МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования «Санкт-петербургский государственный морской технический университет»

Факультет: Цифровых промышленных технологий

Кафедра: Киберфизических систем

Дисциплина: Программирование

Группа: 20121

Тип работы: Курсовой проект

Выполнил:  
Наталенко Илья Александрович, группа 20121  
Проверил:  
Поделенюк Павел Петрович, преподаватель СПБГМТУ

**Оглавление**

[1. Цели и формулировка задачи 3](#_Toc138376276)

[2. Разработка классов и UML диаграмма 7](#_Toc138376277)

[3. Реализация программы с использованием функционального программирования языка Python 8](#_Toc138376278)

[3.1. Ход работы 8](#_Toc138376279)

[3.2. Демонстрация работы программы 13](#_Toc138376280)

[3.3. Листинг кода 13](#_Toc138376281)

[4. Результат работы: 13](#_Toc138376282)

[5. Заключение 14](#_Toc138376283)

[6. Список используемых источников 15](#_Toc138376284)

1. Цели и формулировка задачи

**Цель**

Анализ ходов фигуры на шахматной доске

При работе над данным лабораторным проектом освоить следующие методы и инструменты языка:

1. Работа с основами функционального программирования языка Python

2. Работы с основами ООП языка Python

3. Разработка классов и UML диаграмма

4. Работы с пакетами Python

5. Создание GUI приложения с использованием tkinter (или customtkinter)

+ pygame (+OpenGl).

**Формулировка задания**

Дана квадратная шахматная доска размером N x N. На доске уже размещено K фигур. Фигуры размещены так, что находятся не под боем друг друга. Необходимо расставить на доске еще L фигур так, чтобы никакая из фигур на доске не находилась под боем любой другой фигуры. Требуется найти одно решение для визуализации и все возможные решения для вывода в файл. Если решение не найдено, то необходимо вывести соответствующее сообщение.

1) Необходимо создать UML диаграмму взаимодействия классов. При необходимости можно делать несколько UML диаграмм. Для диаграммы создается табличка взаимодействия (Имя класса :: Имя базового класса :: Описание). Также для каждого класса создается таблица методов и атрибутов (Методы и атрибуты :: Описание ). При описании методов обязательно прописывать тип параметров, а также выходной тип данных. При описании атрибутов необходимо прописывать его тип.

2) Необходимо составить программу с использованием функционального программирования языка Python, где:

1. Входные данные в файле input.txt. На первой строке файла записаны три числа: N L K (через пробел). Далее следует K строк, содержащих числа x и y (через пробел) - координаты уже стоящей на доске фигуры (фигуры стоят правильно). Координаты отсчитываются от 0 до N-1. 1 <= N <= 20.
2. Выходные данные в файл output.txt. На каждое найденное решение необходимо записать в файл одну строку. Строка состоит из пар (x,y) - координаты фигур на доске. В решение следует вывести координаты всех фигур, находящихся на доске. Каждую фигуру необходимо записать в виде пары координат, разделенных запятой и обрамленных скобками. Координаты отсчитываются от 0 до N-
3. Порядок, в котором фигуры перечислены в решении, не имеет значения. Если не было найдено ни одного решения, в файл необходимо записать no solutions.
4. Выходные данные на консоль — это доска N\*N, где фигура обозначается #, ее ходы обозначаются \*, а пустые клетки обозначаются 0.

3) Необходимо составить программу с использованием ООП языка Python, а также разработать интерфейса при помощи пакетов tkinter (или customtkinter) + pygame (+OpenGl).

Реализовать любой из предложенных вариантов:

1. tkinter (или customtkinter) + pygame(+OpenGl)).

1.1. tkinter (или customtkinter)

1.1.1. На интерфейсе есть два поля ввода и кнопка. В первом поле вводится размер доски N, во втором – количество фигур, которые необходимо расставить с помощью алгоритма L, и с помощью кнопки создается новое окно.

Необходимо проверять правильность ввода данных.

1.2. pygame (+OpenGl)

1.2.1. В данном интерфейсе есть шахматная доска и кнопка. На созданной шахматной доске с помощью ПКМ/ЛКМ необходимо расставить/убрать стоящие на доске K фигур. Необходимо визуализировать саму фигуру и ее ходы (визуализацию фигуры и ее ходов выбираете сами, но так, чтобы они отличались). Также необходимо проверить, чтобы поставленная вами фигура не находилась под боем.

С помощью кнопки создается новое окно.

1.2.2. В данном интерфейсе есть шахматная доска и кнопка. На созданной шахматной доске показываются фигуры, расставленные пользователем и найденные алгоритмом, (нужно показать любое найденное решение). Визуализация пользовательских и найденных алгоритмом фигур должна отличаться (визуализация ходов одинаковая).

Если решение не было найдено, то вывести сообщение об этом и закрыть данный интерфейс.

С помощью кнопки происходит вывод данных в файл output.txt. На каждое найденное решение необходимо записать в файл одну строку. Строка состоит из пар (x,y) - координаты фигур на доске. В решение следует вывести координаты всех фигур, находящихся на доске. Каждую фигуру необходимо записать в виде пары координат, разделенных запятой и обрамленных скобками. Координаты отсчитываются от 0 до N. Порядок, в котором фигуры перечислены в решении, не имеет значения.

2. tkinter (или customtkinter) + pygame(+OpenGl)).

2.1. tkinter (или customtkinter)

2.1.1. На интерфейсе есть три поля ввода и кнопка. В первом поле вводится размер доски N, во втором – количество фигур, которые необходимо расставить с помощью алгоритма L, в третьем – количество стоящих на доске фигур K. С помощью кнопки создается новое окно.

Необходимо проверять правильность ввода данных.

2.1.2. В данном интерфейсе есть K полей ввода и кнопка. В каждое поле необходимо ввести числа x и y (через пробел) - координаты уже стоящей на доске фигуры. С помощью кнопки создается новое окно.

Необходимо проверять правильность ввода данных.

2.2. pygame (+OpenGl)

2.2.1. В данном интерфейсе есть шахматная доска и кнопка. На созданной шахматной доске показываются фигуры, расставленные пользователем и найденные алгоритмом, (нужно показать любое найденное решение). Визуализация пользовательских и найденных алгоритмом фигур должна отличаться (визуализация ходов одинаковая).

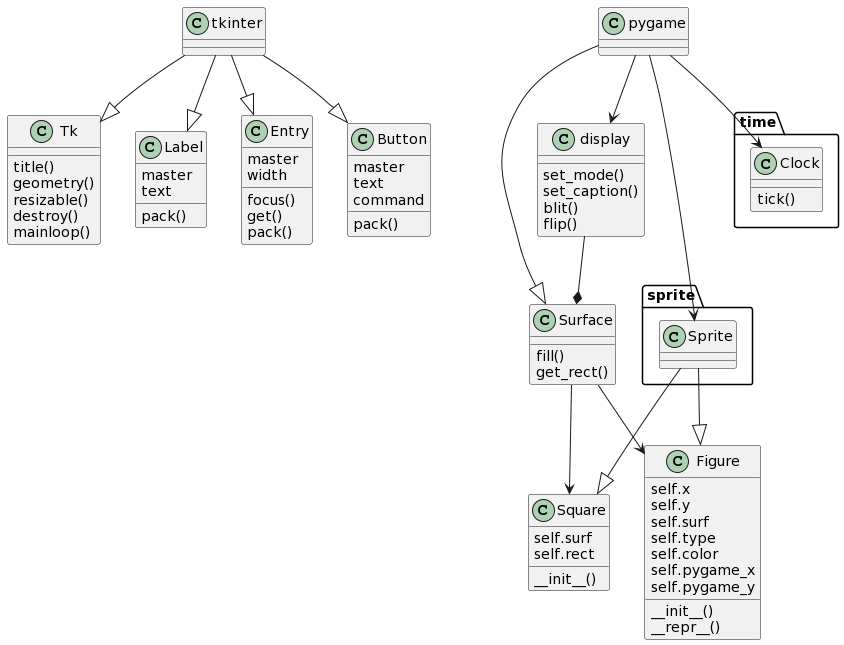
Если решение не было найдено, то вывести сообщение об этом и закрыть данный интерфейс.

С помощью кнопки происходит вывод данных в файл output.txt. На каждое найденное решение необходимо записать в файл одну строку. Строка состоит из пар (x,y) - координаты фигур на доске.

В решение следует вывести координаты всех фигур, находящихся на доске. Каждую фигуру необходимо записать в виде пары координат, разделенных запятой и обрамленных скобками. Координаты отсчитываются от 0 до N. Порядок, в котором фигуры перечислены в решении, не имеет значения.

1. **Разработка классов и UML диаграмма**

**1)**



1. **Реализация программы с использованием функционального программирования языка Python**
   1. **Ход работы**

1) В целях упрощения создания визуализации шахматной доски были написаны два класса.

Класс Square отвечает за создание спрайта в виде квадрата размером c\_size чёрного или белого цвета, в зависимости от того, какое значение примет color (может быть только ‘black’ или ‘white’).

class Square:  
 def \_\_init\_\_(self, c\_size: float, color: str):  
 super(Square, self).\_\_init\_\_()  
 self.surf = pg.Surface((c\_size, c\_size))  
 if color == 'white':  
 self.surf.fill(colors['white'])  
 else:  
 self.surf.fill(colors['black'])  
 self.rect = self.surf.get\_rect()

2) Класс Figure отвечает за создание спрайта в виде квадрата красного, жёлтого или синего цвета с начальными координатами. Конечные координаты pygame\_x и pygame\_y вычисляются в инициализаторе.  
Цвета соответствуют следующим типам заполнения:

* Красный – фигура, поставленная пользователем
* Жёлтый – фигура, поставленная алгоритмом
* Синий – клетки под боем красной или жёлтой фигуры

Помимо инициализатора в классе Figure был прописан волшебный метод \_\_repr\_\_, распознающий, является спрайт фигурой или клеткой под боем и отражающий это в окне отладки.

class Figure:def \_\_init\_\_(self, x: int, y: int, type: str):  
 self.x = x  
 self.y = y  
 self.type = type  
 self.surf = pg.Surface((cell\_size, cell\_size))  
 if type == 'figure':  
 self.surf.fill(colors['red'])  
 self.color = colors['red']  
 elif type == 'new figure':  
 self.surf.fill(colors['yellow'])  
 self.color = colors['yellow']  
 else:  
 self.surf.fill(colors['blue'])  
 self.color = colors['blue']  
 self.rect = self.surf.get\_rect()  
 self.pygame\_x = border\_width \* (x + 1) + cell\_size \* self.x  
 self.pygame\_y = border\_width \* (y + 1) + cell\_size \* self.y  
  
 def \_\_repr\_\_(self):  
 if self.type == 'figure' or self.type == 'new figure':  
 return 'Figure'  
 else:  
 return 'Beaten Cell'

3) Функция error() используется для создания окна с ошибкой на случай, если ввод каких-то данных был неправильным. Может создаваться при вводе значений N, L, K или при вводе координат K поставленных фигур (красных).

def error():  
 error\_window = tk.Tk()  
 error\_window.title('Ошибка!')  
 error\_window.geometry('230x47')  
 tk.Label(master=error\_window, text='Неверный формат ввода!').pack()  
 tk.Button(master=error\_window, text='Ок', command= lambda: error\_window.destroy()).pack()  
 error\_window.mainloop()

4) Функция ввода чисел N, L, K, проверяющая, являются ли введённые данные числами и входят ли они в нужные диапазоны.

* Для размера доски N диапазон 1≤N≤20
* Для необходимых к расстановке фигур L диапазон L≥0
* Для расставленных фигур K диапазон K≥0

def enter():  
 global correct, N, L, K  
 N, L, K = n\_entry.get().strip(), l\_entry.get().strip(), k\_entry.get().strip()  
 window.destroy()  
 if not N.isdigit() or not L.isdigit() or not K.isdigit():  
 error()  
 else:  
 N, L, K = int(N), int(L), int(K)  
 if not 1 <= N <= 20 or not L >= 0 or not K >= 0:  
 error()

5) Функция enter\_coords принимает K пар координат x и y, делает такие же проверки, что и функция enter() и помещает в список координат, с которым далее будет осуществляться работа.

def enter\_coords(entry\_fields: list):  
 for e in enter\_array:  
 string = e.get()  
 coords\_pair = tuple(string.split())  
 if len(coords\_pair) == 2 and False not in tuple(map(lambda x: x.isdigit(), coords\_pair)):  
 if 0 <= int(coords\_pair[0]) < N and 0 <= int(coords\_pair[1]) < N:  
 coords.append((int(coords\_pair[0]), int(coords\_pair[1])))  
 else:  
 error()  
 else:  
 error()  
 k\_window.destroy()

6) Функция output() меняет значение show\_board на True и закрывает окно вывода. Если функция была вызвана, то позже шахматная доска появится.

def output():  
 global show\_board  
 show\_board = True  
 output\_window.destroy()

7) Функция beaten\_cells() ищет все координаты клеток под боем для фигуры Конь-Дракон (Король + Слон).

* Сначала находятся клетки вокруг фигуры путём прибавления к координатам фигуры значений от -1 до 1.
* Далее, находятся клетки по диагонали. Чтобы не трогать все остальные, не только расширяется диапазон, но и увеличивается шаг. Таким образом, по окончании каждой внешней итерации будет записано только четыре пары координат.

def beaten\_cells(size: int, x: int, y: int):  
 beaten = []  
 *# Клетки вокруг фигуры.* for row in range(-1, 2):  
 for col in range(-1, 2):  
 if 0 <= x + row < size and 0 <= y + col < size and (x + row, y + col) not in beaten:  
 beaten.append((x + row, y + col))  
 step = 2  
 *# Клетки по диагонали.* while step < size:  
 for row in range(x - step, x + step + 1, step \* 2):  
 for col in range(y - step, y + step + 1, step \* 2):  
 if 0 <= row < size and 0 <= col < size and (row, col) not in beaten:  
 beaten.append((row, col))  
 step += 1  
 return beaten

8) Создаётся интерфейс для ввода чисел N, L, K.

window = tk.Tk()  
window.title('Курсовой проект "Шахматы"')  
window.geometry('335x150')  
window.resizable(False, False)  
n, n\_entry = tk.Label(master=window, text='Размер доски (N):'), tk.Entry(width=10)  
n.pack()  
n\_entry.pack()  
n\_entry.focus()  
l, l\_entry = tk.Label(master=window, text='Фигур надо поставить (L):'), tk.Entry(width=10)  
l.pack()  
l\_entry.pack()  
k, k\_entry = tk.Label(master=window, text='Фигур уже расставлено (K):'), tk.Entry(width=10)  
k.pack()  
k\_entry.pack()  
correct, N, L, K = False, 0, 0, 0  
enter\_button = tk.Button(text="Ввод", command=enter)  
enter\_button.pack()  
window.mainloop()

9) Создаётся интерфейс для ввода K пар координат.

coords = []  
k\_window = tk.Tk()  
k\_window.geometry(f'250x{20 \* (K + 1)}')  
k\_window.title('Координаты')  
enter\_array = [tk.Entry(master=k\_window) for o in range(K)]  
for i in range(K):  
 enter\_array[i].pack()  
enter\_button = tk.Button(text="Ввод", command=lambda: enter\_coords(enter\_array))  
enter\_button.pack()  
if K > 0:  
 k\_window.mainloop()

10) Получение координат под боем и не под боем. Выполняется только для уже расставленных фигур. Дальше данные два списка не изменяются.

for fig in coords:  
 if fig not in beaten\_coords:  
 beaten\_coords.extend(beaten\_cells(N, fig[0], fig[1]))  
 beaten\_coords = list(set(beaten\_coords))  
 else:  
 error()  
beaten\_coords.sort()  
not\_beaten\_coords = list(filter(lambda x: x not in beaten\_coords, all\_coords))

11) Вычислительная часть с использованием выше упомянутых списков. В каждом варианте создаётся временный список координат под боем, в который добавляются координаты новых фигур и координаты их возможных ходов. Если координаты новой фигуры уже есть во временном списке координат под боем, решение недействительно, и берётся следующее.

В каждом решении гарантированно ставится только L фигур, так как решения составляются на основе множества, состоящего из перестановок строки из L единиц и нулей. Количество нулей в строке равно разности количества свободных клеток и L.

Имея такие строки и список not\_beaten\_cells такой же длины, мы можем перебрать все варианты расстановки с учётом уже расставленных фигур, а уже из них убрать те, в которых фигуры находятся под боем друг друга.

solution\_list = []  
if L > 0:  
 for variant in set(map(lambda x: ''.join(list(x)), set(permutations(('1' \* L).zfill(len(not\_beaten\_coords)))))):  
 valid = True  
 start\_index = 0  
 temp\_beaten\_coords = beaten\_coords[:]  
 new\_coords = []  
 for i in range(L):  
 index = variant.find('1', start\_index, len(variant))  
 if not\_beaten\_coords[index] not in temp\_beaten\_coords:  
 temp\_beaten\_coords += beaten\_cells(N, not\_beaten\_coords[index][0], not\_beaten\_coords[index][1])  
 new\_coords.append(not\_beaten\_coords[index])  
 else:  
 valid = False  
 break  
 start\_index = index + 1  
 if valid:  
 solution\_list.append(coords + new\_coords)

12) Действительные решения, записанные в solution\_list, выводятся в файл output.txt. Квадратные скобки от списка убираются срезом строки: str(valid\_variant)[1:-1] Каждое решение представляет набор кортежей, по два числа в каждом. Сначала идут координаты уже поставленных фигур, а затем фигур, поставленных алгоритмом.

out\_file = open('output.txt', 'w')  
for valid\_variant in solution\_list:  
 out\_file.write(str(valid\_variant)[1:-1] + '\n')

13) Создаётся окно вывода, где пользователь решает, выводить шахматную доску со случайным найденным решением (кнопка output\_button, меняющая состояние show\_board с False на True) или завершить работу программы, ограничившись выводом решений в файл.

output\_window = tk.Tk()  
output\_window.geometry('275x20')  
output\_window.title("Вывод данных")  
show\_board = False  
output\_button = tk.Button(master=output\_window, text="Вывести", command=output)  
output\_button.pack()  
output\_window.mainloop()

14) Если show\_board приняла значение True, происходит инициализация данных для вывода шахматной доски. Создаётся словарь для цветов, игровой дисплей серого цвета, задаётся размер границ между клетками, рассчитывается размер самих клеток, создаются часы pygame.time.Clock() и выбирается случайный вариант из списка решений.

В цикле while устанавливается значение тиков для часов и создаётся чистая шахматная доска из чёрных и белых клеток. Если сумма координат x и y при делении на два даёт в остатке ноль, то создаётся белая клетка, а иначе чёрная. Таким образом, шахматный порядок соблюдается. Для примера:

* Для координаты (0; 0): 0 + 0 % 2 = 0 (белая)
* Для координаты (0; 1): 0 + 1 % 2 = 1 (чёрная)
* Для координаты (1; 0): 1 + 0 % 2 = 1 (чёрная)
* Для координаты (1; 1): 1 + 1 % 2 = 0 (белая)

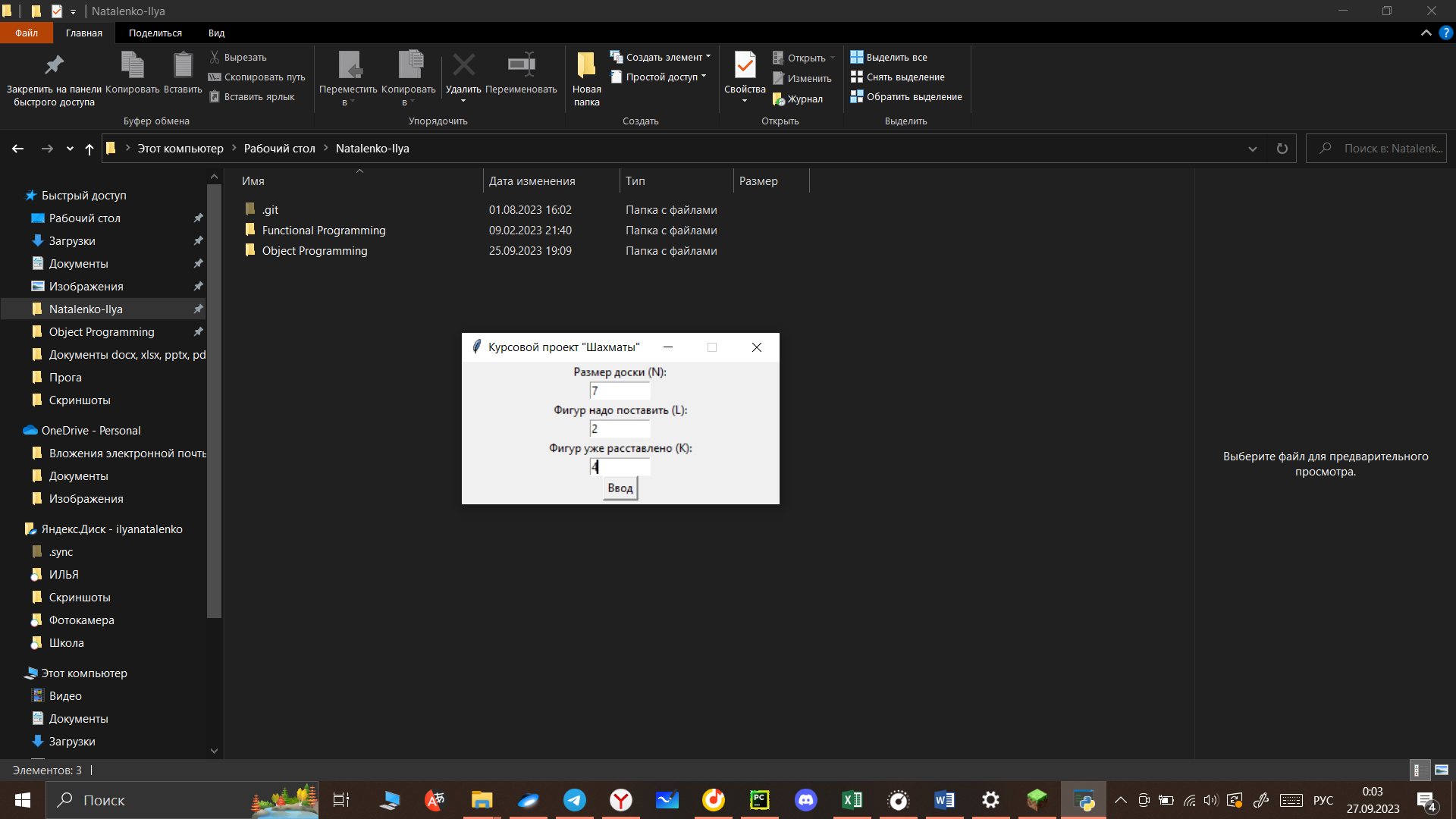
Далее рассматриваются координаты из случайного решения. По этим координатам сначала рисуются ходы фигуры (синие клетки), а потом и сама фигура (красные и жёлтые клетки). Такой порядок обусловлен тем, что в координатах ходов фигуры есть и координаты самой фигуры, и её ячейка тоже закрашивается синим.

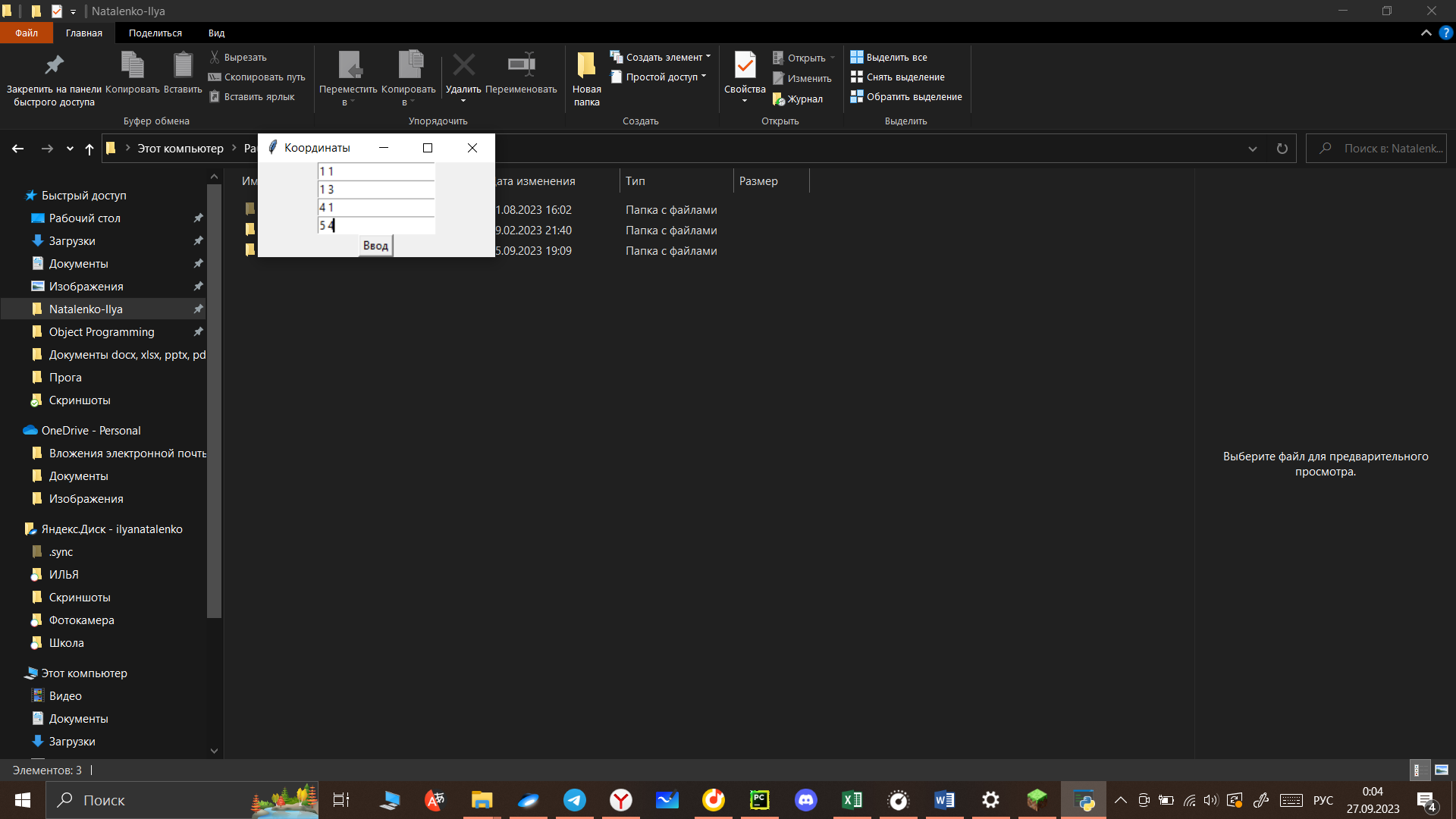
Следом идёт обновление экрана и цикл for для игровых событий. В данном варианте имеется только одно событие – закрытие окна с доской нажатием на крестик. При нажатии на него game\_on меняет состояние с True на False, и игровой цикл вместе со всей программой завершает свою работу.

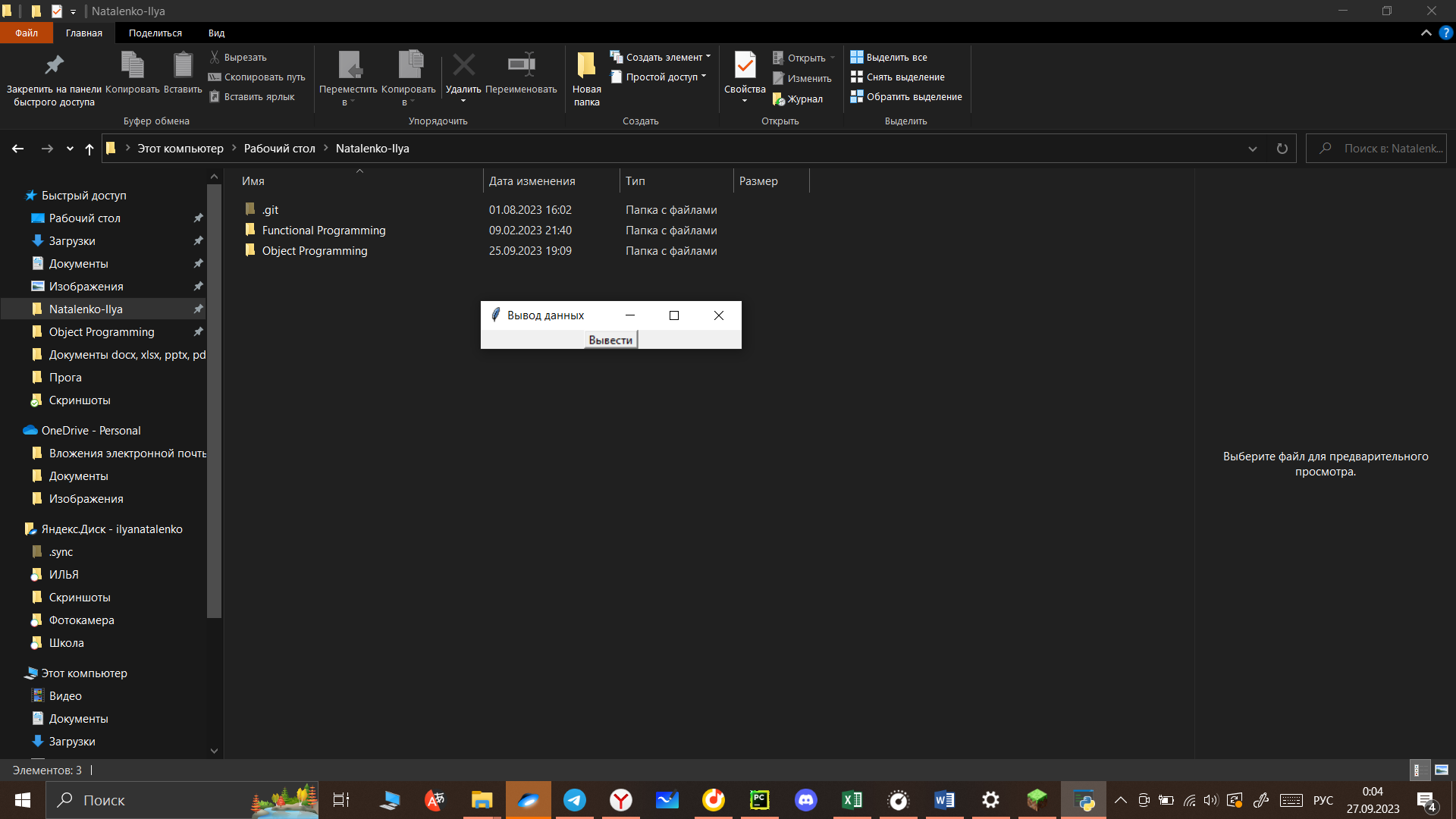
if show\_board:  
 *# Данные для шахматной доски.* colors = {'white': (255, 255, 255),  
 'black': (0, 0, 0),  
 'gray': (128, 128, 128),  
 'red': (200, 25, 25),  
 'blue': (25, 25, 200),  
 'yellow': (255, 255, 0)}  
 chess = pg.display.set\_mode((800, 800))  
 pg.display.set\_caption('Шахматы')  
 chess.fill(colors['gray'])  
 border\_width = 2  
 cell\_size = (800 - (N + 1) \* border\_width) / N  
 clock = pg.time.Clock()  
  
 game\_on = True  
 random\_variant = rnd.choice(solution\_list)  
 *# Игровой цикл.* while game\_on:  
 clock.tick(50)  
 c\_array = []  
 c\_coords\_array = []  
 *# Отрисовка чёрно-белых клеток.* for x in range(N):  
 c\_array.append([])  
 c\_coords\_array.append([])  
 for y in range(N):  
 coord\_x, coord\_y = border\_width \* (x + 1) + cell\_size \* x, border\_width \* (y + 1) + cell\_size \* y  
 if (x + y) % 2 == 0:  
 c\_array[x].append(Square(cell\_size, 'white').surf)  
 else:  
 c\_array[x].append(Square(cell\_size, 'black').surf)  
 c\_coords\_array[x].append((coord\_x, coord\_y, coord\_x + cell\_size, coord\_y + cell\_size))  
 chess.blit(source=c\_array[x][y], dest=(coord\_x, coord\_y))  
 count = 0  
 *# Отрисовка случайного найденного решения.* for x\_and\_y in random\_variant:  
 for new\_beaten\_cell in beaten\_cells(N, x\_and\_y[0], x\_and\_y[1]):  
 cell = Figure(new\_beaten\_cell[0], new\_beaten\_cell[1], 'beaten cell')  
 chess.blit(source=cell.surf, dest=(cell.pygame\_x, cell.pygame\_y))  
 if count >= K:  
 fig = Figure(x\_and\_y[0], x\_and\_y[1], 'new figure')  
 else:  
 fig = Figure(x\_and\_y[0], x\_and\_y[1], 'figure')  
 chess.blit(source=fig.surf, dest=(fig.pygame\_x, fig.pygame\_y))  
 count += 1  
 *# Обновление дисплея.* pg.display.flip()  
  
 *# Выход из цикла.* for i in pg.event.get():  
 if i.type == pg.QUIT:  
 game\_on = False

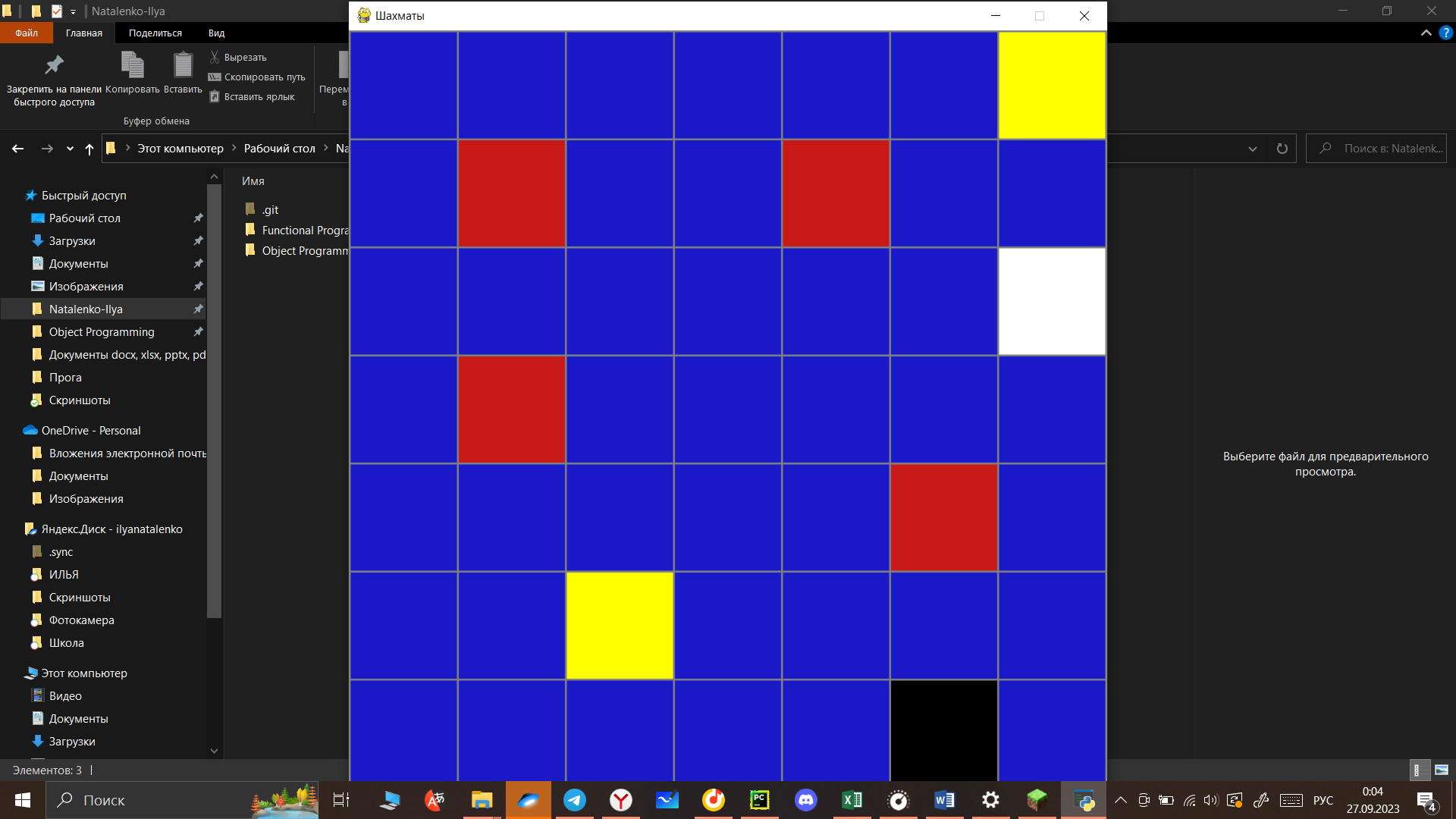
Для реализации программы с использованием ООП языка Python, а также разработки интерфейса при помощи пакетов tkinter (или  
customtkinter) + pygame (+OpenGL) был выбран вариант без использования customtkinter и OpenGL.

* 1. **Демонстрация работы программы**









* 1. **Листинг кода**

1. *# Импорт библиотек.*import tkinter as tk  
   import pygame as pg  
   import random as rnd  
   from itertools import permutations  
     
     
   *# Класс для клетки.*class Square:  
    def \_\_init\_\_(self, c\_size: float, color: str):  
    super(Square, self).\_\_init\_\_()  
    self.surf = pg.Surface((c\_size, c\_size))  
    if color == 'white':  
    self.surf.fill(colors['white'])  
    else:  
    self.surf.fill(colors['black'])  
    self.rect = self.surf.get\_rect()  
     
     
   *# Класс для фигуры.*class Figure:  
    def \_\_init\_\_(self, x: int, y: int, type: str):  
    self.x = x  
    self.y = y  
    self.type = type  
    self.surf = pg.Surface((cell\_size, cell\_size))  
    if type == 'figure':  
    self.surf.fill(colors['red'])  
    self.color = colors['red']  
    elif type == 'new figure':  
    self.surf.fill(colors['yellow'])  
    self.color = colors['yellow']  
    else:  
    self.surf.fill(colors['blue'])  
    self.color = colors['blue']  
    self.rect = self.surf.get\_rect()  
    self.pygame\_x = border\_width \* (x + 1) + cell\_size \* self.x  
    self.pygame\_y = border\_width \* (y + 1) + cell\_size \* self.y  
     
    def \_\_repr\_\_(self):  
    if self.type == 'figure' or self.type == 'new figure':  
    return 'Figure'  
    else:  
    return 'Beaten Cell'  
     
     
   *# Вызов окна ошибки.*def error():  
    error\_window = tk.Tk()  
    error\_window.title('Ошибка!')  
    error\_window.geometry('230x47')  
    tk.Label(master=error\_window, text='Неверный формат ввода!').pack()  
    tk.Button(master=error\_window, text='Ок', command=lambda: error\_window.destroy()).pack()  
    error\_window.mainloop()  
     
     
   *# Обработка ввода N, L, K, формирует окно с ошибкой, если формат не тот.*def enter():  
    global correct, N, L, K  
    N, L, K = n\_entry.get().strip(), l\_entry.get().strip(), k\_entry.get().strip()  
    window.destroy()  
    if not N.isdigit() or not L.isdigit() or not K.isdigit():  
    error()  
    else:  
    N, L, K = int(N), int(L), int(K)  
    if not 1 <= N <= 20 or not L >= 0 or not K >= 0:  
    error()  
     
     
   *# Обработка введённых координат.*def enter\_coords(entry\_fields: list):  
    for e in enter\_array:  
    string = e.get()  
    coords\_pair = tuple(string.split())  
    if len(coords\_pair) == 2 and False not in tuple(map(lambda x: x.isdigit(), coords\_pair)):  
    if 0 <= int(coords\_pair[0]) < N and 0 <= int(coords\_pair[1]) < N:  
    coords.append((int(coords\_pair[0]), int(coords\_pair[1])))  
    else:  
    error()  
    else:  
    error()  
    k\_window.destroy()  
     
     
   *# Создание окна с кнопкой для вывода.*def output():  
    global show\_board  
    show\_board = True  
    output\_window.destroy()  
     
     
   *# Поиск клеток под боем для Конь-Дракона (Король + Слон).*def beaten\_cells(size: int, x: int, y: int):  
    beaten = []  
    *# Клетки вокруг фигуры.* for row in range(-1, 2):  
    for col in range(-1, 2):  
    if 0 <= x + row < size and 0 <= y + col < size and (x + row, y + col) not in beaten:  
    beaten.append((x + row, y + col))  
    step = 2  
    *# Клетки по диагонали.* while step < size:  
    for row in range(x - step, x + step + 1, step \* 2):  
    for col in range(y - step, y + step + 1, step \* 2):  
    if 0 <= row < size and 0 <= col < size and (row, col) not in beaten:  
    beaten.append((row, col))  
    step += 1  
    return beaten  
     
     
   *# Создание окна для ввода.*window = tk.Tk()  
   window.title('Курсовой проект "Шахматы"')  
   window.geometry('335x150')  
   window.resizable(False, False)  
   n, n\_entry = tk.Label(master=window, text='Размер доски (N):'), tk.Entry(width=10)  
   n.pack()  
   n\_entry.pack()  
   n\_entry.focus()  
   l, l\_entry = tk.Label(master=window, text='Фигур надо поставить (L):'), tk.Entry(width=10)  
   l.pack()  
   l\_entry.pack()  
   k, k\_entry = tk.Label(master=window, text='Фигур уже расставлено (K):'), tk.Entry(width=10)  
   k.pack()  
   k\_entry.pack()  
   correct, N, L, K = False, 0, 0, 0  
   enter\_button = tk.Button(text="Ввод", command=enter)  
   enter\_button.pack()  
   window.mainloop()  
     
   *# Создание окна с вводом координат.*coords = []  
   k\_window = tk.Tk()  
   k\_window.geometry(f'250x{20 \* (K + 1)}')  
   k\_window.title('Координаты')  
   enter\_array = [tk.Entry(master=k\_window) for o in range(K)]  
   for i in range(K):  
    enter\_array[i].pack()  
   enter\_button = tk.Button(text="Ввод", command=lambda: enter\_coords(enter\_array))  
   enter\_button.pack()  
   if K > 0:  
    k\_window.mainloop()  
     
   *# Подготовка к вычислительной части.*all\_coords = list((m, n) for m in range(N) for n in range(N))  
   beaten\_coords = []  
     
   *# Получение координат под боем K фигур.*for fig in coords:  
    if fig not in beaten\_coords:  
    beaten\_coords.extend(beaten\_cells(N, fig[0], fig[1]))  
    beaten\_coords = list(set(beaten\_coords))  
    else:  
    error()  
     
   *# Получение координат не под боем K фигур.*beaten\_coords.sort()  
   not\_beaten\_coords = list(filter(lambda x: x not in beaten\_coords, all\_coords))  
     
   *# Перебор вариантов методом бинарной строки.*solution\_list = []  
   if L > 0:  
    for variant in set(map(lambda x: ''.join(list(x)), set(permutations(('1' \* L).zfill(len(not\_beaten\_coords)))))):  
    valid = True  
    start\_index = 0  
    temp\_beaten\_coords = beaten\_coords[:]  
    new\_coords = []  
    for i in range(L):  
    index = variant.find('1', start\_index, len(variant))  
    if not\_beaten\_coords[index] not in temp\_beaten\_coords:  
    temp\_beaten\_coords += beaten\_cells(N, not\_beaten\_coords[index][0], not\_beaten\_coords[index][1])  
    new\_coords.append(not\_beaten\_coords[index])  
    else:  
    valid = False  
    break  
    start\_index = index + 1  
    if valid:  
    solution\_list.append(coords + new\_coords)  
     
     
   *# Вывод в файл.*out\_file = open('output.txt', 'w')  
   for valid\_variant in solution\_list:  
    out\_file.write(str(valid\_variant)[1:-1] + '\n')  
     
   *# Cоздание окна вывода.*output\_window = tk.Tk()  
   output\_window.geometry('275x20')  
   output\_window.title("Вывод данных")  
   show\_board = False  
   output\_button = tk.Button(master=output\_window, text="Вывести", command=output)  
   output\_button.pack()  
   output\_window.mainloop()  
     
   *# Визуализация доски.*if show\_board:  
    *# Данные для шахматной доски.* colors = {'white': (255, 255, 255),  
    'black': (0, 0, 0),  
    'gray': (128, 128, 128),  
    'red': (200, 25, 25),  
    'blue': (25, 25, 200),  
    'yellow': (255, 255, 0)}  
    chess = pg.display.set\_mode((800, 800))  
    pg.display.set\_caption('Шахматы')  
    chess.fill(colors['gray'])  
    border\_width = 2  
    cell\_size = (800 - (N + 1) \* border\_width) / N  
    clock = pg.time.Clock()  
     
    game\_on = True  
    random\_variant = rnd.choice(solution\_list)  
    *# Игровой цикл.* while game\_on:  
    clock.tick(50)  
    c\_array = []  
    c\_coords\_array = []  
    *# Отрисовка чёрно-белых клеток.* for x in range(N):  
    c\_array.append([])  
    c\_coords\_array.append([])  
    for y in range(N):  
    coord\_x, coord\_y = border\_width \* (x + 1) + cell\_size \* x, border\_width \* (y + 1) + cell\_size \* y  
    if (x + y) % 2 == 0:  
    c\_array[x].append(Square(cell\_size, 'white').surf)  
    else:  
    c\_array[x].append(Square(cell\_size, 'black').surf)  
    c\_coords\_array[x].append((coord\_x, coord\_y, coord\_x + cell\_size, coord\_y + cell\_size))  
    chess.blit(source=c\_array[x][y], dest=(coord\_x, coord\_y))  
    count = 0  
    *# Отрисовка случайного найденного решения.* for x\_and\_y in random\_variant:  
    for new\_beaten\_cell in beaten\_cells(N, x\_and\_y[0], x\_and\_y[1]):  
    cell = Figure(new\_beaten\_cell[0], new\_beaten\_cell[1], 'beaten cell')  
    chess.blit(source=cell.surf, dest=(cell.pygame\_x, cell.pygame\_y))  
    if count >= K:  
    fig = Figure(x\_and\_y[0], x\_and\_y[1], 'new figure')  
    else:  
    fig = Figure(x\_and\_y[0], x\_and\_y[1], 'figure')  
    chess.blit(source=fig.surf, dest=(fig.pygame\_x, fig.pygame\_y))  
    count += 1  
    *# Обновление дисплея.* pg.display.flip()  
     
    *# Выход из цикла.* for i in pg.event.get():  
    if i.type == pg.QUIT:  
    game\_on = False
2. **Результат работы**

Программа работает исправно, однако по ходу написания выяснилось, что время её работы напрямую зависит от количества свободных клеток, оставшихся после поставленных изначально фигур (not\_beaten\_coords) и количества фигур, которые ещё нужно поставить (L). Оптимизация программы частично помогла уменьшить время, требуемое на выполнение.

1. **Заключение**

При работе над данным лабораторным проектом была проведена работа с основами функционального программирования языка Python, использованы принципы объектно-ориентированного программирования. Использованы методы работы с классами и пакетами Python. Освоены принципы построения UML диаграмм. Создано GUI приложение с использованием tkinter (или customtkinter) + pygame (+OpenGl).

В программе были использованы знания языка программирования Python, обозначенные в целях, что позволяет считать цели курсовой работы достигнутыми.

1. **Список используемых источников**
2. «Программирование на Python в примерах и задачах», Васильев А. Н., 2022 год, издательство «Бомбора».
3. «Простой Python. Современный стиль программирования». 2-е изд. Любанович Б.
4. «Справочник PYTHON. Кратко, быстро, под рукой», Кольцов Д. М. Дубовик Е. В., 2021 год, Издательство «Наука и Техника».
5. «Python 3 и PyQt 6. Разработка приложений», Прохоренок Н. А., 2023 год, Издательство БХВ.
6. «Высокопроизводительные Python-приложения. Практическое руководство по эффективному программированию», Миша Горелик, Йен Освальд, 2022 год, издательство «Бомбора».
7. https://pyga.me/docs/ref/sprite.html#pygame.sprite.spritecollide