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 | Шпаковский Евгений Евгеньевич |
| Проверил | Поделенюк Павел Петрович |

Санкт-Петербург

2023 год.

содержание

[Цели 3](#_Toc136518822)

[Формулировка задачи 4](#_Toc136518823)

[Разработка классов и UML диаграмма 7](#_Toc136518824)

[Результаты работы 9](#_Toc136518825)

[Заключение 26](#_Toc136518826)

[Список используемых источников 27](#_Toc136518827)

# 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)

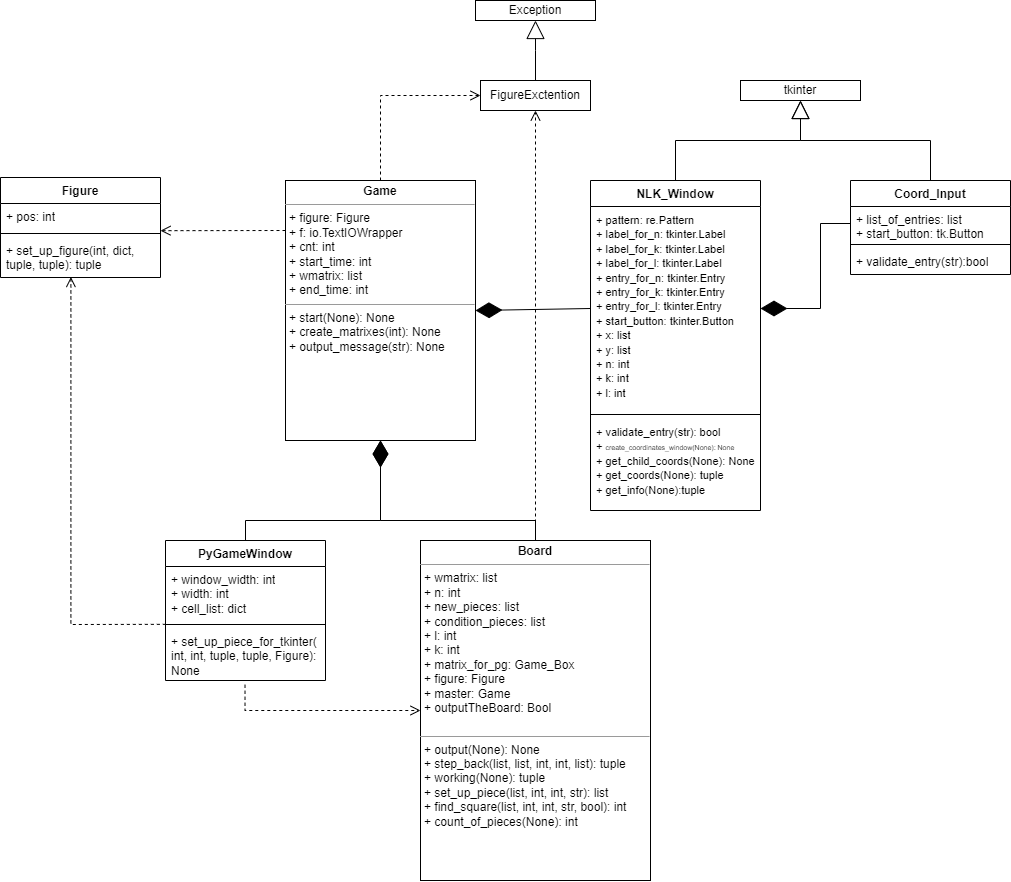
2.2.1. В данном интерфейсе есть шахматная доска и кнопка. На созданной шахматной доске показываются фигуры, расставленные пользователем и найденные алгоритмом, (нужно показать любое найденное решение). Визуализация пользовательских и найденных алгоритмом фигур должна отличаться (визуализация ходов одинаковая).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



FigureException(Exception) – исключение, вызываемое, когда ставящаяся фигура находится под атакой.

Game – класс, отвечающий за первоначальный ввод и формирование данных, также отвечает за завершение всей программы.

Figure - класс, отвечающий за нашу фигуру - KingHorse.

Board - класс, определяющий доску, содержащую фигуры. Также производит расчеты всевозможных ходов

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

NLK\_Window(tk.Tk) - класс, отвечающий за первое окно Tkinter, которое содержит поля ввода N, L, K и кнопку перехода к вводу координат

Coord\_Input - класс, отвечающий за ввод координат фигур, поставленных в ходе условия.

# 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.

Первым шагом будет реализация доски как двумерного массива размером N×N, заполненного нулями, причем нумерация каждой ячейки может идти как (i; j), где i – номер строки от 0 до 7, а j – номер столбца от 0 до 7, так и числом от 0 до N-1.

Расстановка фигур из условия осуществляется простым заполнением двумерного массива соответствующими значениями, где “#” - фигура и “\*” – клетки, которые находятся под атакой какой-либо фигуры.

Таким образом получается двумерный массив с исходными данными, который имеет (или не имеет) свободные и занятые ячейки.

Пронумеруем L-фигуры номерами от 1 до L.

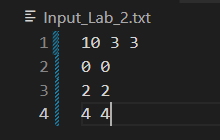
Алгоритм расстановки L-фигур таков:

1. Ищется ближайшая свободная ячейка к ячейке с индексом 0. Обозначим эту свободную ячейку буквой F
2. Ставится первая фигура на ячейку F. Заполняется поле атак этой фигуры.
3. Снова ищется ближайшая свободная ячейка к ячейке с индексом 0. Перезаписывается ячейка F.
4. Ставится вторая фигура на ячейку F. Заполняется поле атак этой фигуры.
5. Шаги 3-4 повторяются до последней ячейке, не включая ее, для последней фигуры алгоритм особый.
6. Последняя фигура (фигура с номером L) проходится по всем оставшимся свободным ячейкам от F до N-1, записывая решения в файл.
7. Фигура с номером L-1 смещается на следующее свободное от нее место.
8. Шаг 6-7 повторяется вплоть до того момента, пока это возможно. Даже если фигуру L нельзя поставить фигура L-1 продолжает двигаться до ячейки N-1 и проверять возможные варианты.
9. Когда фигура L-1 дошла до ячейки N-1, она убирается с доски вместе с остальными фигурами большего номера и 7 шаг происходит уже с фигурой L-2.

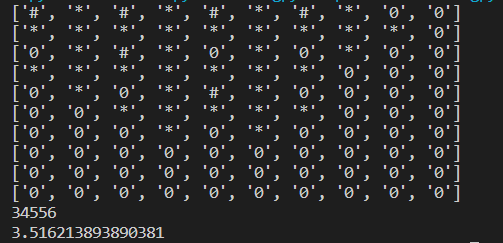
Таким образом у нас происходит смещение всех фигур с самой «младшей» (номер L) до самой старшей (номер 1) с записью всевозможных решений в файл.

Демонстрация работы программы

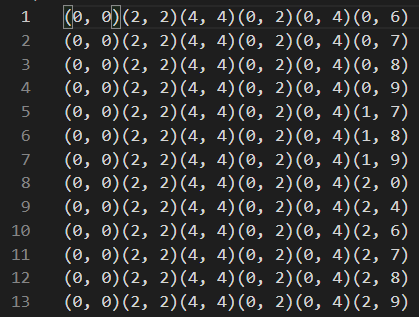
Для входных данных:



Вывод в консоль:



Вывод в файл:



Листинг кода







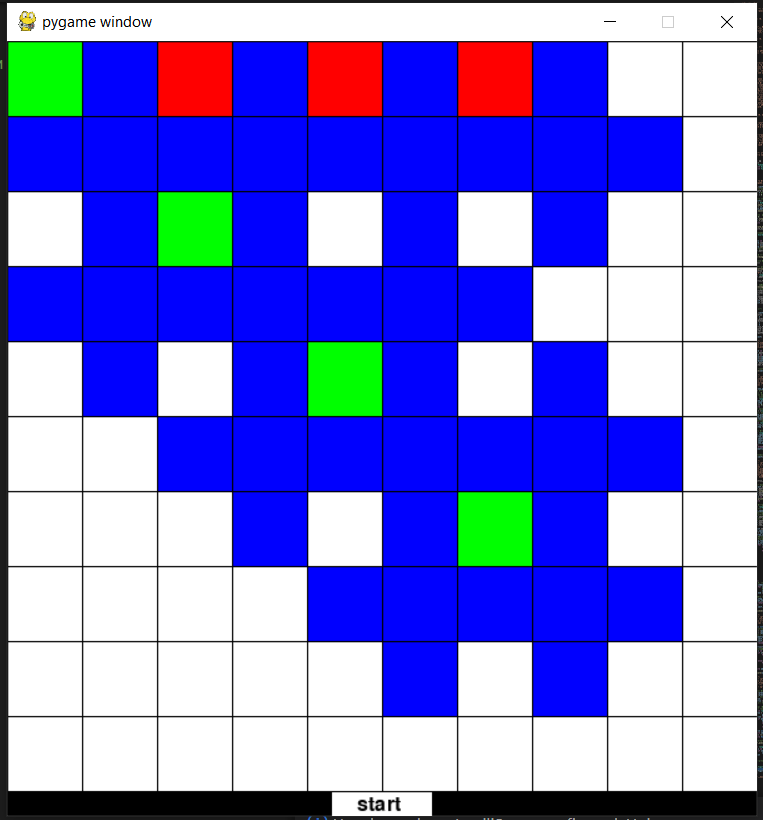
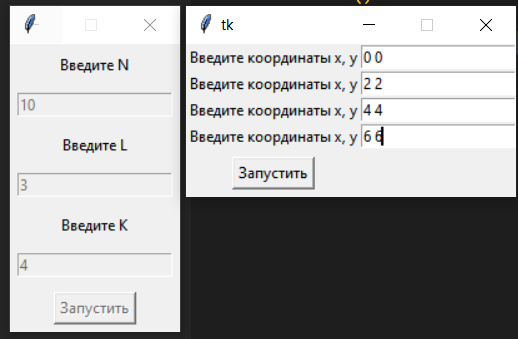


Реализация программы с использованием ООП языка Python.

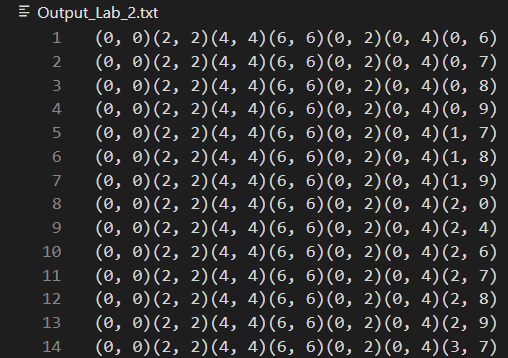
Ход работы.

Сначала вызывается NLK\_Window для того, чтобы считать данные о размере доски, количестве фигур, поставленных условием и количестве фигур, которые необходимо расставить программно. Далее, по нажатию кнопки, NLK\_Window блокируется и запускается следующее окно – Coord\_Input, куда вводятся координаты фигур K. По нажатию кнопки окна закрываются и данные передаются в класс Game, где обрабатываются и передаются классу Board, который просчитывает решения согласно функциональному решению. Первое решение демонстрируется с помощью класса PyGameWindow, 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\_Lab\_2.txt.

Демонстрации работы программы

Ввод значений и демонстрация первого решения:

Вывод в файл:

****

Листинг кода

Файл main.py:





















Файл figure.py:





Файл gui.py:







Файл exceptions.py:



# Заключение

В результате работы над курсовым проектом я укрепил свои знания в области работы с основами функционального программирования языка Python; впервые реализовал полноценный проект с использованием основ ООП языка Python; научился работать с UML диаграммами, создавать GUI с использованием customtkinter и pygame, правильно оформлять код с помощью docstring, правильно использовать аннотацию с помощью модуля Typing.

Список используемых источников

1. [Шпаргалка по UML-диаграммам](http://hitrome.ru/home/-/asset_publisher/2XeqHUh5V1PA/content/%25D0%25A8%25D0%25BF%25D0%25B0%25D1%2580%25D0%25B3%25D0%25B0%25D0%25BB%25D0%25BA%25D0%25B0-%25D0%25BF%25D0%25BE-%25D0%25BE%25D0%25B1%25D0%25BE%25D0%25B7%25D0%25BD%25D0%25B0%25D1%2587%25D0%25B5%25D0%25BD%25D0%25B8%25D1%258F%25D0%25BC-uml)
2. <https://github.com/pygame/pygame>
3. <https://docs-python.ru/standart-library/modul-typing-python/>
4. <https://docs-python.ru/standart-library/modul-functools-python/funktsija-partial-modulja-functools/>