|  |  |
| --- | --- |
| Picture 1 | **МИНОБРНАУКИ РОССИИ**  **федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01.03 "Интеллектуальные технологии киберфизических систем"

**«Контрольный Проект»**

Студент 1 курса группы 20121

Очного отделения

Лисачев А. Ю.

Проверил:

Поделенюк П. П.

2024

Оглавление

[Цель работы 3](#_Toc170747158)

[Разработка классов и UML диаграмма 3](#_Toc170747159)

[Результаты работы 4](#_Toc170747160)

[Реализация программы с использованием функционального программирования 4](#_Toc170747161)

[Ход работы 4](#_Toc170747162)

[Демонстрация работы программы 5](#_Toc170747163)

[Листинг кода 6](#_Toc170747164)

[Реализация программы с использованием ООП языка Python 9](#_Toc170747165)

[Ход работы 9](#_Toc170747166)

[Демонстрация работы программы 12](#_Toc170747167)

[Листинг кода 14](#_Toc170747168)

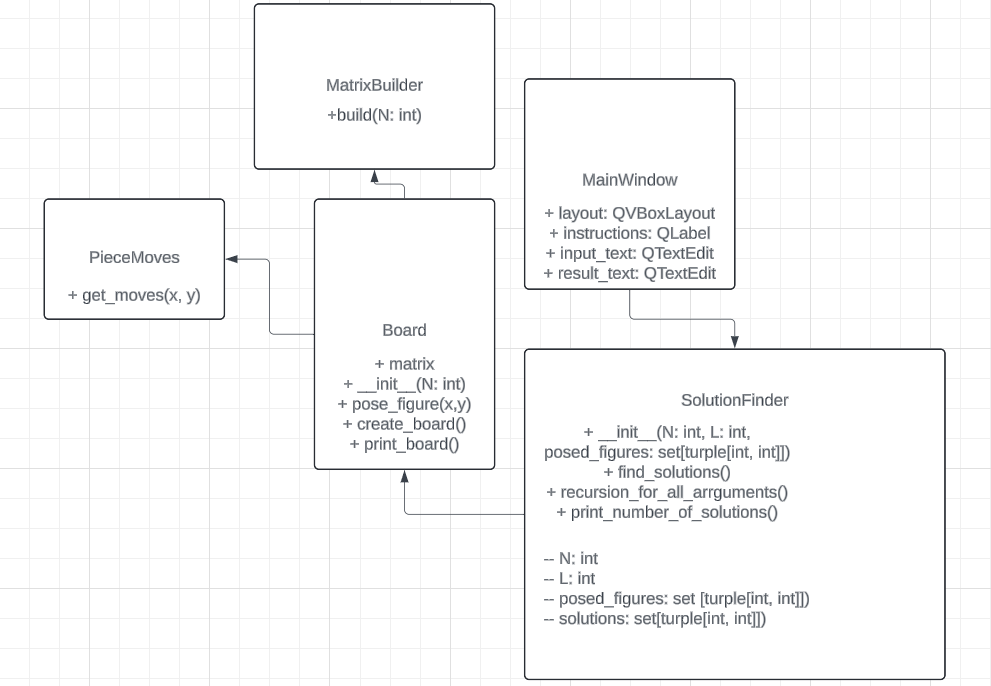
[Заключение 17](#_Toc170747169)

# Цель работы

Целью данной лабораторной работы является разработка и оптимизация алгоритма для размещения дополнительных фигур на шахматной доске таким образом, чтобы никакая фигура не находилась под боем другой. А также создание GUI приложения.

# Разработка классов и UML диаграмма

1. Класс MainWindow использует SolutionFinder для нахождения решений и отображения результатов в GUI
2. Класс SolutionFinder использует Board для работы с доской
3. Класс Board использует MatrixBuilder для создания начальной матрицы и PieceMoves для определения возможных ходов фигуры



+ обозначает публичные методы или атрибуты, которые могут быть доступны из любого места в коде.

- обозначает приватные методы или атрибуты, которые могут быть доступны только внутри самого класса.

# Результаты работы

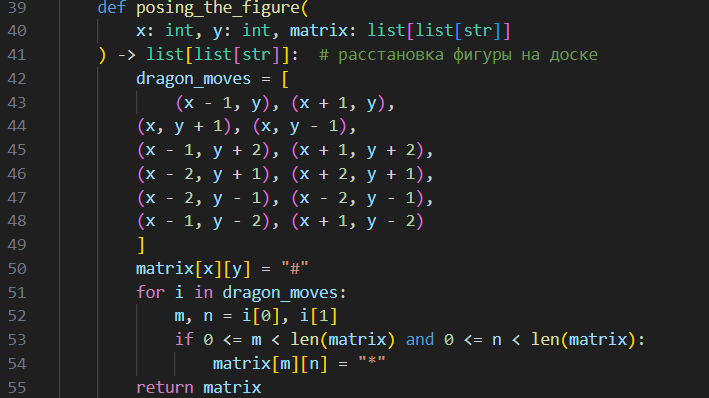
## Реализация программы с использованием функционального программирования

### Ход работы

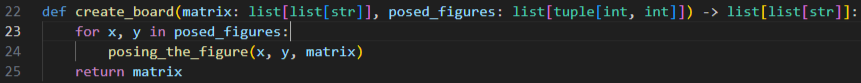
Была написана функция, которая создает доску в виде матрицы:



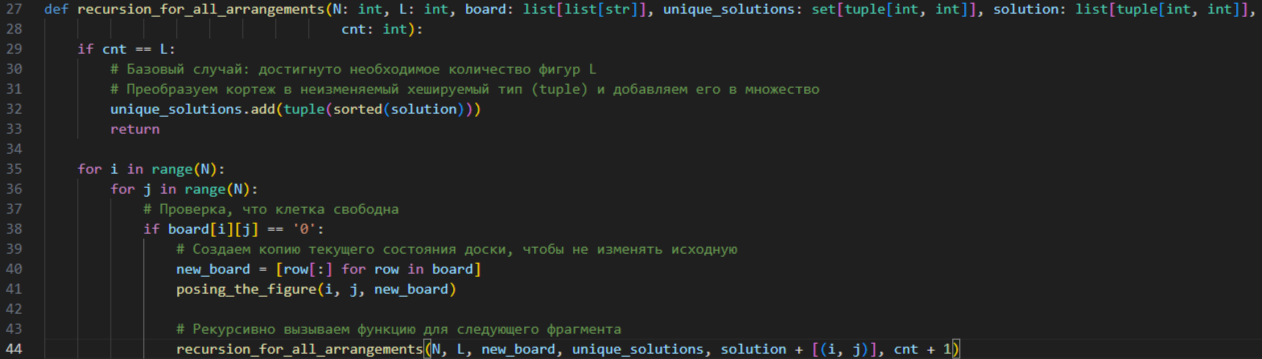
Также была создана функция, которая ставит на доску фигуру (#) и обозначает клетки, которые эта фигура бьет (\*):



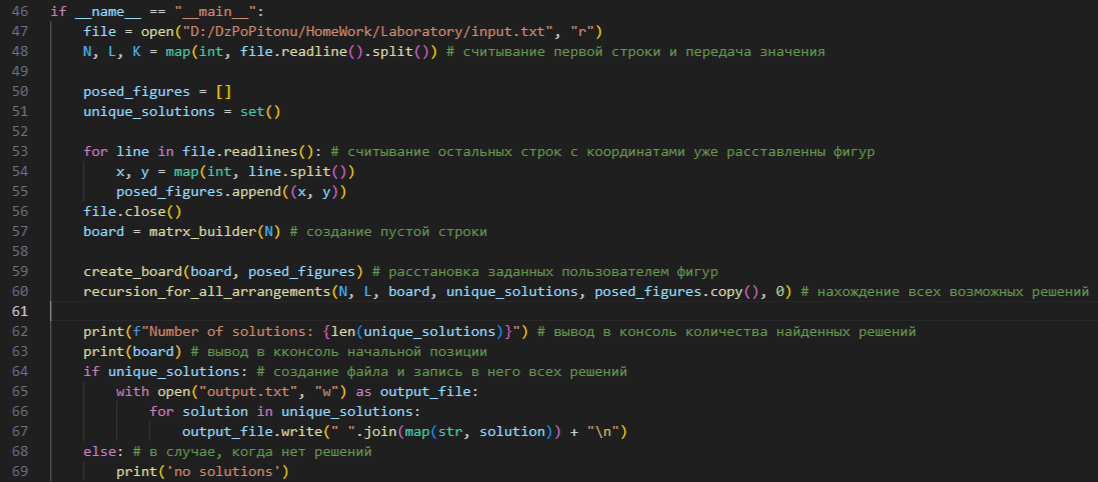
Была создана функция для создания начальной позиции, учитывая расставленные фигуры:



Далее была создана основная функция, которая расставляет фигуры на доске всеми возможными способами и заносит ответ в unique\_solutions, используя при это рекурсию:



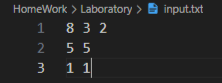
Основная часть кода с считыванием с input.txt, созданием output.txt, использованием вышеуказанных функций и выводом в консоль количества решений и начальной позиции:



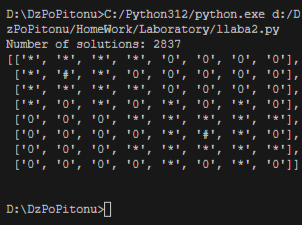
### Демонстрация работы программы

**Когда решения есть**

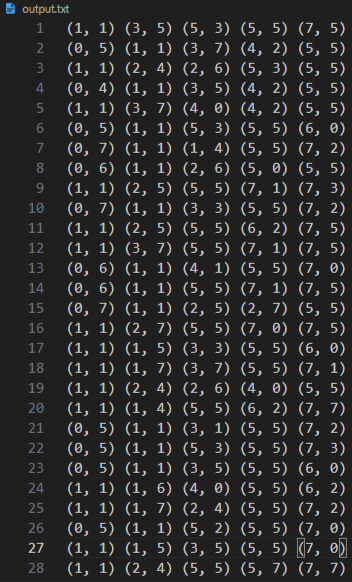
Input.txt



Консоль

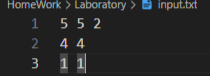


Часть файла Output.txt

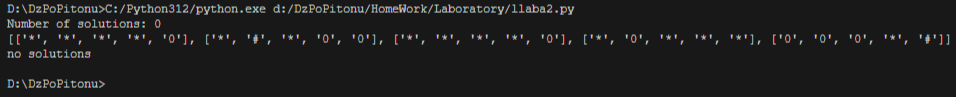


**Когда решений нет**

Input.txt



Консоль



Output.txt не создан

### Листинг кода

def matrx\_builder(N: int) -> list[list[str]]:

    return [['0' for \_ in range(N)] for \_ in range(N)]

def piece\_moves(x, y):

    moves = {

        (x - 1, y), (x + 1, y),

        (x, y + 1), (x, y - 1),

        (x - 1, y + 2), (x + 1, y + 2),

        (x - 2, y + 1), (x + 2, y + 1),

        (x - 2, y - 1), (x - 2, y - 1),

        (x - 1, y - 2), (x + 1, y - 2)

    }

    return moves

def posing\_the\_figure(x: int, y: int, matrix: list[list[str]]) -> list[list[str]]:

    dragon\_moves = [

        (x - 1, y), (x + 1, y),

        (x, y + 1), (x, y - 1),

        (x - 1, y + 2), (x + 1, y + 2),

        (x - 2, y + 1), (x + 2, y + 1),

        (x - 2, y - 1), (x - 2, y - 1),

        (x - 1, y - 2), (x + 1, y - 2)

    ]

    matrix[x][y] = '#'

    for i in dragon\_moves:

        m, n = i[0], i[1]

        if 0 <= m < len(matrix) and 0 <= n < len(matrix):

            matrix[m][n] = '\*'

    return matrix

def create\_board(matrix: list[list[str]], posed\_figures: list[tuple[int, int]]) -> list[list[str]]:

    for x, y in posed\_figures:

        posing\_the\_figure(x, y, matrix)

    return matrix

def print\_board(matrix: list[list[str]]):

    for row in matrix:

        print(" ".join(row))

def recursion\_for\_all\_arrangements(N: int, L: int, solutions: set[tuple[int, int]], solution: set[tuple[int, int]], cnt: int, last\_x: int, last\_y: int):

    if cnt == L:

        unique\_solution = tuple(solution)

        solutions.add(unique\_solution)

        # Вывод первого решения

        if len(solutions) == 1:

            print("First solution:")

            print\_board(create\_board(matrx\_builder(N), unique\_solution))

        return

    # (last\_x, last\_y) - координаты последней поставленной фигуры

    for i in range(last\_x, N):

        if i == last\_x: # если на этой строке уже поставлена фигура, то проход идет начиная с координат последней фигуры

            start\_y = last\_y

        else: # иначе будет проход всей строки с начала

            start\_y = 0

        for j in range(start\_y, N):

            if (i, j) not in solution and not piece\_moves(i, j).intersection(solution):

                solution.add((i, j))

                recursion\_for\_all\_arrangements(N, L, solutions, solution, cnt + 1, i, j)

                solution.remove((i, j))

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

    file = open("C:/sem1/laba2/input.txt", "r")

    N, L, K = map(int, file.readline().split())

    posed\_figures = set()

    solutions = set()

    for line in file.readlines():

        x, y = map(int, line.split())

        posed\_figures.add((x, y))

    file.close()

    recursion\_for\_all\_arrangements(N, L, solutions, posed\_figures, 0, 0, 0)

    print(f"Number of solutions: {len(solutions)}")

    if solutions:

        solutions\_str = [" ".join(map(str, solution)) + "\n" for solution in solutions]

        with open("C:/sem1/laba2/output.txt", "w") as output\_file:

            output\_file.writelines(solutions\_str)

    else:

        print('no solutions')

## Реализация программы с использованием ООП языка Python

### Ход работы

Был написан класс MatrixBuilder, который содержит статические методы для работы с доской.

class MatrixBuilder:

build(N): Создает пустую доску размера N x N, представленную в виде двумерного списка.

    @staticmethod

    def build(N: int) -> list[list[str]]: # создание доски

        return [["0" for \_ in range(N)] for \_ in range(N)]

get\_moves(x, y): Возвращает множество координат, на которые может сходить фигура с координатами (x, y).

@staticmethod

    def get\_moves(x, y) -> set[tuple[int, int]]:

        moves = {

        (x - 1, y), (x + 1, y),

        (x, y + 1), (x, y - 1),

        (x - 1, y + 2), (x + 1, y + 2),

        (x - 2, y + 1), (x + 2, y + 1),

        (x - 2, y - 1), (x - 2, y - 1),

        (x - 1, y - 2), (x + 1, y - 2)

        }

        return moves

create\_board(matrix, posed\_figures): Создает доску matrix с уже размещенными фигурами из pose\_figures.

class Board:

    def \_\_init\_\_(self, N):

        self.matrix = MatrixBuilder.build(N)

    def pose\_figure(self, x: int, y: int):

        dragon\_moves = PieceMoves.get\_moves(x, y)

        self.matrix[x][y] = '#'

        for m, n in dragon\_moves:

            if 0 <= m < len(self.matrix) and 0 <= n < len(self.matrix):

                self.matrix[m][n] = '\*'

    def create\_board(self, posed\_figures: list[tuple[int, int]]):

        for x, y in posed\_figures:

            self.pose\_figure(x, y)

    def print\_board(self):

        for row in self.matrix:

            print(" ".join(row))

Метод recursion\_for\_all\_arrangements класса SolutionFinder используется для поиска всех возможных комбинаций расстановки L фигур на доске размера N x N.

Рекурсивный подход позволяет проверять и генерировать все возможные комбинации, начиная с заданной позиции (x, y) на доске.

def \_recursion\_for\_all\_arrangements(self, N, L, solutions, solution, cnt, last\_x, last\_y):

        if cnt == L:

            unique\_solution = tuple(solution)

            solutions.add(unique\_solution)

            # Вывод первого решения

            if len(solutions) == 1:

                print("First solution:")

                board = Board(N)

                board.create\_board(list(unique\_solution))

                board.print\_board()

            return

        for i in range(last\_x, N):

            start\_y = last\_y if i == last\_x else 0

            for j in range(start\_y, N):

                if (i, j) not in solution and not PieceMoves.get\_moves(i, j).intersection(solution):

                    solution.add((i, j))

                    self.\_recursion\_for\_all\_arrangements(N, L, solutions, solution, cnt + 1, i, j)

                    solution.remove((i, j))

Класс MainWindow отвечает за создание и управление графическим интерфейсом пользователя, обработку вводимых данных, запуск основного алгоритма и отображение результатов. Основные методы этого класса обеспечивают инициализацию, обработку и отображение

class MainWindow(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Board Solutions")

        self.setGeometry(100, 100, 800, 600)

        self.layout = QVBoxLayout()

        self.instructions = QLabel("Размер доски, количество фигур, количество расставленных фигур")

        self.layout.addWidget(self.instructions)

        self.input\_text = QTextEdit()

        self.layout.addWidget(self.input\_text)

        self.result\_text = QTextEdit()

        self.result\_text.setReadOnly(True)

        self.layout.addWidget(self.result\_text)

        self.setLayout(self.layout)

        self.\_run()

\_\_init\_\_(self) - Конструктор класса, который инициализирует все основные элементы GUI и устанавливает начальную структуру интерфейса. Метод создает и настраивает виджеты QLabel, QTextEdit, а так же вызывает метод \_run(), чтобы запустить основную логику приложения

def \_run(self):

        # Пример входных данных

        input\_data = "5 3 2\n1 1\n2 3\n"

        self.input\_text.setPlainText(input\_data)

        self.\_process\_input()

Метод \_run(self) Вызывает метод \_process\_input() для обработки входных данных и запуска алгоритма

def \_process\_input(self):

        data = self.input\_text.toPlainText().strip().split('\n')

        N, L, K = map(int, data[0].split())

        posed\_figures = set(tuple(map(int, line.split())) for line in data[1:])

        solution\_finder = SolutionFinder(N, L, posed\_figures)

        solution\_finder.find\_solutions()

        solution\_finder.print\_number\_of\_solutions()

        result\_str = f"Number of solutions: {len(solution\_finder.solutions)}\n"

        if solution\_finder.solutions:

            for solution in solution\_finder.solutions:

                board = Board(N)

                board.create\_board(list(solution))

                result\_str += "\n".join(" ".join(row) for row in board.matrix) + "\n\n"

        else:

            result\_str += 'No solutions\n'

        self.result\_text.setPlainText(result\_str)

Метод \_process\_input(self) считывает данные из текстового поля input\_text и обрабатывает их. Создает множество posed\_figures, содержащее начальные координаты фигур, заданные пользователем, объект SolutionFinder с полученными значениями N, L и posed\_figures.

Форматирует результаты и отображает их в текстовом поле result\_text.

Запуск приложения.

if \_\_name\_\_ == "\_\_main\_\_": # запуск  приложения

    app = QApplication(sys.argv)

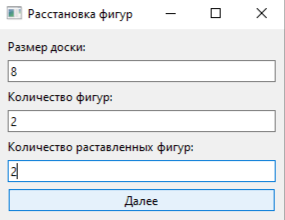
    window = MainWindow()

    window.show()

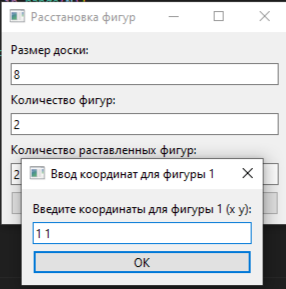
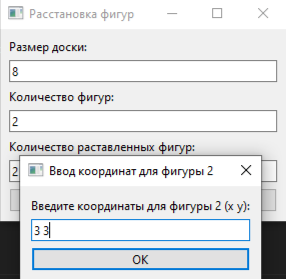
    sys.exit(app.exec())

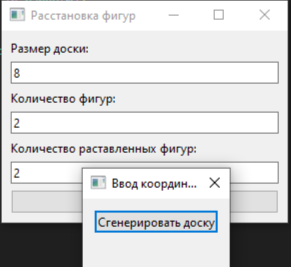
### Демонстрация работы программы

Ввод данных пользователя(размер доски, количество фигур)



Ввод координат уже расставленных фигур



Получение результата



### Листинг кода

from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QTextEdit

from PySide6.QtCore import Qt

import sys

class MatrixBuilder:

    @staticmethod

    def build(N: int) -> list[list[str]]:

        return [['0' for \_ in range(N)] for \_ in range(N)]

class PieceMoves:

    @staticmethod

    def get\_moves(x, y):

        return {

            (x - 1, y), (x + 1, y),

            (x, y + 1), (x, y - 1),

            (x - 1, y + 2), (x + 1, y + 2),

            (x - 2, y + 1), (x + 2, y + 1),

            (x - 2, y - 1), (x - 2, y - 1),

            (x - 1, y - 2), (x + 1, y - 2)

        }

class Board:

    def \_\_init\_\_(self, N):

        self.matrix = MatrixBuilder.build(N)

    def pose\_figure(self, x: int, y: int):

        dragon\_moves = PieceMoves.get\_moves(x, y)

        self.matrix[x][y] = '#'

        for m, n in dragon\_moves:

            if 0 <= m < len(self.matrix) and 0 <= n < len(self.matrix):

                self.matrix[m][n] = '\*'

    def create\_board(self, posed\_figures: list[tuple[int, int]]):

        for x, y in posed\_figures:

            self.pose\_figure(x, y)

    def print\_board(self):

        for row in self.matrix:

            print(" ".join(row))

class SolutionFinder:

    def \_\_init\_\_(self, N, L, posed\_figures):

        self.N = N

        self.L = L

        self.posed\_figures = posed\_figures

        self.solutions = set()

    def find\_solutions(self):

        self.\_recursion\_for\_all\_arrangements(self.N, self.L, self.solutions, set(self.posed\_figures), 0, 0, 0)

    def \_recursion\_for\_all\_arrangements(self, N, L, solutions, solution, cnt, last\_x, last\_y):

        if cnt == L:

            unique\_solution = tuple(solution)

            solutions.add(unique\_solution)

            # Вывод первого решения

            if len(solutions) == 1:

                print("First solution:")

                board = Board(N)

                board.create\_board(list(unique\_solution))

                board.print\_board()

            return

        for i in range(last\_x, N):

            start\_y = last\_y if i == last\_x else 0

            for j in range(start\_y, N):

                if (i, j) not in solution and not PieceMoves.get\_moves(i, j).intersection(solution):

                    solution.add((i, j))

                    self.\_recursion\_for\_all\_arrangements(N, L, solutions, solution, cnt + 1, i, j)

                    solution.remove((i, j))

    def print\_number\_of\_solutions(self):

        print(f"Number of solutions: {len(self.solutions)}")

        if self.solutions:

            solutions\_str = [" ".join(map(str, solution)) + "\n" for solution in self.solutions]

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

                output\_file.writelines(solutions\_str)

        else:

            print('no solutions')

class MainWindow(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Board Solutions")

        self.setGeometry(100, 100, 800, 600)

        self.layout = QVBoxLayout()

        self.instructions = QLabel("Размер доски, количество фигур, количество расставленных фигур")

        self.layout.addWidget(self.instructions)

        self.input\_text = QTextEdit()

        self.layout.addWidget(self.input\_text)

        self.result\_text = QTextEdit()

        self.result\_text.setReadOnly(True)

        self.layout.addWidget(self.result\_text)

        self.setLayout(self.layout)

        self.\_run()

    def \_run(self):

        # Пример входных данных

        input\_data = "5 3 2\n1 1\n2 3\n"

        self.input\_text.setPlainText(input\_data)

        self.\_process\_input()

    def \_process\_input(self):

        data = self.input\_text.toPlainText().strip().split('\n')

        N, L, K = map(int, data[0].split())

        posed\_figures = set(tuple(map(int, line.split())) for line in data[1:])

        solution\_finder = SolutionFinder(N, L, posed\_figures)

        solution\_finder.find\_solutions()

        solution\_finder.print\_number\_of\_solutions()

        result\_str = f"Number of solutions: {len(solution\_finder.solutions)}\n"

        if solution\_finder.solutions:

            for solution in solution\_finder.solutions:

                board = Board(N)

                board.create\_board(list(solution))

                result\_str += "\n".join(" ".join(row) for row in board.matrix) + "\n\n"

        else:

            result\_str += 'No solutions\n'

        self.result\_text.setPlainText(result\_str)

if \_\_name\_\_ == "\_\_main\_\_": # Запуск приложения

    app = QApplication(sys.argv)

    window = MainWindow()

    window.show()

    sys.exit(app.exec())

# Заключение

В ходе выполнения лабораторной работы был разработан и оптимизирован алгоритм размещения дополнительных фигур на шахматной доске. Полученный алгоритм работает эффективно и находит все возможные решения задачи

Таким образом, цель работы достигнута, и разработанный алгоритм может быть использован для эффективного решения аналогичных задач размещения фигур на шахматной доске, а также разработано приложение на основе этого алгоритма.