**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 «Сиджа»»**

**Пояснительная записка**

Р.02069337. 21/827-13 ПЗ-01

Листов 3

**Руководитель разработки**:

доцент каф. ИВК, к.т.н., доцент

*Шишкин Вадим Викторинович*

« » 2024 г.

**Исполнитель**:

студент гр. ИСТбд-32

*Денисов Данила Алексеевич*

« » 2024 г.

**2024**

Содержание

Аннотация…………………………………………………………………………3

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

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

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

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

**Аннотация**

Аннотация курсовой работы:

Данная курсовая работа посвящена созданию программы для игры "Сиджа". "Сиджа" - это стратегическая настольная игра для двух игроков, цель которой заключается в поедании шашек противника и выведении своих шашек за его пределы. В рамках курсовой работы будет разработано программное обеспечение на языке программирования Python с использованием графической библиотеки Tkinter для создания пользовательского интерфейса.

Основные цели и задачи курсовой работы:

1. Изучение правил игры "Сиджа" и ее стратегий.

2. Проектирование архитектуры программы, включая структуру классов и взаимодействие между ними.

3. Реализация игровой логики, включая расстановку шашек на игровом поле, выполнение ходов игроков и определение условий победы или поражения.

4. Разработка графического интерфейса с помощью библиотеки Tkinter, включающего игровое поле, элементы управления и отображение текущего состояния игры.

5. Тестирование программы на соответствие правилам игры, а также на стабильность и корректность работы.

Ожидаемые результаты курсовой работы включают в себя полностью функционирующую программу для игры "Сиджа" с интуитивно понятным интерфейсом, способную обрабатывать ходы игроков и определять итог игры в соответствии с правилами.

**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 «Сиджа»»**

Р.02069337. 21/827-13 ТЗ-01

Листов 5

**Исполнитель**:

студент гр. ИСТбд-32

*Денисов Данила Алексеевич*

« » 2024 г.

**2024**

**Введение**

Игра «Сиджа».

Поле 5 на 5. Положение фигур в случайном порядке. Игроки поочередно совершают ход. Если игрок “съел” шашку противника - он продолжает ход. За одну фазу игры игрок может закончить игру в свою пользу все время совершая верные ходы. В случае, если нет ходов, при которых игрок мог бы съесть шашку оппонента - ход переходит к противнику. Игра считается завершенной в том случае, если у игрока одной из сторон не осталось шашек на поле.

Функциональные возможности приложения: авторизация, регистрация, игра Сиджа.

**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 «Сиджа». Данное приложение должно иметь графический интерфейс. Необходимо реализовать авторизацию, регистрацию. 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;

В данной работе использовались библиотеки:

1. Tkinter – для создания и отображения меню

2.2.2 Требования к составу функций приложения

Приложение включает в себя следующие классы:

1. Checkers – представляет из себя класс пешки
2. Field – 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. Проверять правильность хода.
5. Выявлять победителя.
6. 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 надёжности**

Необходимо, чтобы входные данные можно было бы зашифровать с помощью Виженера, 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 22H2.
2. Среда разработки: PyCharm Community Edition Version: 2023.3.4
3. Версия языка: Python 3.10.
4. Библиотеки: 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 2025 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 «Сиджа»»**

**Пояснительная записка**

Р.02069337. 21/827-13 ПЗ-01

Листов 8

**Исполнитель**:

студент гр. ИСТбд-32

*Денисов Данила Алексеевич*

« » 2024 г.

**2024**

**Введение**

Реализована игра «Сиджа».

Поле 5 на 5. Положение фигур в случайном порядке. Игроки поочередно совершают ход. Если игрок “съел” шашку противника - он продолжает ход. За одну фазу игры игрок может закончить игру в свою пользу все время совершая верные ходы. В случае, если нет ходов, при которых игрок мог бы съесть шашку оппонента - ход переходит к противнику. Игра считается завершенной в том случае, если у игрока одной из сторон не осталось шашек на поле.

Функциональные возможности приложения: авторизация, регистрация, игра Сиджа.

Реализованы такие структуры данных, как: checker, field, game, move

Checker: реализация ходов отдельной шашки.

Field: реализация игрового поля.

Game: отвечает за GUI и правила игры.

Move: отвечает за ходы шашек по полю

**1. Проектная часть**

**1.1 Постановка задачи на разработку приложения**

Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании (приложение 1).

**1.2 Математические методы**

Математические методы не применялись.

**1.3 Архитектура и алгоритмы**

1.3.1. Архитектура

Структура Checker– представляет из себя шашку, включает в себя:

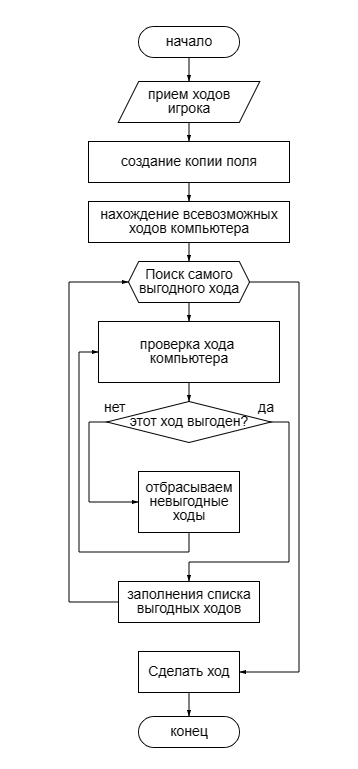
тип шашки.

Field: содержит двухмерную матрицу, которая представляет из себя доску, генерация поля, получение координат шашки, создает копию поля для расчета перемещения ии.

Game: изображения для шашек, отрисовка поля, отрисовка шашек, совершение ходов игроком, совершение ходов ии, алгоритм оптимального хода для ии, алгоритм проверки наихудшего хода для ии, проверка ходов игрока, проверка на конец игры.

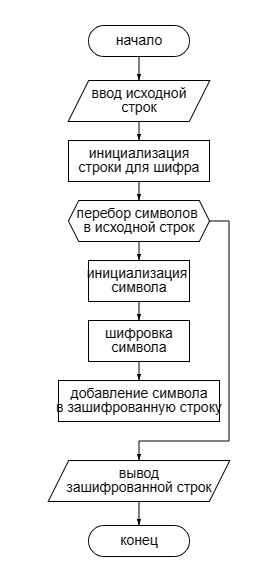
**1.3.2. Алгоритм МИНИМАКС**

Алгоритм реализует поиск оптимального хода.



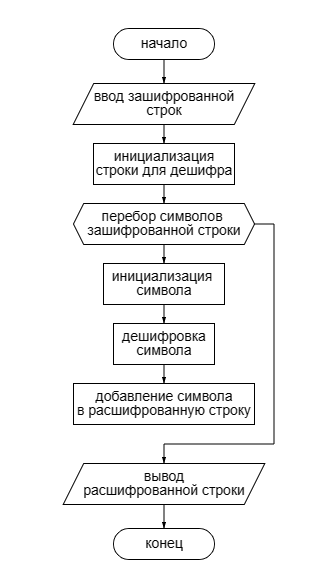
**1.3.2 Алгоритм шифрования Виженера**

Алгоритм реализует шифрование пароля при регистрации.



**1.3.3 Алгоритм дешифрования Виженера**

Алгоритм реализует дешифрование пароля при авторизации.



1.3.1 Алгоритм создания поля

Данный алгоритм осуществляет отрисовку и заполнение поля.

1.3.2 Алгоритм совершения ходов ии

Алгоритм осуществляет поиск всех возможных ходов ии.

1.3.3 Алгоритм поиска оптимального хода для ии

Данный алгоритм осуществляет поиск оптимального хода ии.

1.3.4 Алгоритм поиска худшего хода ии

Алгоритм осуществляет поиск ходов ии, которые он может не осуществлять.

1.3.5 Алгоритм проверки ходов игрока

Данный алгоритм учитывает ходы игрока для наилучшего хода ии.

1.3.6 Алгоритм хода пешки

Алгоритм меняет положение шашки, удаляет съеденные.

1.3.7 Алгоритм проверки на окончание игры

Алгоритм проверяет можно ли закончить игру

**1.4 Тестирование**

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

Для тестирования взяты два распространенных сценария наиболее часто приводящих к ошибке.

1 Сценарий. Неверный ввод пользователем данных.

"Действие пользователя: пустой ввод"

"Ожидаемый результат: пользователь получит сообщение "Ничего не введено""

"Фактический результат: пользователь получает сообщение "Ничего не введено""

2 Сценарий. Попытка авторизации пользователя, ранее не зарегистрированного.

"Действие пользователя: авторизация несуществующего аккаунта"

"Ожидаемый результат: пользователь получит сообщение "Вы ввели неправильный логин или пароль""

"Фактический результат: пользователь получает сообщение "Вы ввели неправильный логин или пароль""

**2. Источники, использованные при разработке**

1. **Лутц, М.** Изучаем Python [Текст]: учеб. - метод, пособие — 6-e изд. - 2019. (дата обращения: 16.03.2024).
2. Tkinter // tkinter URL: https://metanit.com/python/tkinter/ (дата обращения: 15.03.2024).

3. PyQt6 // articles URL: https://habr.com/ru/companies/skillfactory/articles

/599599/ (дата обращения: 18.03.2024).

4. Сиджа // wikipedia URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%B8

%D0%B4%D0%B6%D0%B0 (дата обращения: 14.03.2024).

**3. Приложения**

1) Приложение-1. Техническое задание.

2) Приложение-2. Руководство программиста.

**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 «Сиджа»»**

**Руководство программиста**

Р.02069337. 21/827-13 РП-01

Листов 6

**Исполнитель**:

студент гр. ИСТбд-32

*Денисов Данила Алексеевич*

« » 2024 г.

**2024**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Игра «Сиджа».

Поле 5 на 5. Положение фигур в случайном порядке. Игроки поочередно совершают ход. Если игрок “съел” шашку противника - он продолжает ход. За одну фазу игры игрок может закончить игру в свою пользу все время совершая верные ходы. В случае, если нет ходов, при которых игрок мог бы съесть шашку оппонента - ход переходит к противнику. Игра считается завершенной в том случае, если у игрока одной из сторон не осталось шашек на поле.

Функциональные возможности приложения: авторизация, регистрация, игра Сиджа.

**1.2 Условия, необходимые для использования приложения**

1. ОС: Windows 10 22H2.
2. Среда разработки: PyCharm Community Edition Version: 2023.3.4
3. Версия языка: Python 3.10
4. Библиотеки: tkinter.

В качестве БД использовать файл.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк кода – 840.

Количество алгоритмов – 30.

Количество методов – 35.

Порядок работы:

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.

Кнопка Регистрация позволяет регистрировать пользователя, Авторизация – авторизоваться.

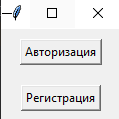


Рис. 1

При успешной регистрации, появляется окно с подтверждением регистрации (Рис. 2).

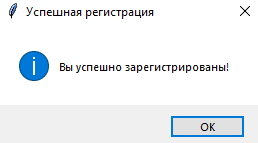


Рис. 2

При успешной авторизации, появляется окно игры (Рис. 3). Также на Рис.3 представлен геймплей. Кнопки игрового поля позволяют совершать ходы пользователю в соответствии с правилами игры.

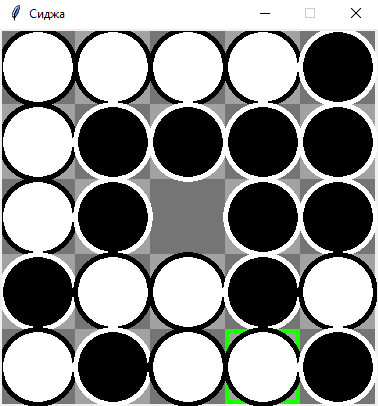


Рис. 3

Случай победы можно увидеть на Рис. 4. Появляются Победа белых или Победа черных.

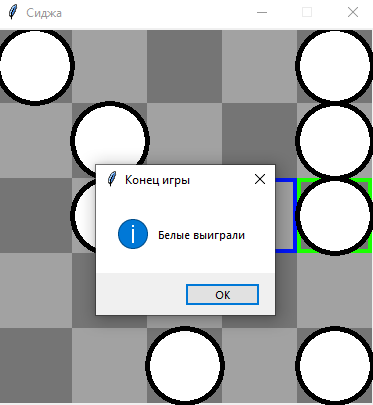


Рис. 4

**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 – для создания доски и меню;

**3. Обращение к программе**

АТД Ckecker требует реализации следующих методов:

1. change\_type – изменяет тип шашки

АТД Field требует реализации следующих методов:

1. copy() – создает копию поля
2. generate() – создает поле в начальном положении с шашками
3. is\_within – определяет, в пределах поля лежит ли точка
4. type\_at – определяет тип шашки по координатам
5. at – определяет шашку по координатам

АТД Game требует реализации следующих методов:

1. \_\_init\_images – инициализирует изображения
2. \_\_draw\_field\_grid – отрисовка поля
3. \_\_draw\_chekers – отрисовка шашек
4. mouse\_move – движение мыши
5. mouse\_down – нажатие левой кнопки мыши
6. \_\_handle\_move – совершение хода
7. \_\_check\_capture – выполнение съедания шашки противника
8. \_\_get\_next\_capture\_move – следующий возможный захват для шашки
9. \_\_handle\_player\_turn – обработка хода игрока и возможности продолжения хода
10. \_\_handle\_enemy\_turn – определение и выполнение хода ии
11. \_\_predict\_optimal\_moves – предсказание оптимального хода для ии
12. \_\_can\_capture\_again – проверка на еще один захват
13. \_\_check\_for\_game\_over – проверка на конец игры
14. \_\_check\_lines – проверка вертикальных и горизонтальных линия для поедания шашек для ии
15. \_\_get\_moves\_list – получение списка ходов
16. \_\_get\_required\_moves\_list – получение списка обязательных ходов
17. \_\_get\_optional\_moves\_list – получение списка необязательных ходов

Для реализации алгоритма МИНИМАКС необходим следующий функционал:

1. \_\_predict\_optimal\_moves – оценочный алгоритм, поиск лучшего хода
2. \_\_get\_moves\_list – возвращает все ходы, которые можно сделать. Учитывает в себе необходимость бить (правило игры).

Библиотеки:

1. Tkinter – использовалась для создания шахматной доски и меню

В виде БД обычные файлы txt.

**4. Сообщения**

При победе программа отображает победителя, в виде сообщений: «Белые победили!» или «Черные победили!»

При некорректном вводе логина и пароля отображается замечание: «Вы ввели неправильный логин или пароль»

**Код программы**

**main.py:**

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

from Sidja.game import Game

from Sidja.constants import X\_SIZE, Y\_SIZE, CELL\_SIZE

from Authorization.cypher import vigenere\_encrypt

from pathlib import Path

USER\_DATA\_PATH = Path('users.txt') # Путь к файлу с данными пользователей

key = "key"

def main():

main\_window = Tk()

main\_window.title('Сиджа')

def open\_auth():

main\_window.destroy()

auth\_window = Tk()

auth\_window.title('Авторизация')

def handle\_auth():

user\_login = input\_login.get()

user\_password = input\_password.get()

user\_data = {"log": user\_login, "pas": user\_password}

user\_encrypted = vigenere\_encrypt(str(user\_data), key)

with USER\_DATA\_PATH.open('r', encoding='utf-8') as file:

existing\_users = file.readlines()

if user\_encrypted + '\n' in existing\_users:

auth\_window.destroy()

run\_tkinter\_app()

else:

messagebox.showerror("Ошибка", "Вы ввели неправильный логин или пароль")

label\_login = Label(auth\_window, text="Введите логин:")

label\_login.pack()

input\_login = Entry(auth\_window)

input\_login.pack()

label\_password = Label(auth\_window, text="Введите пароль:")

label\_password.pack()

input\_password = Entry(auth\_window, show="\*")

input\_password.pack()

login\_btn = Button(auth\_window, text="OK", command=handle\_auth)

login\_btn.pack()

auth\_window.mainloop()

def open\_reg():

main\_window.destroy()

reg\_window = Tk()

reg\_window.title('Регистрация')

def handle\_reg():

user\_login = new\_login.get()

user\_password = new\_pass.get()

user\_data = {"log": user\_login, "pas": user\_password}

user\_encrypted = vigenere\_encrypt(str(user\_data), key)

if len(user\_login) > 16:

messagebox.showerror("Ошибка", "Длина логина превысила 16 символов")

elif len(user\_login) == 0:

messagebox.showerror("Ошибка", "Ничего не введено")

else:

with USER\_DATA\_PATH.open('r', encoding='utf-8') as file:

existing\_users = file.readlines()

if user\_encrypted + '\n' in existing\_users:

messagebox.showerror("Ошибка", "Такой аккаунт уже зарегистрирован")

else:

with USER\_DATA\_PATH.open('a', encoding='utf-8') as file:

file.write(user\_encrypted + '\n')

messagebox.showinfo("Успешная регистрация", "Вы успешно зарегистрированы!")

reg\_window.destroy()

main()

label\_login = Label(reg\_window, text="Введите логин:")

label\_login.pack()

new\_login = Entry(reg\_window)

new\_login.pack()

label\_password = Label(reg\_window, text="Введите пароль:")

label\_password.pack()

new\_pass = Entry(reg\_window, show="\*")

new\_pass.pack()

reg\_btn = Button(reg\_window, text="OK", command=handle\_reg)

reg\_btn.pack()

reg\_window.mainloop()

auth\_btn = Button(main\_window, text="Авторизация", command=open\_auth)

auth\_btn.pack(pady=10)

reg\_btn = Button(main\_window, text="Регистрация", command=open\_reg)

reg\_btn.pack(pady=10)

main\_window.mainloop()

def run\_tkinter\_app():

tk\_window = Tk()

tk\_window.title('Сиджа')

tk\_window.resizable(0, 0)

canvas = Canvas(tk\_window, width=CELL\_SIZE \* X\_SIZE, height=CELL\_SIZE \* Y\_SIZE)

canvas.pack()

game = Game(canvas, X\_SIZE, Y\_SIZE)

canvas.bind("<Motion>", game.mouse\_move)

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

tk\_window.mainloop()

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

main()

**checker.py:**

from Sidja.enums import CheckerType

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):

'''Изменение типа шашки'''

self.\_\_type = type

**constants.py:**

from Sidja.point import Point

from Sidja.enums import CheckerType, SideType

# Сторона за которую играет игрок

PLAYER\_SIDE = SideType.WHITE

# Размер поля

X\_SIZE = Y\_SIZE = 5

# Размер ячейки (в пикселях)

CELL\_SIZE = 75

# Ширина рамки (Желательно должна быть чётной)

BORDER\_WIDTH = 2 \* 2

# Цвета игровой доски

FIELD\_COLORS = ['#757575', '#A2A2A2']

# Цвет рамки при наведении на ячейку мышкой

HOVER\_BORDER\_COLOR = '#1AFF00'

# Цвет рамки при выделении ячейки

SELECT\_BORDER\_COLOR = '#0015FF'

# Цвет кружков возможных ходов

POSIBLE\_MOVE\_CIRCLE\_COLOR = '#FF00E2'

# Возможные смещения ходов шашек

MOVE\_OFFSETS = [

Point(1, 0), # Вправо

Point(-1, 0), # Влево

Point(0, 1), # Вниз

Point(0, -1) # Вверх

]

# Массивы типов белых и чёрных шашек [Обычная пешка]

WHITE\_CHECKERS = [CheckerType.WHITE\_REGULAR]

BLACK\_CHECKERS = [CheckerType.BLACK\_REGULAR]

**enums.py:**

from enum import Enum, auto

class SideType(Enum):

WHITE = auto()

BLACK = auto()

def opposite(side):

if (side == SideType.WHITE):

return SideType.BLACK

elif (side == SideType.BLACK):

return SideType.WHITE

else: raise ValueError()

class CheckerType(Enum):

NONE = auto()

WHITE\_REGULAR = auto()

BLACK\_REGULAR = auto()

BLOCKED = auto()

field.py:

from Sidja.enums import CheckerType

from Sidja.checker import Checker

import random

from Sidja.constants import WHITE\_CHECKERS, BLACK\_CHECKERS

from functools import reduce

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):

'''Создаёт копию поля из образца'''

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):

'''Генерация поля с шашками'''

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

# Заблокировать центральную клетку

central\_x = self.x\_size // 2

central\_y = self.y\_size // 2

self.\_\_checkers[central\_y][central\_x].change\_type(CheckerType.BLOCKED)

# Расставляем 12 черных шашек

black\_coordinates = set() # Множество для хранения выбранных координат черных шашек

black\_count = 0

while black\_count < 12:

x = random.randint(0, self.x\_size - 1)

y = random.randint(0, self.y\_size - 1)

if (x, y) not in black\_coordinates and self.\_\_checkers[y][x].type == CheckerType.NONE:

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

black\_coordinates.add((x, y)) # Добавляем координаты в множество

black\_count += 1

# Расставляем 12 белых шашек

white\_coordinates = set() # Множество для хранения выбранных координат белых шашек

white\_count = 0

while white\_count < 12:

x = random.randint(0, self.x\_size - 1)

y = random.randint(0, self.y\_size - 1)

if (x, y) not in black\_coordinates and (x, y) not in white\_coordinates and self.\_\_checkers[y][

x].type == CheckerType.NONE:

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

white\_coordinates.add((x, y)) # Добавляем координаты в множество

white\_count += 1

# Разблокировать центральную клетку

self.\_\_checkers[central\_y][central\_x].change\_type(CheckerType.NONE)

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

if 0 <= x < self.x\_size and 0 <= y < self.y\_size:

if self.\_\_checkers[y][x] is not None:

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

return None

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

'''Получение шашки на поле по координатам'''

if 0 <= x < self.x\_size and 0 <= y < self.y\_size:

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

else:

# Возвращаем пустую шашку за пределами поля

return Checker() # Или любое другое значение, указывающее на отсутствие шашки

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

'''Определяет лежит ли точка в пределах поля'''

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

**game.py:**

from tkinter import Canvas, Event, messagebox

import random

from math import inf

from typing import Tuple

from Sidja.field import Field

from Sidja.move import Move

from Sidja.constants import \*

from Sidja.enums import CheckerType, SideType

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(-1, -1)

self.\_\_white\_score = 0 # Инициализация счета для белых

self.\_\_black\_score = 0 # Инициализация счета для черных

self.checkers\_type\_for\_game()

self.\_\_draw()

# Если игрок играет за чёрных, то совершить ход противника

if PLAYER\_SIDE == SideType.BLACK:

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

def checkers\_type\_for\_game(self):

'''Инициализация изображений'''

self.\_\_images = {

CheckerType.WHITE\_REGULAR: 'white',

CheckerType.BLACK\_REGULAR: 'black',

}

def \_\_draw(self):

'''Отрисовка сетки поля и шашек'''

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

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

self.\_\_draw\_checkers()

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

'''Отрисовка сетки поля'''

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 is not None and x == self.\_\_selected\_cell.x and y == self.\_\_selected\_cell.y:

self.\_\_canvas.create\_rectangle(x \* CELL\_SIZE + BORDER\_WIDTH // 2, y \* CELL\_SIZE + BORDER\_WIDTH // 2, x \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2, y \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2, outline=SELECT\_BORDER\_COLOR, width=BORDER\_WIDTH, tag='border')

elif (x == self.\_\_hovered\_cell.x and y == self.\_\_hovered\_cell.y):

self.\_\_canvas.create\_rectangle(x \* CELL\_SIZE + BORDER\_WIDTH // 2, y \* CELL\_SIZE + BORDER\_WIDTH // 2, x \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2, y \* CELL\_SIZE + CELL\_SIZE - BORDER\_WIDTH // 2, outline=HOVER\_BORDER\_COLOR, width=BORDER\_WIDTH, tag='border')

# Отрисовка возможных точек перемещения, если есть выбранная ячейка

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):

'''Отрисовка шашек'''

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

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

checker\_type = self.\_\_field.type\_at(x, y)

if checker\_type != CheckerType.NONE:

color = 'white' if checker\_type == CheckerType.WHITE\_REGULAR else 'black'

border\_color = 'black' if color == 'white' else 'white'

border\_width = 5 # Устанавливаем ширину границы

self.\_\_canvas.create\_oval(

x \* CELL\_SIZE, y \* CELL\_SIZE,

(x + 1) \* CELL\_SIZE, (y + 1) \* CELL\_SIZE,

fill=color, outline=border\_color, width=border\_width

)

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

'''Событие перемещения мышки'''

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

if (x != self.\_\_hovered\_cell.x or y != self.\_\_hovered\_cell.y):

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

# Если ход игрока, то перерисовать

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

self.\_\_draw()

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

'''Событие нажатия мышки'''

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

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

# Если точка не внутри поля

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):

try:

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)):

if self.\_\_field.type\_at(x, y) == CheckerType.NONE and move in self.\_\_get\_moves\_list(PLAYER\_SIDE):

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

else:

# Если не ход игрока, то ход противника

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

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

except:

pass

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

"""Совершение хода и проверка на захват шашек без перемещения"""

# Сохраняем тип шашки до перемещения

moving\_checker\_type = self.\_\_field.type\_at(move.from\_x, move.from\_y)

# Перемещаем шашку

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

self.\_\_field.at(move.to\_x, move.to\_y).change\_type(moving\_checker\_type)

# Проверяем возможность захвата без фактического перемещения шашки игрока

captured = self.\_\_check\_capture(move.to\_x, move.to\_y, moving\_checker\_type)

if draw:

self.\_\_draw()

return captured

def \_\_check\_capture(self, x, y, player\_type) -> bool:

"""Проверка на захват и выполнение захвата по вертикали и горизонтали."""

captured = False

directions = [(0, 1), (1, 0), (0, -1), (-1, 0)] # Ограничиваем направления проверки

for dx, dy in directions:

next\_x, next\_y = x + dx, y + dy

if self.\_\_field.is\_within(next\_x, next\_y):

if self.\_\_field.type\_at(next\_x, next\_y) != player\_type and self.\_\_field.type\_at(next\_x,

next\_y) != CheckerType.NONE:

# Если шашка противника находится рядом, проверяем, можно ли её "съесть"

opposite\_x, opposite\_y = next\_x + dx, next\_y + dy

if self.\_\_field.is\_within(opposite\_x, opposite\_y) and self.\_\_field.type\_at(opposite\_x,

opposite\_y) == player\_type:

# Захватываем шашку противника

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

captured = True

return captured

def \_\_get\_next\_capture\_move(self, x: int, y: int) -> Move:

'''Находит следующий возможный захват для шашки'''

player\_type = self.\_\_field.type\_at(x, y)

enemy\_type = CheckerType.WHITE\_REGULAR if player\_type == CheckerType.BLACK\_REGULAR else CheckerType.BLACK\_REGULAR

for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

next\_x = x + dx \* 2

next\_y = y + dy \* 2

between\_x = x + dx

between\_y = y + dy

if (self.\_\_field.is\_within(next\_x, next\_y) and

self.\_\_field.type\_at(between\_x, between\_y) == enemy\_type and

self.\_\_field.type\_at(next\_x, next\_y) == CheckerType.NONE):

return Move(x, y, next\_x, next\_y)

return None

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

'''Обработка хода игрока и возможность продолжения хода'''

self.\_\_player\_turn = False

# Проверка наличия захвата после выполнения хода игрока

captured = self.\_\_handle\_move(move)

# Если после хода игрока не было захвата, передать ход компьютеру

if not captured:

self.\_\_handle\_enemy\_turn() # Передача хода компьютеру

# Проверка на возможность продолжения хода в случае захвата

while captured and self.\_\_can\_capture\_again(move.to\_x, move.to\_y):

self.\_\_draw() # Обновление доски

move = self.\_\_get\_next\_capture\_move(move.to\_x, move.to\_y)

captured = self.\_\_handle\_move(move)

self.\_\_player\_turn = True

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

self.\_\_selected\_cell = None

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

"""Определение и выполнение хода компьютера."""

enemy\_side = SideType.opposite(PLAYER\_SIDE)

optimal\_moves = self.\_\_predict\_optimal\_moves(enemy\_side)

if optimal\_moves:

chosen\_move = random.choice(optimal\_moves) # Выбираем случайный из оптимальных ходов

captured = self.\_\_handle\_move(chosen\_move)

# Проверка на возможность повторного захвата

while captured and self.\_\_can\_capture\_again(chosen\_move.to\_x, chosen\_move.to\_y):

next\_move = self.\_\_get\_next\_capture\_move(chosen\_move.to\_x, chosen\_move.to\_y)

if next\_move:

chosen\_move = next\_move

captured = self.\_\_handle\_move(chosen\_move)

self.\_\_player\_turn = True

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

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

'''Предсказать оптимальный ход для заданной стороны.'''

optimal\_moves = []

best\_score\_diff = -inf

# Получаем список всех возможных ходов для заданной стороны

possible\_moves = self.\_\_get\_moves\_list(side)

for move in possible\_moves:

# Создаем копию доски для тестирования хода

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

# Выполняем ход на копии доски

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

# Оцениваем результат хода

white\_score, black\_score = self.\_\_check\_lines()

score\_diff = white\_score - black\_score if side == SideType.WHITE else black\_score - white\_score

# Восстанавливаем исходное состояние доски

self.\_\_field = board\_copy

# Обновляем список оптимальных ходов

if score\_diff > best\_score\_diff:

best\_score\_diff = score\_diff

optimal\_moves = [move]

elif score\_diff == best\_score\_diff:

optimal\_moves.append(move)

return optimal\_moves

def \_\_can\_capture\_again(self, x, y) -> bool:

'''Проверяет, может ли шашка на заданных координатах выполнить еще один захват.'''

player\_type = self.\_\_field.type\_at(x, y)

enemy\_type = CheckerType.WHITE\_REGULAR if player\_type == CheckerType.BLACK\_REGULAR else CheckerType.BLACK\_REGULAR

for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:

capture\_x = x + dx \* 2

capture\_y = y + dy \* 2

between\_x = x + dx

between\_y = y + dy

# Проверяем, что по направлению хода есть шашка противника и пустое поле для захвата

if (self.\_\_field.is\_within(capture\_x, capture\_y) and

self.\_\_field.type\_at(between\_x, between\_y) == enemy\_type and

self.\_\_field.type\_at(capture\_x, capture\_y) == CheckerType.NONE):

return True

return False

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

'''Проверка на конец игры'''

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 \_\_check\_lines(self) -> Tuple[int, int]:

'''Проверка наличия линий из шашек и начисление очков'''

white\_score = 0

black\_score = 0

# Проверка горизонтальных линий

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

for x in range(self.\_\_field.x\_size - 1):

if self.\_\_field.type\_at(x, y) == CheckerType.WHITE\_REGULAR and self.\_\_field.type\_at(x+1, y) == CheckerType.WHITE\_REGULAR:

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

self.\_\_field.at(x+1, y).change\_type(CheckerType.NONE)

white\_score += 1

elif self.\_\_field.type\_at(x, y) == CheckerType.BLACK\_REGULAR and self.\_\_field.type\_at(x+1, y) == CheckerType.BLACK\_REGULAR:

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

self.\_\_field.at(x+1, y).change\_type(CheckerType.NONE)

black\_score += 1

# Проверка вертикальных линий

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

for y in range(self.\_\_field.y\_size - 1):

if self.\_\_field.type\_at(x, y) == CheckerType.WHITE\_REGULAR and self.\_\_field.type\_at(x, y+1) == CheckerType.WHITE\_REGULAR:

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

self.\_\_field.at(x, y+1).change\_type(CheckerType.NONE)

white\_score += 1

elif self.\_\_field.type\_at(x, y) == CheckerType.BLACK\_REGULAR and self.\_\_field.type\_at(x, y+1) == CheckerType.BLACK\_REGULAR:

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

self.\_\_field.at(x, y+1).change\_type(CheckerType.NONE)

black\_score += 1

return white\_score, black\_score

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

'''Получение списка ходов'''

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]:

'''Получение списка обязательных ходов'''

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:

new\_x, new\_y = x , y

# Проверяем только вертикальные и горизонтальные ходы

if self.\_\_field.is\_within(new\_x, new\_y):

# Проверяем, что клетка, куда мы хотим сделать ход, пуста

if self.\_\_field.type\_at(new\_x, new\_y) == CheckerType.NONE:

moves\_list.append(Move(x, y, new\_x, new\_y))

return moves\_list

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

'''Получение списка необязательных ходов'''

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:

new\_x, new\_y = x + offset.x, y + offset.y

# Проверяем только вертикальные и горизонтальные ходы

if self.\_\_field.is\_within(new\_x, new\_y):

# Проверяем, что клетка, куда мы хотим сделать ход, пуста

if self.\_\_field.type\_at(new\_x, new\_y) == CheckerType.NONE:

moves\_list.append(Move(x, y, new\_x, new\_y))

return moves\_list

**move.py:**

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):

return self.\_\_from\_x

@property

def from\_y(self):

return self.\_\_from\_y

@property

def to\_x(self):

return self.\_\_to\_x

@property

def to\_y(self):

return self.\_\_to\_y

def \_\_str\_\_(self):

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

def \_\_repr\_\_(self):

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

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

if isinstance(other, Move):

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

**point.py:**

class Point:

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

self.\_\_x = x

self.\_\_y = y

@property

def x(self):

return self.\_\_x

@property

def y(self):

return self.\_\_y

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

if isinstance(other, Point):

return (

self.x == other.x and

self.y == other.y

)

return NotImplemented

**cypher.py:**

key = "key"

def vigenere(text: str, key: str, encrypt=True):

result = ""

for i in range(len(text)):

letter\_n = ord(text[i])

key\_n = ord(key[i % len(key)])

if encrypt:

value = (letter\_n + key\_n) % 1114112

else:

value = (letter\_n - key\_n) % 1114112

result += chr(value)

return result

def vigenere\_encrypt(text: str, key: str):

return vigenere(text=text, key=key, encrypt=True)

def vigenere\_decrypt(text: str, key: str):

return vigenere(text=text, key=key, encrypt=False)