МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет: Цифровых промышленных технологий  
Кафедра: Киберфизических систем

Курсовой проект

по дисциплине «Программирование»

Тема: «анализ ходов фигуры на шахматной доске»

Санкт-Петербург  
2023

СОДЕРЖАНИЕ

Оглавление

[СОДЕРЖАНИЕ 2](#_Toc137070028)

[ЦЕЛИ И ФОРМУЛИРОВКА ЗАДАНИЯ 3](#_Toc137070029)

[Разработка UML схемы 6](#_Toc137070030)

[Демонстрация работы программы 9](#_Toc137070031)

[Листинг кода 12](#_Toc137070032)

[Заключение 23](#_Toc137070033)

ЦЕЛИ И ФОРМУЛИРОВКА ЗАДАНИЯ

**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 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.

Я выбрал 2 способ для создания визуализации, а именно k строк с вводом данных координат фигур.

РАЗРАБОТКА UML СХЕМЫ

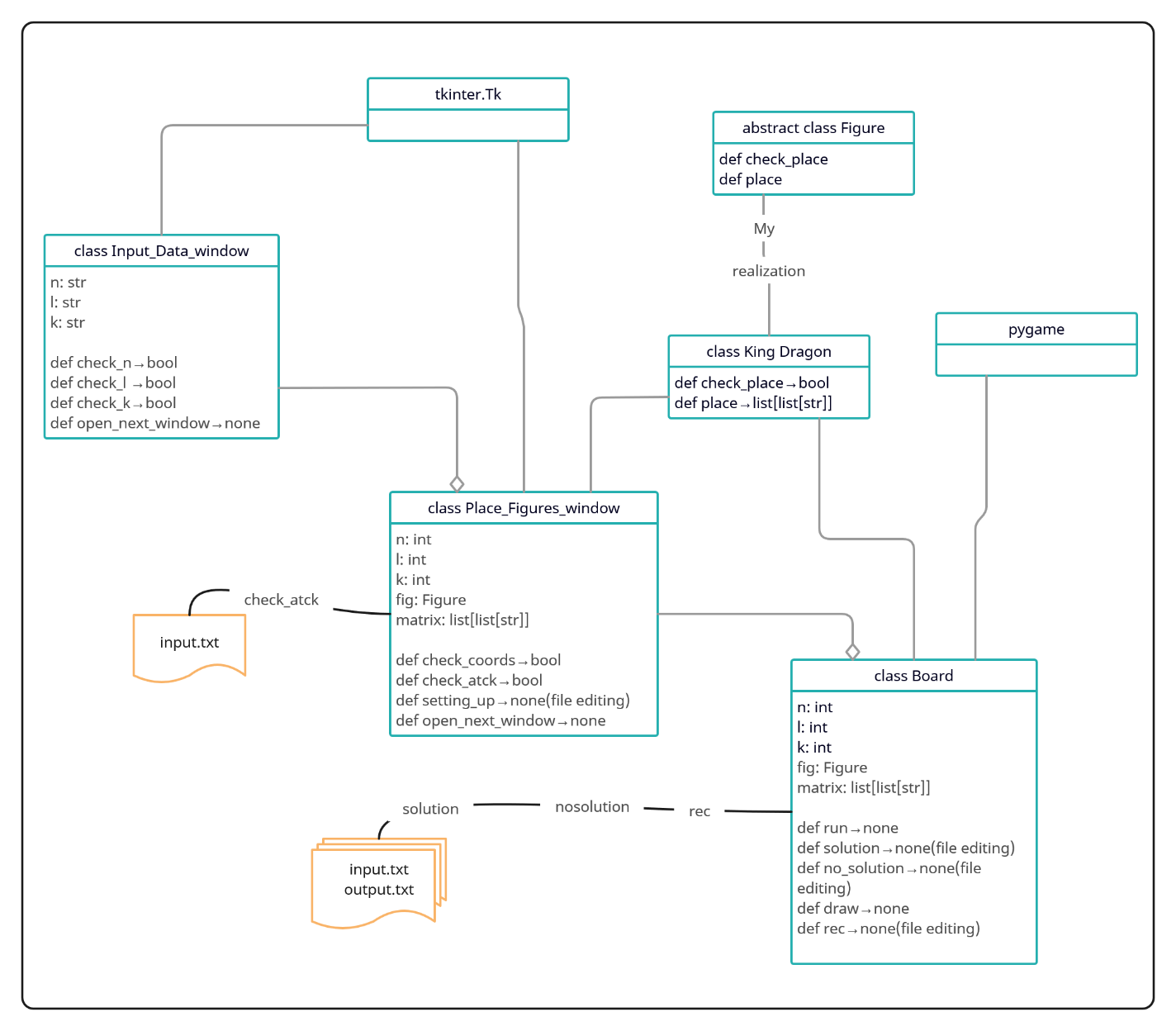
UML таблица – мною была разработана схема обобщаю в общих чертах самые важные отношения между классами разработанной мною программы. В ней указаны наследования классов, взаимосвязь классов, а также взаимодействия программы(чтение\запись) с внешними файлами. Схема указана ниже:

Рис. 1

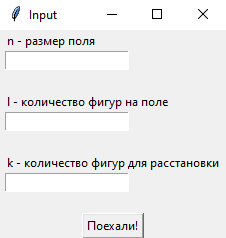
Более подробный описание программы, а именно классов и их методов представлено ниже:

|  |  |  |
| --- | --- | --- |
| **Класс** | **Метод** | **Краткое описание** |
| Input\_Data\_window(tk.Tk) | \_\_init\_\_(self)-> None | Метод инициализации объекта класса. Происходит создание всех виджетов модуля tkinter |
| Input\_Data\_window(tk.Tk) | check\_n(self, entry: str) -> bool | Метод проверки ввода значения n. Проверяет допустимость введенного символа |
| Input\_Data\_window(tk.Tk) | check\_l(self, entry: str) -> bool | Метод проверки ввода значения l. Проверяет допустимость введенного символа |
| Input\_Data\_window(tk.Tk) | check\_k(self, entry: str) -> bool | Метод проверки ввода значения k. Проверяет допустимость введенного символа |
| Input\_Data\_window(tk.Tk) | open\_next\_window(self)->None | Метод делает окончательную проверку на пустоту полей ввода. По прохождению проверки создаёт следующее окно класса Place\_Figures\_window(tk.Tk) передавая туда значения n, l, k: int |
| Place\_Figures\_window(tk.Tk) | \_\_init\_\_(self, n, l, k)->None | Метод инициализации объекта класса. В роли аргументов выступают значения полученные в предыдущем окне-классе: n, l, k: int. В методе происходит размещение объекта canvas и добавление на него объекта frame содержащего упакованные виджеты |
| Place\_Figures\_window(tk.Tk) | check\_coords(self, coords: str) -> bool | Метод получает значения из полей ввода и проверяет допустимость символов |
| Place\_Figures\_window(tk.Tk) | check\_atck(self, coord: tuple[int], all\_coords: list[tuple[int]], top\_limit: int) -> bool | Данный метод проверяет допустимость введенных координат по расположению фигур, вызывается из другого метода |
| Place\_Figures\_window(tk.Tk) | setting\_up(self, coords)->list | Подготавливает input файл и матрицу с раставленными фигурами для следующего окна-класса, вызывается из другого метода |
| Place\_Figures\_window(tk.Tk) | open\_next\_window(self)->None | Метод выполняет окончательную проверку полей ввода(вызывая метод check\_coords и check\_atck), а по успешному прохождению проверку форматирует координаты и отправляет их в качестве аргументов в метод setting\_up, после чего происходит вызов следующего окна-класса |
| Board | \_\_init\_\_(self, n, l, k, matrix)->None | Метод инициализации объекта класса, создаёт все необходимые атрибуты для создания Pygame окна, на вход получает значения n, l, k: int и matrix: list – подготовленную для корректной работы матрицу, все значения получаются от предыдущего класса-окна |
| Board | solution(self, method: str)->list  method = “single\_oop” | Метод вызывается с аргументом method: str, в данном случае происходят единичный поиск мест для расстановки фигур по матрице matrix и её вывод |
| Board | solution(self, method: str)->None  method = “full\_algorithm” | Метод вызывается при ином значение аргумента method, в этом случае происходят подготовка output и input файлов, а также запуска рекурсивного метода rec |
| Board | run(self)->None | Основной метод требуемый для корректной работы окна Pygame и обработки ввода, из него запускается метод solution при method = “full\_algorithm” |
| Board | draw(self)->None | Метода производящий отрисовку интерфейса окна GUI |
| Board | no\_solution(self)->None | Метод для обработки случаев отсутсвия решения, отображения в GUI и записи в файл |
| Board | rec(self, desk: list, figur\_lst: list, n, x\_now, y\_now, output)->None | Рекурсивный метод для обхода шахматный доски и поиска всех возможных вариантов расстановки фигур, осуществляет попутный вывод в файл output |

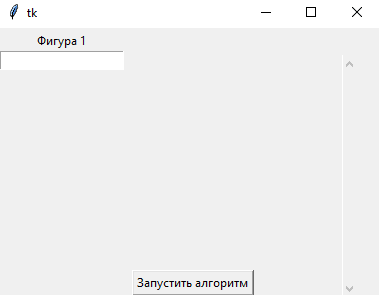
Табл. 1

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

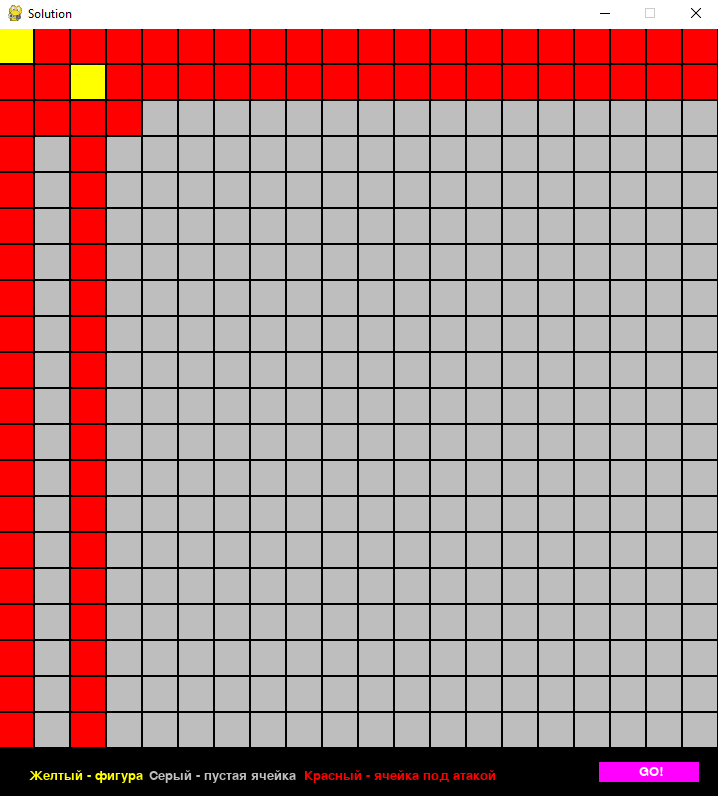
* Окно ввода начальных данны: n, l, k

**

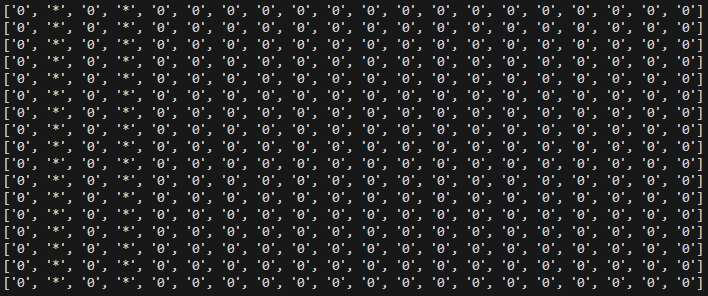
* Окно ввода координат фигур

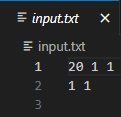
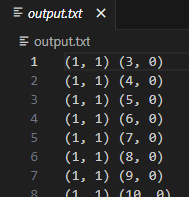
**

* Окно графического вывода одного из решений с возможностью запустить дальнейши алгоритм

**

* Пример вывода матрицы одного из решений в консоли, при запуске дальнейшего алгоритма

**

* Получившиеся input и output файлы
* ** **

ЛИСТИНГ КОДА

Ниже приведен листинг кода программы по файлам:

* main.py
* import window\_tk
* def main():
* window = window\_tk.Input\_Data\_window()
* if \_\_name\_\_ == '\_\_main\_\_':
* main()
* figure.py
* import abc
* class Figure:
* @abc.abstractmethod
* def check\_place(self, x, y, table):
* ...
* @abc.abstractmethod
* def place(self, x, y, table):
* ...
* class King\_Dragon(Figure):
* def check\_place(self, x, y, table) -> bool:
* if table[y][x] == "#" or table[y][x] == "\*": return False
* return True
* def place(self, x, y, table) -> list:
* for i in range(len(table)):
* table[i][x] = "\*"
* # добавляю по горизонтали
* for i in range(len(table)):
* table[y][i] = "\*"
* # ячейка слева
* if x > 0:
* table[y][x - 1] = "\*"
* # ячейка справа
* if x < len(table) - 1:
* table[y][x + 1] = "\*"
* # ячейка сверху
* if y > 0:
* table[y - 1][x] = "\*"
* # ячейка снизу
* if y < len(table) - 1:
* table[y + 1][x] = "\*"
* # слева-сверху
* if y > 0 and x > 0:
* table[y - 1][x - 1] = "\*"
* # справа-сверху
* if y > 0 and x < len(table) - 1:
* table[y - 1][x + 1] = "\*"
* # слева-снизу
* if y < len(table) - 1 and x > 0:
* table[y + 1][x - 1] = "\*"
* # справа-снизу
* if y < len(table) - 1 and x < len(table) - 1:
* table[y + 1][x + 1] = "\*"
* table[y][x] = "#"
* # self.coords = (x, y)
* return table
* window\_tk.py
* import tkinter as tk
* import figure
* import pygame
* class Input\_Data\_window(tk.Tk):
* def \_\_init\_\_(self):
* super().\_\_init\_\_()
* self.title("Input")
* # Ставим лейбл на ввод n потом ентри n потом лейбл на ошибку по n
* self.n\_label = tk.Label(self, text="n - размер поля")
* self.n\_label.pack(anchor="nw", padx=4)
* self.n\_input = tk.Entry(self, validate="key", validatecommand=(self.register(self.check\_n), "%P"))
* self.n\_input.pack(anchor="nw", padx=5)
* self.n\_err\_msg = tk.StringVar()
* self.n\_err\_label = tk.Label(self, textvariable=self.n\_err\_msg, foreground="red", wraplength=130)
* self.n\_err\_label.pack(anchor="nw", padx=4)
* # Ставим лейбл на ввод l потом ентри l потом лейбл на ошибку по l
* self.l\_label = tk.Label(self, text="l - количество фигур на поле")
* self.l\_label.pack(anchor="nw", padx=4)
* self.l\_input = tk.Entry(self, validate="key", validatecommand=(self.register(self.check\_l), "%P"))
* self.l\_input.pack(anchor="nw", padx=5)
* self.l\_err\_msg = tk.StringVar()
* self.l\_err\_label = tk.Label(self, textvariable=self.l\_err\_msg, foreground="red", wraplength=130)
* self.l\_err\_label.pack(anchor="nw", padx=4)
* # Ставим лейбл на ввод K потом ентри k потом лейбл на ошибку по k
* self.k\_label = tk.Label(self, text="k - количество фигур для расстановки")
* self.k\_label.pack(anchor="nw", padx=4)
* self.k\_input = tk.Entry(self, validate="key", validatecommand=(self.register(self.check\_k), "%P"))
* self.k\_input.pack(anchor="nw", padx=5)
* self.k\_err\_msg = tk.StringVar()
* self.k\_err\_label = tk.Label(self, textvariable=self.k\_err\_msg, foreground="red", wraplength=130)
* self.k\_err\_label.pack(anchor="nw", padx=4)
* self.button = tk.Button(self, text="Поехали!")
* self.button["command"] = self.open\_next\_window
* self.button.pack()
* self.mainloop()
* def check\_n(self, entry: str) -> bool:
* if (entry.isnumeric() and int(entry) <= 20) or entry == "":
* self.n\_err\_msg.set("")
* return True
* elif entry.isnumeric() and int(entry) > 20:
* self.n\_err\_msg.set("Размер поля - число не больше 20")
* return False
* elif "-" in entry:
* self.n\_err\_msg.set("Не может быть отрицательным")
* return False
* elif not entry.isnumeric():
* self.n\_err\_msg.set("Значение должно быть числом")
* return False
* def check\_l(self, entry: str) -> bool:
* try:
* if (entry.isnumeric() and int(entry) <= int(self.n\_input.get())\*\*2) or entry == "":
* self.l\_err\_msg.set("")
* return True
* elif "-" in entry:
* self.l\_err\_msg.set("Не может быть отрицательным")
* return False
* elif entry.isnumeric() and int(entry) > int(self.n\_input.get())\*\*2:
* self.l\_err\_msg.set("Количество фигур не может быть больше количества клеток")
* return False
* elif not entry.isnumeric():
* self.l\_err\_msg.set("Значение должно быть числом")
* return False
* except ValueError:
* self.l\_err\_msg.set("Размер доски не указан")
* return False
* def check\_k(self, entry: str) -> bool:
* try:
* if (entry.isnumeric() and int(entry) <= int(self.n\_input.get())\*\*2) or entry == "":
* self.k\_err\_msg.set("")
* return True
* elif "-" in entry:
* self.k\_err\_msg.set("Не может быть отрицательным")
* return False
* elif entry.isnumeric() and int(entry) > int(self.n\_input.get())\*\*2:
* self.k\_err\_msg.set("Количество фигур не может быть больше количества клеток")
* return False
* elif not entry.isnumeric():
* self.k\_err\_msg.set("Значение должно быть числом")
* return False
* except ValueError:
* self.k\_err\_msg.set("Размер доски не указан")
* return False
* def open\_next\_window(self):
* if self.n\_input.get() == "":
* self.n\_err\_msg.set("Не может быть пустым")
* elif self.l\_input.get() == "":
* self.l\_err\_msg.set("Не может быть пустым")
* elif self.k\_input.get() == "":
* self.k\_err\_msg.set("Не может быть пустым")
* elif (self.n\_input.get() != "" and self.l\_input.get() != "" and self.k\_input.get() != "") and (self.check\_n(self.n\_input.get()) and
* self.check\_l(self.l\_input.get()) and self.check\_k(self.k\_input.get())):
* n, l, k = int(self.n\_input.get()),  int(self.l\_input.get()), int(self.k\_input.get())
* self.destroy()
* Place\_Figures\_window(n, l, k)
* class Place\_Figures\_window(tk.Tk):
* def \_\_init\_\_(self, n, l, k):
* super(Place\_Figures\_window, self).\_\_init\_\_()
* self.entries = []
* self.maxsize(400, 400)
* self.n = n
* self.l = l
* self.k = k
* self.fig = figure.King\_Dragon()
* # Создаю Canvas чтобы можно было скроллить и фрейм в котором будут поля ввода
* self.canvas = tk.Canvas()
* self.canvas.pack(anchor="nw", expand=1)
* # Создаю лейбл и текст о неправильных координатах
* self.err\_msg = tk.StringVar()
* self.err\_label = tk.Label(textvariable=self.err\_msg, foreground="red", wraplength=175)
* self.err\_label.place(rely=0.75, relx=0.35)
* # Важная пометка: чтобы скроллить по канвазу нужно чтобы фрейм был больше канваза, как следствие нужно чтобы
* # высота менялась в зависимости от кол-ва фигур
* self.frame = tk.Frame(master=self.canvas, height=50\*l)
* # Цикл создающий l лейблов и ентриев с принадлежностью к фрейму, упаковываю при создании
* for i in range(1, l+1):
* tk.Label(self.frame, text=f"Фигура {i}").pack()
* self.entries.append(tk.Entry(self.frame, validate="key", validatecommand=(self.register(self.check\_coords),"%P")))
* self.entries[i-1].pack()
* # Фрейм упаковывать не надо он сразу как виджет отправляется в канваз
* self.canvas.create\_window(10, 10, anchor="nw", window=self.frame)
* # Создаю скроллбар и ставлю его в окне
* self.scroll = tk.Scrollbar(orient=tk.VERTICAL , command=self.canvas.yview)
* self.scroll.place(relheight=0.9, width=15, relx=0.9, rely=0.1)
* # В канвазе настраиваем скролл и присваиваем команду скролинга недавно созданому виджету
* self.canvas.configure(scrollregion=self.canvas.bbox("all"))
* self.canvas["yscrollcommand"] = self.scroll.set
* # Кнопка запуска алгоритма
* self.run\_button = tk.Button(text="Запустить алгоритм", command=self.open\_next\_window)
* self.run\_button.place(rely=0.9, relx=0.35)
* self.mainloop()
* def check\_coords(self, coords: str) -> bool:
* chrs = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "0", " "]
* space\_count = coords.count(" ") # кол-во пробелов
* for chr in coords:
* if chr not in chrs: # правильность символа
* self.err\_msg.set("Неправильный символ")
* return False
* self.err\_msg.set("")
* return True
* def check\_atck(self, coord: tuple[int], all\_coords: list[tuple[int]], top\_limit: int) -> bool:
* matrix = [["0" for \_ in range(int(self.n))] for \_ in range(int(self.n))]
* self.fig = figure.King\_Dragon()
* for i in range(top\_limit):
* matrix = self.fig.place(all\_coords[i][0], all\_coords[i][1], matrix)
* return figure.King\_Dragon().check\_place(coord[0], coord[1], matrix)
* def setting\_up(self, coords): # подготавливаю матрицу для отображения на доске в следующем окне и файлы для чтения
* desk = [["0" for \_ in range(self.n)] for \_ in range(self.n)]
* input = open("input.txt", "w+", encoding="utf-8")
* print(type(coords[0]))
* # записываем начальные данные n l k
* input.write(f"{self.n} ")
* input.write(f"{self.k} ")
* input.write(f"{self.l}\n")
* for c in coords: # записыванные координаты меняют матрицу через класс фигуры
* desk = self.fig.place(int(c[0]), int(c[1]), desk)
* for i in c: # попутно записываем координта в файл
* input.write(str(i+1) + " ") # с+1 т.к. в программе индексы с 0, а в файле с 1
* input.write("\n") # переводим каретку на след. строчку и там запишутся другие координаты
* input.close()
* return desk
* def open\_next\_window(self):
* entries = [entry.get() for entry in self.entries]
* frmted\_coords = ["" for \_ in range(len(entries))]
* i = 0
* right = True
* for coords in entries:
* coords = coords.strip()
* try:
* space = coords.index(" ")
* except:
* self.err\_msg.set(f"Фигура {i+1} имеет неправильные координаты")
* right = False
* break
* x = coords[:space]
* y = coords[space + 1:]
* if not x.isdigit() or not y.isdigit(): # если переменные оказываются не числами то строка неправтльная
* self.err\_msg.set(f"Фигура {i+1} имеет неправильные координаты")
* right = False
* break
* if not self.check\_coords(coords):
* right = False
* if int(x) > self.n:
* self.err\_msg.set("Координаты не могут быть больше доски")
* right = False
* elif int(y) > self.n:
* self.err\_msg.set("Координаты не могут быть больше доски")
* right = False
* if not self.check\_atck((int(x) - 1, int(y) - 1), frmted\_coords, i):
* self.err\_msg.set(f"Фигура {i+1} находится под боем")
* right = False
* frmted\_coords[i] = (int(x) - 1, int(y) - 1)
* i += 1
* not\_repeat = True
* for a in frmted\_coords:
* for b in frmted\_coords[frmted\_coords.index(a)+1:]:
* if a == b:
* self.err\_msg.set("Координаты не могут повторятся")
* not\_repeat = False
* if right and not\_repeat:
* self.destroy()
* matrix\_to\_show = self.setting\_up(frmted\_coords) # мне возвращается матрица с расставленными фигурами
* Board(self.n, self.l, self.k, matrix\_to\_show)
* class Board:
* def \_\_init\_\_(self, n, l, k, matrix):
* pygame.font.init()
* pygame.display.init()
* self.n = n
* self.l = l
* self.k = k
* pygame.init()
* self.resolution = (720, 768)
* self.screen = pygame.display.set\_mode(self.resolution)
* pygame.display.set\_caption("Solution")
* self.clock = pygame.time.Clock()
* self.matrix = matrix
* self.fig = figure.King\_Dragon()
* self.font = pygame.font.Font(None, size=20)
* # главное не перебарщивать со скоростью
* self.clock.tick(60)
* self.matrix = self.solution("single\_oop")
* if self.matrix != ["0\_0"]:
* self.draw()
* else:
* self.no\_solution()
* self.run()
* def solution(self, method: str):
* if method == "single\_oop":
* c = 0 # figures placed
* i = 0 # y
* j = 0 # x
* while c < self.k: # так как ищем одно решние для отображения, можно пробежать циклом while
* if self.fig.check\_place(j, i, self.matrix):
* self.matrix = self.fig.place(j, i, self.matrix)
* c += 1
* if i == self.n - 1 and j == self.n - 1:
* break
* if j == self.n - 1:
* i += 1
* j = 0
* else:
* j += 1
* if c < self.k: return ["0\_0"]
* return self.matrix
* elif method == "full\_algorithm": # большая подготовка перед большой рекурсией
* inpt = open("input.txt", "r+", encoding="utf-8")
* out = open("output.txt", "w+", encoding="utf-8")
* lines = inpt.readlines() # считали строки
* lines = [e.strip("\n" ).split(" ") for e in lines] # Сформатировали строки
* fig = figure.King\_Dragon()
* figures = []
* s = int(lines[0][0]) # размечикс
* c = int(lines[0][1]) # скок надо поставить
* for e in lines[1:]:
* figures.append((int(e[0]), int(e[1])))
* matrix = [["0" for \_ in range(s)] for \_ in range(s)]
* for e in figures:
* matrix = fig.place(e[0], e[1], matrix)
* inpt.close()
* self.rec(matrix, figures, c, -1, 0, out)
* out.close()
* out = open("output.txt", mode="r+", encoding="utf-8")
* lines = out.readlines()
* if lines == []:
* print("No solution", file=out)
* out.close()
* else:
* out.close()
* out = open("output.txt", mode="r+", encoding="utf-8")
* lines = out.readline().replace("(", "").replace(")", "").replace(",", "").split(" ") # считали строки
* matrix\_to = [["0" for \_ in range(s)] for \_ in range(s)]
* for i in range(0, len(lines), 2):
* matrix\_to = fig.place(int(lines[i]), int(lines[i+1]), matrix\_to)
* out.close()
* for e in matrix\_to: # единичный вывод в консоль
* print(e)
* def run(self):
* run = True
* while run:
* mouse\_pos = pygame.mouse.get\_pos()
* for event in pygame.event.get():
* # закрытие окна
* if event.type == pygame.QUIT:
* run = False
* elif event.type == pygame.MOUSEBUTTONDOWN:
* if 600 <= mouse\_pos[0] <= 600 + 100 and 733 <= mouse\_pos[1] <= 733 + 20:
* self.solution(method="full\_algorithm")
* pygame.display.flip()
* def draw(self):
* # Создаю легенду
* figs\_text = self.font.render("Желтый - фигура", 1, "Yellow")
* self.screen.blit(figs\_text, (30, 740))
* slot\_text = self.font.render("Серый - пустая ячейка", 1, "Grey")
* self.screen.blit(slot\_text, (150, 740))
* atck\_text = self.font.render("Красный - ячейка под атакой", 1, "Red")
* self.screen.blit(atck\_text, (305, 740))
* # создаю кнопку и текст для полного алгоритма с записью в файл
* full\_algorithm\_button = pygame.draw.rect(self.screen, "Magenta", (600,733, 100, 20))
* full\_algorithm\_text = self.font.render("GO!", 1, "White")
* self.screen.blit(full\_algorithm\_text, (640, 736))
* size = 720 / self.n # размер ячеек
* cy = 0 # врубаю счетчик чтобы располагать ячейки по y
* for y in self.matrix:
* cx = 0 # врубаю счетчик чтобы располагать ячейки по x
* for x in y:
* if self.matrix[cy][cx] == "0":
* drawrect = pygame.draw.rect(self.screen, "Grey", (cx\*size, cy\*size, size-2, size-2))
* if self.matrix[cy][cx] == "\*":
* drawrect = pygame.draw.rect(self.screen, "Red", (cx \* size, cy \* size, size-2, size-2))
* if self.matrix[cy][cx] == "#":
* drawrect = pygame.draw.rect(self.screen, "Yellow", (cx \* size, cy \* size, size-2, size-2))
* cx += 1
* cy += 1
* def no\_solution(self):
* self.font = pygame.font.Font(None, size=60)
* no\_solution = self.font.render("No solution", 1, "Red")
* self.screen.blit(no\_solution, (260, 360))
* output = open("output.txt", "w+", encoding="utf-8")
* output.write("no solution")
* output.close()
* def rec(self, desk: list, figur\_lst: list, n, x\_now, y\_now, output):
* if n == 0:
* print(\*figur\_lst, file=output)
* return
* new\_figurs = list()
* i = y\_now
* j = x\_now
* while True:
* if i == len(desk) - 1 and j == len(desk) - 1:
* break
* if j == len(desk) - 1:
* i += 1
* j = 0
* else:
* j += 1
* ability = True
* for e in figur\_lst:
* if (j == e[0] or i == e[1]) or (abs(j - e[0]) <= 1 and abs(i - e[1]) <= 1):
* ability = False
* if ability:
* new\_figurs.append(tuple([j, i]))
* for e in new\_figurs:
* self.rec(desk, figur\_lst + [e], n - 1, e[0], e[1], output)

ЗАКЛЮЧЕНИЕ

Мною была проведена работа по созданию GUI(Graphical User Interface), для алгоритма расстановки фигур шахмат, с возможностью последующей нетрудозатраной замены классов фигур с сохранением корректной работы графического интерфейса. В процессе работы я ознакомился с модулями языка Python: tkinter и Pygame. Мною была создана удобная UML – схема упрощающая планирование и упорядочивающая создание программы. В конечном итоге я укрепился в базовом понимании работы классов и прицнипов прогромированния ООП на языке Python.