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

ЛАБОРАТОРНАЯ РАБОТА

По дисциплине «Программирование»

Выполнил:

Кузьмин Кирилл Алексеевич гр. 20121

Проверил:

Санкт-Петербург

2024

Оглавление

[1. Цели и формулировка задачи 3](#_Toc187676633)

[2. Результаты работы 4](#_Toc187676634)

[2.1. Реализация программы с использованием функционального программирования языка Python 4](#_Toc187676635)

[2.1.1. Ход работы 4](#_Toc187676636)

[2.1.2. Демонстрация работы программы 5](#_Toc187676637)

[2.1.3. Листинг кода 5](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ](#_Toc187676639) 8

# Цели и формулировка задачи

При работе над данной лабораторной работой будут затронуты следующие вопросы:

1. Работа с текстовыми файлами,
2. Работа с локальными и глобальными переменными,
3. Работа с несколькими функциями, а не только с main,
4. Нисходящий метод разработки программ,
5. Анализ ходов.

# Результаты работы

## **Реализация программы с использованием функционального программирования языка Python**

## Ход работы

1. Проанализировав задачу, я обнаружил следующие полезные особенности:
   1. Если клетка не находится под боем другой фигуры и не занята ею, то можно поставить на эту клетку фигуру и она не будет под боем.
   2. Все возможные ходы фигуры, поставленной на клетку с координатами x и y можно выразить как:  
      (x + 1, y), (x, y - 1), (x - 1, y), (x, y + 1), (x, y + 2), (x, y - 2), (x - 2, y), (x + 2, y)
2. Создаю функцию figure\_moves – генератор, убирающий фигуры с указанной координаты, выходящие за рамки нашего поля
3. Создаю функцию place\_figure, ставящую фигуру на клетку с указанными координатами.
4. Создаю функцию remove\_figure, убирающую фигуру с клетки с указанными координатами.
5. Обе точки остановки возвращают None, но в первой ситуации мы записываем получившуюся расстановку в файл
6. Переменные side и file делаем глобальными для того, чтобы не добавлять размерность игрового поля в параметры каждой функции, и для того, чтобы записывать данные в файл в рамках рекурсии.
7. Считываю данные из файла, расставляем фигуры из файла на поле
8. Вызываю рекуррентную функцию
9. В рекуррентной функции, реализуется следующий алгоритм:
   1. Ставлю фигуру на поле, перехожу на более низкий уровень рекурсии
   2. Если это была последняя фигура из тех, которые необходимо было поставить, то записываю получившуюся расстановку в файл, вывожу поля с расстановками в консоль (если не закомментировано), и возвращаюсь на предыдущий уровень рекурсии
   3. Если вернулся с более низкого уровня рекурсии, снимаю фигуру и перехожу к следующей свободной клетке
10. Точками остановки в рекурсии будут две ситуации:
    1. Поставлены все «дополнительные» фигуры записываю расстановку в файл
    2. Мы не можем поставить на поле больше ни одной фигуры
11. Выводим no solution в файл если не было решений

## Демонстрация работы программы

## 

## Пример входных данных:

## 2 1 1

## 0 1

## Вывод данных в консоль:

\* #

# \*

**Вывод данных в файл:**

(0, 1), (1, 0)

## Пример входных данных:

2 1 2

0 1

1 0

## Вывод данных в консоль:

## Вывод данных в файл:

no solution

## Листинг кода

1. from typing import Generator
2. # Функция для вывода нынешнего состояния доски: "#" - фигура, "0" - свободная клетка, '\*' - клетка под боем(число указывает сколько фигур бьют на эту клетку)
3. def print\_field(field: list[list[int]]) -> None:
4. for i in range(side):
5. for j in field[i]:
6. if j==-1:
7. print('#', sep='', end=' ')
8. elif j>0:
9. print('\*', sep='', end=' ')
10. else:
11. print('0', sep='', end=' ')
12. print()
13. print('-' \* (2 \* len(field[0]) - 1))
14. # Генератор, отсекающий ходы фигуры, выходящие за рамки игрового поля
15. def figure\_moves(x: int, y: int) -> Generator[tuple[int, int], None, None]:
16. for i in ((x + 1, y), (x, y - 1), (x - 1, y), (x, y + 1), (x, y + 2), (x, y - 2), (x - 2, y), (x + 2, y)):
17. if not (i[0] > side - 1 or i[1] > side - 1 or i[0] < 0 or i[1] < 0):
18. yield i
19. # Функция, ставящая фигуру на указанную клетку
20. def place\_figure(x: int, y: int, field: list[list[int]])-> list[list[int]]:
21. # Расстановка клеток боя на поле
22. for i in figure\_moves(x, y):
23. field[i[1]][i[0]] += 1
24. # Ставим саму фигуру
25. field[y][x] = -1
26. return field
27. # Функция, убирающая фигуру с указанной клетки
28. def remove\_figure(x: int, y: int, field: list[list[int]]) -> list[list[int]]:
29. # Убираем бой этой фигуры с тех клеток, на которых он был
30. for i in figure\_moves(x, y):
31. field[i[1]][i[0]] -= 1
32. # Убираем саму фигуру
33. field[y][x] = 0
34. return field
35. # Рекуррентная функция, с точкой остановки "Все дополнительные фигуры поставлены"
36. def recursion(place: int, current\_pos: int, field: list[list[int]], figures:list = [])-> None:
37. # Точка остановки, если нам не нужно больше ставить фигуры на поле, мы записываем нынешнюю расстановку в файл и выходим из этой ветки рекурсии
38. if place==0:
39. # Здесь мы выводим удачные расстановки в консоль, по умолчанию закомментировано для большей оптимизации
40. print\_field(field)
41. file.write(",".join(map(str, const\_figures + figures)) + '\n')
42. return
43. # Тело рекурсии
44. else:
45. # Представляем нашу шахматную доску в виде последовательности клеток, избавляясь от двухмерности. Идти по доске мы начинаем с "Нулевой позиции"
46. for pos in range(current\_pos, side \*\* 2):
47. #Если мы находим пустую клетку, то ставим на неё фигуру, записываем её местоположение, меняем значение "Нулевой позиции" на следующую клетку после той, на которую мы поставили фигуру
48. if field[pos // side][pos % side] == 0:
49. field = place\_figure(pos % side, pos // side, field)
50. figures.append((pos % side, pos // side))
51. recursion(place - 1, pos + 1, field, figures)
52. # По достижению точки остановки мы возвращаемся сюда и переходим в другую ветку, снимая поставленную ранее на эту клетку фигуру
53. figures.pop(len(figures) - 1)
54. field = remove\_figure(pos % side, pos // side, field)
56. # Объявление глобальных переменных. 1)side - Сторона поля 2)file - Само шахматное поле 3)const\_figures - Фигуры, которые я не могу двигать
57. global side
58. global file
59. global const\_figures
60. const\_figures: list = list()
62. # Считывание данных из файла
63. with open('input.txt','r') as f:
64. side, place, field\_side = map(int, f.readline().split())
65. # Генерация поля, расстановка фигур из файла
66. field: list[list[int]] = [[0] \* side for i in range(side)]
67. for i in range(field\_side):
68. x, y = map(int, f.readline().split())
69. const\_figures.append((x, y))
70. field: list[list[int]] = place\_figure(x, y, field)
71. #Запуск рекурсии, запись всех возможных расстановок в файл
72. file = open("output.txt", 'w')
73. recursion(place, 0, field)
74. file.close()
75. #Вывод no solutions в файл если нет решений
76. with open("output.txt",'r') as f:
77. file = f.readlines()
78. if len(file) == 0:
79. with open("output.txt", 'w') as f:
80. f.write('no solutions')

# ЗАКЛЮЧЕНИЕ

При работе я стал глубже понимать рекуррентные алгоритмы, отработал разбиение крупной задачи на более мелкие, работал с несколькими функциями, появилось более чёткое представление о работе с глобальными переменными, углубил знания в работе с аннотацией в функциях, улучшил навыки анализа поставленной задачи.