МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный морской технический университет»  
 (СПБГМТУ)

ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ

КАФЕДРА КИБЕРФИЗИЧЕСКИХ СИСТЕМ

|  |  |
| --- | --- |
| Выполнил студент группы 20121 | Расщупкин Дмитрий Антонович |
| Проверил | Поделенюк Павел Петрович |

Санкт-Петербург

2023 год.

содержание

[Цели 3](#_Toc160563216)

[Формулировка задачи 4](#_Toc160563217)

[Разработка классов и UML диаграмма 7](#_Toc160563218)

[Результаты работы 8](#_Toc160563219)

[Заключение 19](#_Toc160563220)

# Цели

**Цель:** Анализ ходов фигуры на шахматной доске

При работе над данным лабораторным проектом будут затронуты следующие

вопросы:

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 задачи

**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- 1. Порядок, в котором фигуры перечислены в решении, не имеет значения. Если не было найдено ни одного решения, в файл необходимо записать no solutions.

3. 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). Ваша задача 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)

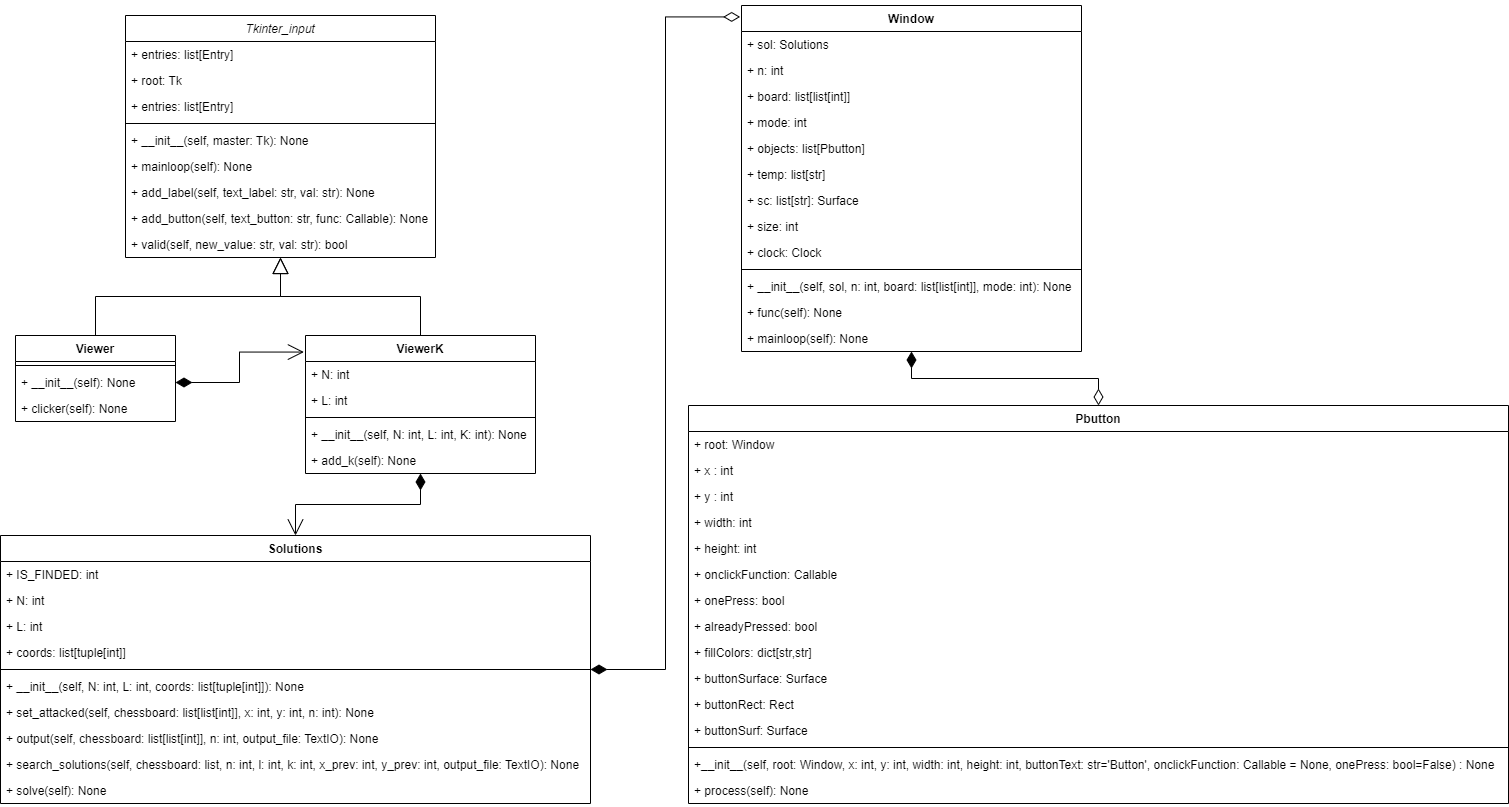
2.2.1. В данном интерфейсе есть шахматная доска и кнопка. На созданной шахматной доске показываются фигуры, расставленные пользователем и найденные алгоритмом, (нужно показать любое найденное решение). Визуализация пользовательских и найденных алгоритмом фигур должна отличаться (визуализация ходов одинаковая).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

*Дополнительная информация*:

1. Обязательное использование аннотации и комментариев

2. Проверка ввода данных.

# Разработка классов и UML диаграмма



Tkinter\_input – класс, отвечающий за создание полей и кнопок.

Viewer – класс, отвечающий за ввод параметров доски.

ViewerK - класс, отвечающий за первое окно Tkinter, которое содержит поля ввода N, L, K и кнопку перехода к вводу координат.

Solutions - класс, отвечающий за поиск решений.

Window - класс, отвечающий за создание окна с визуализацией доски, содержащей первое решение с кнопкой на Pygame.

PButton – кнопка для окна Pygame.

# Результаты работы

Реализация программы с помощью функционального решения. Ход работы:

На вход поступает 3 числа: N, L, K, где N -размерность квадратной доски, L - количество фигур, которые необходимо поставить самостоятельно и K – количество фигур, координаты которых даны. Далее идет K строчек, содержащих координаты фигур.

Объявляем глобальные переменные: MAX\_N (отвечает за максимальный размер доски), FREE (значение, при котором поле свободно), BUSY (значение, при котором поле занято самой фигурой), ATTACKED (значение, при котором поле находиться под атакой), IS\_FINDED (флаг, отвечающий за то, что существует хотя бы одна расстановка, а так же равняется количеству расстановок

Создаем двумерный массив chessboard длиной N, заполненный 0, считываем с файла input N, L, K.

Устанавливаем первые фигуры по координатам из файла input. С помощью глобальной переменной BUSY и функции set\_attacked устанавливаем значения на поля 1 и 2 соответственно из этого, значит на эти поля нельзя устанавливать какие-либо фигуры.

Функция set\_attacked расставляет 2 на поля, которые будут находиться под атакой с помощью двух циклов, которые ставят 2 на те поля, которые находятся по диагонали и на прямых по нашему условию.

Далее создается файл output и в последствии начинает выполняться рекурсивная функция search\_solution по параметрам: доски, размера доски, количества фигур, которые надо расставить, временной переменной и координат x и y.

Функция search\_solution создает дополнительную доску, в которую в последствии мы будем копировать нашу основную доску. Далее происходит перебор x, y при котором мы проверяем пустое ли поле (параметр FREE), если поле свободно, то копируем основную доску в дополнительную доску, после занимаем поле (параметр BUSY) и отмечаем поля под атакой.

Далее если количество фигур (k), которые мы уже расставили сравнялось с L (количество фигур, которые надо расставить) записываем расстановки в файл с помощью созданной функции output (просто записывает координаты наших фигур в файл) и устанавливаем флаг, что мы нашли хотя бы одну расстановку, дополнительно выводим в консоль дополнительную доску с помощью функции print\_board (функция выводит доску заменяя пустые поля (FREE) на 0, занятые фигурами поля (BUSY) на «#» и поля под атакой (ATTACKED) на «\*»).

Если количество фигур (k), которые мы расставили, не сравнялось с L (количество фигур, которые надо расставить), то мы к параметру k добавляем единицу и передаем дополнительную доску, n (размер доски), l (количество фигур, которые надо расставить) и передаем координаты x, y и повторяем итерацию.

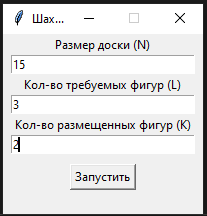
Реализация программы с использованием ООП языка Python.

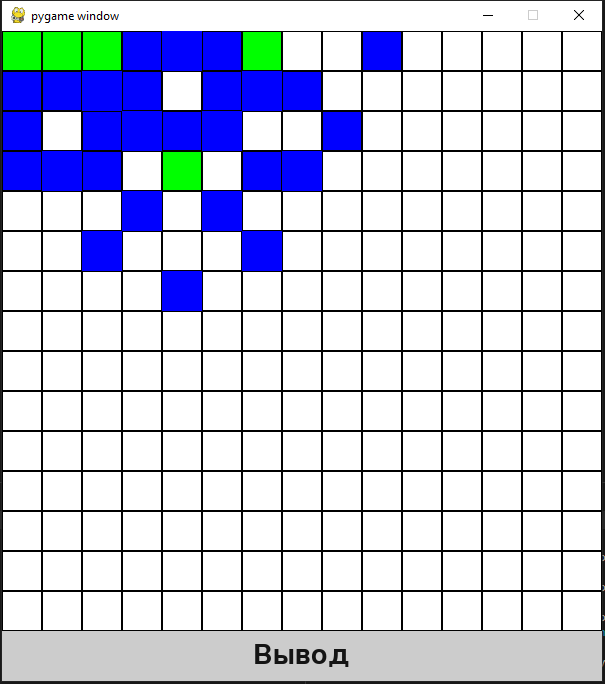
Ход работы.

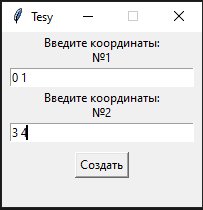
Сначала вызывается Viewer для того, чтобы считать данные о размере доски, количестве требуемых фигур, поставленных условием и количестве размещенных фигур,. Далее, после нажатия кнопки, Viewer блокируется и запускается следующее окно – ViewerK, куда вводятся координаты фигур K. По нажатию кнопки окна закрываются и данные передаются в класс Solutions, в котором просчитываются решения. Первое решение демонстрируется с помощью класса Window, который реализован с помощью библиотеки Pygame. Фигуры, поставленные условием, обозначены зеленым цветом, динамические фигуры обозначены красным, свободные клетки обозначены белым. Далее, по нажатию кнопки окно закрывается и рассчитываются оставшиеся ходы, которые записываются в файл output.txt.

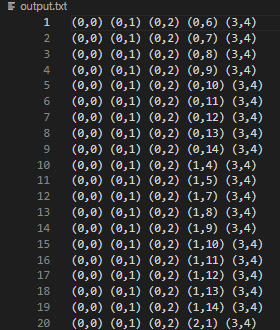
Демонстрации работы программы

Ввод значений и демонстрация первого решения:





****

****Листинг кода

**from** **typing** **import** Callable, TextIO

**import** **pygame** **as** **pg**

**from** **tkinter** **import** \*

**class** **Tkinter\_input**:

"""Абстрактный класс для создания полей и кнопок"""

**def** **\_\_init\_\_**(self, master: Tk) -> None:

"""

Инициализация

:param master:

Окно

"""

self.entries = []

self.root = master

self.root.resizable(False, False)

**def** **mainloop**(self) -> None:

"""Цикл окна"""

self.root.mainloop()

**def** **add\_label**(self, text\_label: str, val: str) -> None:

"""

Добавление надписи и поля для ввода

:param text\_label:

Надпись над полем для ввода

:param val:

Параметр для функции валидации

"""

lbl = Label(self.root, text=text\_label)

lbl\_text = Entry(self.root, width=**30**, validate='key',

validatecommand=(self.root.register(self.valid), '%P', val))

lbl.pack()

lbl\_text.pack()

self.entries.append(lbl\_text)

**def** **add\_button**(self, text\_button: str, func: Callable) -> None:

"""

Добавлении кнопки

:param text\_button:

Надпись кнопки

:param func:

Функция для исполнения при нажатии на кнопку

"""

myButton = Button(self.root, text=text\_button, command=func)

myButton.pack(pady=**10**)

**def** **valid**(self, new\_value: str, val: str) -> bool:

"""

Функция валидации

:param new\_value:

Значение из поля

:param val:

Параметр опредиляюций способ валидации

:return:

"""

**if** val == '1':

**try**:

**return** new\_value[-**1**].isnumeric()

**except**:

**return** new\_value == ''

**if** val == '2':

**try**:

**if** new\_value[-**1**].isnumeric():

**return** True

**elif** new\_value[-**1**].isascii() **and** new\_value[-**1**] == ' ' **and** new\_value.count(' ') <= **1**:

**return** True

**else**:

**return** False

**except**:

**return** new\_value == ''

**class** **Viewer**(Tkinter\_input):

"""Окно для ввода параметров шахматной доски"""

**def** **\_\_init\_\_**(self) -> None:

"""Создание полей окна"""

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

self.root.title("Шахматы")

self.root.geometry("200x180")

self.add\_label('Размер доски (N)', '1')

self.add\_label('Кол-во требуемых фигур (L)', '1')

self.add\_label('Кол-во размещенных фигур (K)', '1')

self.add\_button('Запустить', self.clicker)

**def** **clicker**(self) -> None:

"""Взятие значений из полей, удаление, создание окна для ввода K расставленных фигур"""

N = int(self.entries[**0**].get())

L = int(self.entries[**1**].get())

K = int(self.entries[**2**].get())

self.root.destroy()

second\_window = ViewerK(N, L, K)

**class** **ViewerK**(Tkinter\_input):

"""Окно для ввода координат K расставленных фигур"""

**def** **\_\_init\_\_**(self, N: int, L: int, K: int) -> None:

"""Создание полей окна"""

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

self.N = N

self.L = L

self.root.title("Tesy")

self.root.geometry("200x{0}".format(**50** \* K + **75**)) # добавить второй столбик, если больше 10

**for** i **in** range(**1**, K + **1**):

self.add\_label('Введите координаты:**\n**№{0}'.format(i), '2')

self.add\_button("Создать", self.add\_k)

**def** **add\_k**(self) -> None:

"""Запись координат расставленных фигур, удаление окна, поиск решений"""

coords = []

**for** i **in** self.entries:

coords.append(tuple(map(int, i.get().split(' '))))

self.root.destroy()

sol = Solutions(self.N, self.L, coords)

sol.solve()

**class** **Solutions**:

"""Класс для поиска решений (расстановок шахматных фигур)"""

MAX\_N = **20**

FREE = **0**

BUSY = **1**

ATTACKED = **2**

**def** **\_\_init\_\_**(self, N: int, L: int, coords: list[tuple[int]]) -> None:

"""

Инициализация основных параметров шахматной доски

:param N:

Размер доски

:param L:

Сколько фигур нужно еще расставить

:param coords:

Координаты расставленных фигур

"""

self.IS\_FINDED = **0** # флаг, что хоть одна расстановка найдена (= кол-во расстановок)

self.N = N

self.L = L

self.coords = coords

**def** **set\_attacked**(self, chessboard: list[list[int]], x: int, y: int, n: int) -> None:

"""

Просчет атаки фигуры

:param chessboard:

Шахматная доска двухмерный массив

:param x:

Координата

:param y:

Координата

:param n:

Размер шахматной доски

"""

**for** i **in** range(-**2**, **3**, **1**): # устанавливаем атаку на диагоналях

**if** i == **0**:

**continue**

**if** x + i >= **0** **and** y + i >= **0** **and** x + i < n **and** y + i < n:

chessboard[x + i][y + i] = self.ATTACKED

**if** x + i >= **0** **and** y - i >= **0** **and** x + i < n **and** y - i < n:

chessboard[x + i][y - i] = self.ATTACKED

**for** i **in** range(-**3**, **4**, **6**): # устанавливаем атаку на прямых

**if** x + i >= **0** **and** x + i < n:

chessboard[x + i][y] = self.ATTACKED

**if** y + i >= **0** **and** y + i < n:

chessboard[x][y + i] = self.ATTACKED

**def** **output**(self, chessboard: list[list[int]], n: int, output\_file: TextIO) -> None:

"""

Вывод расстановки в файл

:param chessboard:

Шахматная доска двухмерный массив

:param n:

Размер шахматной доски

:param output\_file:

Файл для вывода решения

"""

**for** x **in** range(**0**, n):

**for** y **in** range(**0**, n):

**if** chessboard[x][y] == self.BUSY:

output\_file.write("({0},{1}) ".format(x, y))

output\_file.write("**\n**")

**def** **search\_solutions**(self, chessboard: list[list[int]], n: int, l: int, k: int, x\_prev: int, y\_prev: int,

output\_file: TextIO) -> None:

"""

Поиск Решений

:param chessboard:

Шахматная доска двухмерный массив

:param n:

Размер шахматной доски

:param l:

Сколько фигур нужно еще расставить

:param k:

Сколько фигур уже расставлено

:param x\_prev:

Предыдущая координата

:param y\_prev:

Предыдущая координата

:param output\_file:

Файл для вывода решения

"""

board = [[**0**] \* self.MAX\_N **for** i **in** range(self.MAX\_N)] # доп. доска

**for** x **in** range(x\_prev,

n): # начинаем проходить по полям с прошлой итерации (или с начальной 0 0), ищем свободное поле

y1 = y\_prev **if** x == x\_prev **else** **0**

**for** y **in** range(y1, n):

**if** chessboard[x][y] == self.FREE: # если поле свободно, то

**for** x1 **in** range(**0**, n): # копируем доску в доп. доску

**for** y1 **in** range(**0**, n):

board[x1][y1] = chessboard[x1][y1]

board[x][y] = self.BUSY # занимаем поле

self.set\_attacked(board, x, y, n) # отмечаем поля, находящиеся под ударом

**if** k + **1** == l: # если кол-во расставленных фигур равно L, то

**if** self.IS\_FINDED == **0**:

self.IS\_FINDED += **1** # флаг, что хоть одна расстановка найдена

win = Window(self, n, board, **1**)

win.mainloop()

self.output(board, n, output\_file) # вывод расстановки в файл

**else**:

self.search\_solutions(board, n, l, k + **1**, x, y,

output\_file) # вызов ф-ции для следующей установки фигуры

**def** **solve**(self) -> None:

"""Поиск и вывод решений"""

output\_file = open("output.txt", mode="w") # для очищения файла вывода

output\_file.close()

chessboard = [[**0**] \* self.MAX\_N **for** i **in** range(self.MAX\_N)]

**for** x, y **in** self.coords: # ввод начальной расстановки из файла

**if** x >= self.N **or** y >= self.N: # no solutions

win = Window(self, self.N, chessboard, **0**)

win.mainloop()

**if** self.L == **0**: # если L == 0, то выводим в файл начальную расстановку

self.IS\_FINDED = **1**

**break**

chessboard[x][y] = self.BUSY # занимаем поле

self.set\_attacked(chessboard, x, y, self.N) # отмечаем поля, находящиеся под ударом

**with** open("output.txt", mode="a") **as** output\_file: # открываем файл для вывода перед вызовом рекурсивной функции

self.search\_solutions(chessboard, self.N, self.L, **0**, **0**, **0**, output\_file) # поиск всех возможных расстановок

**if** self.IS\_FINDED == **0**: # если расстановки не найдены, вывод no solutions

win = Window(self, self.N, chessboard, **0**)

win.mainloop()

**class** **Window**:

"""Окно для вывода решения"""

RES = **600**, **650**

WHITE = (**255**, **255**, **255**)

BLUE = (**0**, **0**, **255**)

GREEN = (**0**, **255**, **0**)

BLACK = (**0**, **0**, **0**)

FPS = **60**

**def** **\_\_init\_\_**(self, sol, n: int, board: list[list[int]], mode: int) -> None:

"""

Инициализация окна pygame

:param sol:

Ссылка на Solutions

:param n:

Размер шахматной доски

:param board:

Шахматная доска двухмерный массив

:param mode:

Параметр для вывода 0 - no solutions 1 - вывод окна с решением

"""

self.sol = sol

self.n = n

self.board = board

self.mode = mode

self.objects = []

self.temp = []

pg.init()

self.sc = pg.display.set\_mode(self.RES)

self.size = **600** // n

self.clock = pg.time.Clock()

self.sc.fill(self.WHITE)

customButton = Pbutton(self, **0**, **600**, **600**, **50**, 'Вывод', self.func)

**for** x **in** range(**0**, n):

**for** y **in** range(**0**, n):

**if** board[x][y] == self.sol.BUSY:

self.temp.append("{0},{1}".format(x, y))

**def** **func**(self) -> None:

"""Вывод решений в файл"""

**if** self.mode == **1**:

self.sol.solve()

**else**:

**with** open("output.txt", "w") **as** output\_file:

output\_file.write("no solutions")

exit()

**def** **mainloop**(self) -> None:

"""Цикл окна pygame"""

**while** True:

**for** event **in** pg.event.get():

**if** self.mode == **1**:

**for** i **in** range(len(self.temp)):

x1, y1 = self.temp[i].split(',')

x1, y1 = int(y1), int(x1)

**for** x **in** range(**0**, self.n):

**for** y **in** range(**0**, self.n):

**if** x1 == x **and** y1 == y:

pg.draw.rect(self.sc, self.GREEN,

(self.size \* x, self.size \* y, self.size, self.size))

pg.draw.rect(self.sc, self.BLACK,

(self.size \* x, self.size \* y, self.size, self.size), **1**)

**else**:

pg.draw.rect(self.sc, self.BLACK,

(self.size \* x, self.size \* y, self.size, self.size), **1**)

**for** i **in** range(-**2**, **3**, **1**): # устанавливаем атаку на диагоналях

**if** i == **0**:

**continue**

**if** x1 + i >= **0** **and** y1 + i >= **0** **and** x1 + i < self.n **and** y1 + i < self.n:

pg.draw.rect(self.sc, self.BLUE,

(self.size \* (x1 + i), self.size \* (y1 + i), self.size, self.size))

**if** x1 + i >= **0** **and** y1 - i >= **0** **and** x1 + i < self.n **and** y1 - i < self.n:

pg.draw.rect(self.sc, self.BLUE,

(self.size \* (x1 + i), self.size \* (y1 - i), self.size, self.size))

**for** i **in** range(-**3**, **4**, **6**): # устанавливаем атаку на прямых

**if** x1 + i >= **0** **and** x1 + i < self.n:

pg.draw.rect(self.sc, self.BLUE,

(self.size \* (x1 + i), self.size \* y1, self.size, self.size))

**if** y1 + i >= **0** **and** y1 + i < self.n:

pg.draw.rect(self.sc, self.BLUE,

(self.size \* x1, self.size \* (y1 + i), self.size, self.size))

**elif** self.mode == **0**:

f1 = pg.font.Font(None, **44**)

text1 = f1.render('No solutions', True, (**255**, **50**, **50**))

place = text1.get\_rect(center=(**300**, **275**))

self.sc.blit(text1, place)

**if** event.type == pg.QUIT:

pg.quit()

quit()

**for** object **in** self.objects:

object.process()

self.sc.blit(self.sc, (**0**, **0**))

pg.display.flip()

self.clock.tick(self.FPS)

**class** **Pbutton**:

"""Кнопка для окна Pygame"""

**def** **\_\_init\_\_**(self, root: Window, x: int, y: int, width: int, height: int, buttonText: str = 'Button',

onclickFunction: Callable = None, onePress: bool = False) -> None:

"""

Инициализация кнопки pygame

:param root:

Окно где будет отображаться кнопка

:param x:

Координата

:param y:

Координата

:param width:

Ширина

:param height:

Высота

:param buttonText:

Текст кнопки

:param onclickFunction:

Функция для исполнения при нажатии на кнопку

:param onePress:

Параметр определяющий возможность повторного нажатия на кнопку

"""

self.root = root

self.x = x

self.y = y

self.width = width

self.height = height

self.onclickFunction = onclickFunction

self.onePress = onePress

self.alreadyPressed = False

self.fillColors = {

'normal': '#cccccc',

'hover': '#666666',

'pressed': '#333333',

}

self.buttonSurface = pg.Surface((self.width, self.height))

self.buttonRect = pg.Rect(self.x, self.y, self.width, self.height)

self.buttonSurf = pg.font.Font(None, **40**).render(buttonText, True, (**20**, **20**, **20**))

root.objects.append(self)

**def** **process**(self) -> None:

"""Отслеживание нажатия на кнопку"""

mousePos = pg.mouse.get\_pos()

self.buttonSurface.fill(self.fillColors['normal'])

**if** self.buttonRect.collidepoint(mousePos):

self.buttonSurface.fill(self.fillColors['hover'])

**if** pg.mouse.get\_pressed(num\_buttons=**3**)[**0**]:

self.buttonSurface.fill(self.fillColors['pressed'])

**if** self.onePress:

self.onclickFunction()

**elif** **not** self.alreadyPressed:

self.onclickFunction()

self.alreadyPressed = True

**else**:

self.alreadyPressed = False

self.buttonSurface.blit(self.buttonSurf, [

self.buttonRect.width / **2** - self.buttonSurf.get\_rect().width / **2**,

self.buttonRect.height / **2** - self.buttonSurf.get\_rect().height / **2**

])

self.root.sc.blit(self.buttonSurface, self.buttonRect)

**def** **main**():

app = Viewer()

app.mainloop()

**if** \_\_name\_\_ == "\_\_main\_\_":

main()

# Заключение

В результате работы над курсовым проектом были улучшены знания в области работы с основами функционального программирования языка Python, был реализован первый полноценный проект с использованием основ ООП языка Python, в который также входят разработка классов и работа с UML диаграммами и был создавать GUI с использованием customtkinter и pygame.