(self.x\_size):

if ((y + x) % 2):

if (y < 3):

self.\_\_checkers[y][x].change\_type(CheckerType.BLACK\_REGULAR)

elif (y >= self.y\_size - 3):

self.\_\_checkers[y][x].change\_type(CheckerType.WHITE\_REGULAR)

def type\_at(self, x: int, y: int) -> CheckerType:

return self.\_\_checkers[y][x].type

def at(self, x: int, y: int) -> Checker:

return self.\_\_checkers[y][x]

def is\_within(self, x: int, y: int) -> bool:

return (0 <= x < self.x\_size and 0 <= y < self.y\_size)

@property

def white\_checkers\_count(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type in WHITE\_CHECKERS), checkers, 0) for checkers in self.\_\_checkers)

@property

def black\_checkers\_count(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type in BLACK\_CHECKERS), checkers, 0) for checkers in self.\_\_checkers)

@property

def white\_score(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type == CheckerType.WHITE\_REGULAR) + (checker.type == CheckerType.WHITE\_QUEEN) \* 3, checkers, 0) for checkers in self.\_\_checkers)

@property

def black\_score(self) -> int:

return sum(reduce(lambda acc, checker: acc + (checker.type == CheckerType.BLACK\_REGULAR) + (checker.type == CheckerType.BLACK\_QUEEN) \* 3, checkers, 0) for checkers in self.\_\_checkers)

game.py :

from tkinter import Canvas, Event, messagebox

from PIL import Image, ImageTk

from random import choice

from pathlib import Path

from time import sleep

from math import inf

from typing import List

from checkers.field import Field

from checkers.enums import CheckerType, SideType

from checkers.point import Point

from checkers.enums import CheckerType, SideType

from PIL import Image, ImageTk, ImageDraw

from pathlib import Path

import random

class Move:

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

self.\_\_from\_x = from\_x

self.\_\_from\_y = from\_y

self.\_\_to\_x = to\_x

self.\_\_to\_y = to\_y

@property

def from\_x(self) -> int:

return self.\_\_from\_x

@property

def from\_y(self) -> int:

return self.\_\_from\_y

@property

def to\_x(self) -> int:

return self.\_\_to\_x

@property

def to\_y(self) -> int:

return self.\_\_to\_y

def \_\_repr\_\_(self) -> str:

return f'{self.from\_x}-{self.from\_y} -> {self.to\_x}-{self.to\_y}'

def \_\_eq\_\_(self, other: object) -> bool:

if isinstance(other, self.\_\_class\_\_):

return (

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

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

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

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

)

return NotImplemented

def clone(self) -> 'Move':

return self.\_\_class\_\_(self.from\_x, self.from\_y, self.to\_x, self.to\_y)

# Генерируем случайное значение

PLAYER\_SIDE = random.choice([SideType.WHITE, SideType.BLACK])

# Размер поля

X\_SIZE = 10

Y\_SIZE = 8

CELL\_SIZE = 90

ANIMATION\_SPEED = 5

MAX\_PREDICTION\_DEPTH = 3

BORDER\_WIDTH = 6

FIELD\_COLORS = ['#EEEEEE', '#111111']

HOVER\_BORDER\_COLOR = '#111111'

SELECT\_BORDER\_COLOR = '#00FF00'

POSIBLE\_MOVE\_CIRCLE\_COLOR = '#00FF00'

# Возможные смещения ходов шашек

MOVE\_OFFSETS = [

Point(-1, -1),

Point( 1, -1),

Point(-1, 1),

Point( 1, 1)

]

# Массивы типов белых и чёрных шашек [Обычная пешка, дамка]

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

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

class Game:

def \_\_init\_\_(self, canvas: Canvas, x\_field\_size: int, y\_field\_size: int):

self.\_\_canvas = canvas

self.\_\_field = Field(x\_field\_size, y\_field\_size)

self.\_\_player\_turn = True

self.\_\_hovered\_cell = Point()

self.\_\_selected\_cell = Point()

self.\_\_animated\_cell = Point()

self.\_\_init\_images()

self.\_\_draw()

# Если игрок играет за чёрных, то совершить ход противника

if (PLAYER\_SIDE == SideType.BLACK):

self.\_\_handle\_enemy\_turn()

def \_\_init\_images(self):

self.\_\_images = {

CheckerType.WHITE\_REGULAR: ImageTk.PhotoImage(Image.open(Path('assets', 'white.png')).resize((CELL\_SIZE, CELL\_SIZE))),

CheckerType.BLACK\_REGULAR: ImageTk.PhotoImage(Image.open(Path('assets', 'black.png')).resize((CELL\_SIZE, CELL\_SIZE))),

CheckerType.WHITE\_QUEEN: ImageTk.PhotoImage(Image.open(Path('assets', 'white-q.png')).resize((CELL\_SIZE, CELL\_SIZE))),

CheckerType.BLACK\_QUEEN: ImageTk.PhotoImage(Image.open(Path('assets', 'black-q.png')).resize((CELL\_SIZE, CELL\_SIZE))),

}

def \_\_animate\_move(self, move: Move):

self.\_\_animated\_cell = Point(move.from\_x, move.from\_y)

self.\_\_draw()

# Создание шашки для анимации

animated\_checker = self.\_\_canvas.create\_image(move.from\_x \* CELL\_SIZE, move.from\_y \* CELL\_SIZE, image=self.\_\_images.get(self.\_\_field.type\_at(move.from\_x, move.from\_y)), anchor='nw', tag='animated\_checker')

# Вектора движения

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

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

# Анимация

for distance in range(abs(move.from\_x - move.to\_x)):

for \_ in range(100 // ANIMATION\_SPEED):

self.\_\_canvas.move(animated\_checker, ANIMATION\_SPEED / 100 \* CELL\_SIZE \* dx, ANIMATION\_SPEED / 100 \* CELL\_SIZE \* dy)

self.\_\_canvas.update()

sleep(0.01)

self.\_\_animated\_cell = Point()

def \_\_draw(self):

self.\_\_canvas.delete('all')

self.\_\_draw\_field\_grid()

self.\_\_draw\_checkers()

def \_\_draw\_field\_grid(self):

for y in range(self.\_\_field.y\_size):

for x in range(self.\_\_field.x\_size):

self.\_\_canvas.create\_rectangle(x \* CELL\_SIZE, y \* CELL\_SIZE, x \* CELL\_SIZE + CELL\_SIZE, y \* CELL\_SIZE + CELL\_SIZE, fill=FIELD\_COLORS[(y + x) % 2], width=0, tag='boards')

if (self.\_\_selected\_cell):

player\_moves\_list = self.\_\_get\_moves\_list(PLAYER\_SIDE)

for move in player\_moves\_list:

if (self.\_\_selected\_cell.x == move.from\_x and self.\_\_selected\_cell.y == move.from\_y):

self.\_\_canvas.create\_oval(move.to\_x \* CELL\_SIZE + CELL\_SIZE / 3, move.to\_y \* CELL\_SIZE + CELL\_SIZE / 3, move.to\_x \* CELL\_SIZE + (CELL\_SIZE - CELL\_SIZE / 3), move.to\_y \* CELL\_SIZE + (CELL\_SIZE - CELL\_SIZE / 3), fill=POSIBLE\_MOVE\_CIRCLE\_COLOR, width=0, tag='posible\_move\_circle' )

def \_\_draw\_checkers(self):

for y in range(self.\_\_field.y\_size):

for x in range(self.\_\_field.x\_size):

# Не отрисовывать пустые ячейки и анимируемую шашку

if (self.\_\_field.type\_at(x, y) != CheckerType.NONE and not (x == self.\_\_animated\_cell.x and y == self.\_\_animated\_cell.y)):

self.\_\_canvas.create\_image(x \* CELL\_SIZE, y \* CELL\_SIZE, image=self.\_\_images.get(self.\_\_field.type\_at(x, y)), anchor='nw', tag='checkers')

def mouse\_down(self, event: Event):

if not (self.\_\_player\_turn): return

x, y = (event.x) // CELL\_SIZE, (event.y) // CELL\_SIZE

if not (self.\_\_field.is\_within(x, y)): return

if (PLAYER\_SIDE == SideType.WHITE):

player\_checkers = WHITE\_CHECKERS

elif (PLAYER\_SIDE == SideType.BLACK):

player\_checkers = BLACK\_CHECKERS

else: return

# Если нажатие по шашке игрока, то выбрать её

if (self.\_\_field.type\_at(x, y) in player\_checkers):

self.\_\_selected\_cell = Point(x, y)

self.\_\_draw()

elif (self.\_\_player\_turn):

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

# Если нажатие по ячейке, на которую можно походить

if (move in self.\_\_get\_moves\_list(PLAYER\_SIDE)):

self.\_\_handle\_player\_turn(move)

# Если не ход игрока, то ход противника

if not (self.\_\_player\_turn):

self.\_\_handle\_enemy\_turn()

def \_\_handle\_move(self, move: Move, draw: bool = True) -> bool:

if (draw): self.\_\_animate\_move(move)

# Изменение типа шашки, если она дошла до края

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

self.\_\_field.at(move.from\_x, move.from\_y).change\_type(CheckerType.WHITE\_QUEEN)

elif (move.to\_y == self.\_\_field.y\_size - 1 and self.\_\_field.type\_at(move.from\_x, move.from\_y) == CheckerType.BLACK\_REGULAR):

self.\_\_field.at(move.from\_x, move.from\_y).change\_type(CheckerType.BLACK\_QUEEN)

# Изменение позиции шашки

self.\_\_field.at(move.to\_x, move.to\_y).change\_type(self.\_\_field.type\_at(move.from\_x, move.from\_y))

self.\_\_field.at(move.from\_x, move.from\_y).change\_type(CheckerType.NONE)

# Вектора движения

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

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

# Удаление съеденных шашек

has\_killed\_checker = False

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

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

x += dx

y += dy

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

self.\_\_field.at(x, y).change\_type(CheckerType.NONE)

has\_killed\_checker = True

if (draw): self.\_\_draw()

return has\_killed\_checker

def \_\_handle\_player\_turn(self, move: Move):

self.\_\_player\_turn = False

has\_killed\_checker = self.\_\_handle\_move(move)

required\_moves\_list = list(filter(lambda required\_move: move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y, self.\_\_get\_required\_moves\_list(PLAYER\_SIDE)))

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

self.\_\_player\_turn = True

self.\_\_selected\_cell = Point()

def \_\_handle\_enemy\_turn(self):

self.\_\_player\_turn = False

optimal\_moves\_list = self.\_\_predict\_optimal\_moves(SideType.opposite(PLAYER\_SIDE))

for move in optimal\_moves\_list:

self.\_\_handle\_move(move)

self.\_\_player\_turn = True

self.\_\_check\_for\_game\_over()

def \_\_check\_for\_game\_over(self):

game\_over = False

white\_moves\_list = self.\_\_get\_moves\_list(SideType.WHITE)

if not (white\_moves\_list):

# Белые проиграли

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

game\_over = True

black\_moves\_list = self.\_\_get\_moves\_list(SideType.BLACK)

if not (black\_moves\_list):

# Чёрные проиграли

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

game\_over = True

if (game\_over):

# Новая игра

self.\_\_init\_\_(self.\_\_canvas, self.\_\_field.x\_size, self.\_\_field.y\_size)

def \_\_predict\_optimal\_moves(self, side: SideType) -> List[Move]:

best\_result = float('-inf')

optimal\_moves = []

predicted\_moves\_list = self.\_\_get\_predicted\_moves\_list(side)

if predicted\_moves\_list:

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

for moves in predicted\_moves\_list:

for move in moves:

self.\_\_handle\_move(move, draw=False)

try:

if side == SideType.WHITE:

result = self.\_\_field.white\_score / self.\_\_field.black\_score

elif side == SideType.BLACK:

result = self.\_\_field.black\_score / self.\_\_field.white\_score

except ZeroDivisionError:

result = float('inf')

if result > best\_result:

best\_result = result

optimal\_moves = [moves]

elif result == best\_result:

optimal\_moves.append(moves)

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

optimal\_move = []

if optimal\_moves:

# Фильтрация хода

for move in choice(optimal\_moves):

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

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

break

optimal\_move.append(move)

return optimal\_move

def \_\_get\_predicted\_moves\_list(self, side: SideType, current\_prediction\_depth: int = 0, all\_moves\_list: List[List[Move]] = None, current\_moves\_list: List[Move] = None, required\_moves\_list: List[Move] = None) -> List[List[Move]]:

if all\_moves\_list is None:

all\_moves\_list = []

if current\_moves\_list is None:

current\_moves\_list = []

if required\_moves\_list is None:

required\_moves\_list = []

if current\_moves\_list:

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

else:

all\_moves\_list.clear()

moves\_list = self.\_\_get\_moves\_list(side) if not required\_moves\_list else required\_moves\_list

if moves\_list and current\_prediction\_depth < MAX\_PREDICTION\_DEPTH:

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

for move in moves\_list:

has\_killed\_checker = self.\_\_handle\_move(move, draw=False)

required\_moves\_list = [required\_move for required\_move in self.\_\_get\_required\_moves\_list(side) if move.to\_x == required\_move.from\_x and move.to\_y == required\_move.from\_y]

# Если есть ещё ход этой же шашкой

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

self.\_\_get\_predicted\_moves\_list(side, current\_prediction\_depth, all\_moves\_list, current\_moves\_list + [move], required\_moves\_list)

else:

self.\_\_get\_predicted\_moves\_list(SideType.opposite(side), current\_prediction\_depth + 1, all\_moves\_list, current\_moves\_list + [move])

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

return all\_moves\_list

def \_\_get\_moves\_list(self, side: SideType) -> list[Move]:

moves\_list = self.\_\_get\_required\_moves\_list(side)

if not (moves\_list):

moves\_list = self.\_\_get\_optional\_moves\_list(side)

return moves\_list

def \_\_get\_required\_moves\_list(self, side: SideType) -> list[Move]:

moves\_list = []

# Определение типов шашек

if (side == SideType.WHITE):

friendly\_checkers = WHITE\_CHECKERS

enemy\_checkers = BLACK\_CHECKERS

elif (side == SideType.BLACK):

friendly\_checkers = BLACK\_CHECKERS

enemy\_checkers = WHITE\_CHECKERS

else: return moves\_list

for y in range(self.\_\_field.y\_size):

for x in range(self.\_\_field.x\_size):

# Для обычной шашки

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

for offset in MOVE\_OFFSETS:

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

if self.\_\_field.type\_at(x + offset.x, y + offset.y) in enemy\_checkers and self.\_\_field.type\_at(x + offset.x \* 2, y + offset.y \* 2) == CheckerType.NONE:

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

# Для дамки

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

for offset in MOVE\_OFFSETS:

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

has\_enemy\_checker\_on\_way = False

for shift in range(1, self.\_\_field.size):

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

# Если на пути не было вражеской шашки

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

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

has\_enemy\_checker\_on\_way = True

continue

# Если на пути союзная шашка - то закончить цикл

elif (self.\_\_field.type\_at(x + offset.x \* shift, y + offset.y \* shift) in friendly\_checkers):

break

# Если на пути была вражеская шашка

if (has\_enemy\_checker\_on\_way):

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

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

else:

break

return moves\_list

def \_\_get\_optional\_moves\_list(self, side: SideType) -> list[Move]:

moves\_list = []

# Определение типов шашек

if (side == SideType.WHITE):

friendly\_checkers = WHITE\_CHECKERS

elif (side == SideType.BLACK):

friendly\_checkers = BLACK\_CHECKERS

else: return moves\_list

for y in range(self.\_\_field.y\_size):

for x in range(self.\_\_field.x\_size):

# Для обычной шашки

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

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

if not (self.\_\_field.is\_within(x + offset.x, y + offset.y)): continue

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

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

# Для дамки

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

for offset in MOVE\_OFFSETS:

if not (self.\_\_field.is\_within(x + offset.x, y + offset.y)): continue

for shift in range(1, self.\_\_field.size):

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

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

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

else:

break

return moves\_list

point.py :

class Point:

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

self.\_\_x = x

self.\_\_y = y

@property

def x(self) -> int:

return self.\_\_x

@property

def y(self) -> int:

return self.\_\_y

def \_\_eq\_\_(self, other: 'Point') -> bool:

if isinstance(other, Point):

return (self.x, self.y) == (other.x, other.y)

return False

def \_\_str\_\_(self) -> str:

return f"({self.x}, {self.y})"

def \_\_repr\_\_(self) -> str:

return f"Point(x={self.x}, y={self.y})"

def move(self, dx: int, dy: int) -> None:

self.\_\_x += dx

self.\_\_y += dy

def distance\_to(self, other: 'Point') -> float:

return ((self.x - other.x) \*\* 2 + (self.y - other.y) \*\* 2) \*\* 0.5

def clone(self) -> 'Point':

return Point(self.x, self.y)