|  |  |
| --- | --- |
| 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

[Оглавление 2](#_Toc170313153)

[Цель работы 3](#_Toc170313154)

[Разработка классов и UML диаграмма 3](#_Toc170313155)

[Результаты работы 4](#_Toc170313156)

[Реализация программы с использованием функционального программирования 4](#_Toc170313157)

[Ход работы 4](#_Toc170313158)

[Демонстрация работы программы 5](#_Toc170313159)

[Листинг кода 7](#_Toc170313160)

[Реализация программы с использованием ООП языка Python 9](#_Toc170313161)

[Ход работы 9](#_Toc170313162)

[Демонстрация работы программы 17](#_Toc170313163)

[Листинг кода 19](#_Toc170313164)

[Заключение 27](#_Toc170313165)

# 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. Чтение входных данных:

Класс InputDialog отвечает за ввод данных пользователем: размер доски N, количество фигур L, и количество уже расставленных фигур K. Он также получает координаты уже расставленных фигур с помощью класса CoordinateInputDialog. Инициализация доски и размещение известных фигур с помощью класса CoordinateInputDialog

1. Инициализация доски и размещение известных фигур

Класс CoordinateInputDialog предоставляет пользователю окно для ввода координат фигур. Он считывает и возвращает координаты для каждой фигуры.

1. Создание доски и размещение фигур

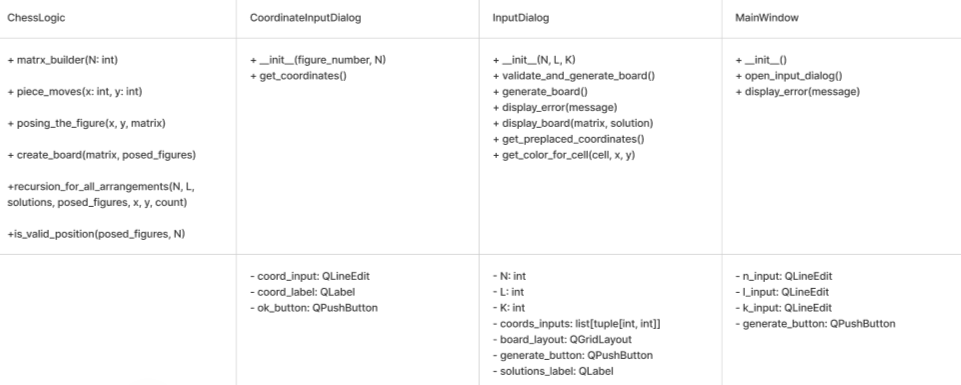
Класс ChessLogic содержит статические методы для работы с логикой шахмат: matrx\_builder для создания пустой доски размера N x N, posing\_the\_figure для размещения фигур на доске, и create\_board для создания доски с расставленными фигурами.

1. Оптимизированный поиск возможных решений

В методе recursion\_for\_all\_arrangements класса ChessLogic рекурсивно проверяются все возможные комбинации размещения L фигур на доске размера N x N.

1. Вывод доски и результатов

Класс MainWindow представляет главное окно приложения, где пользователь вводит данные и может запустить процесс генерации и отображения доски с фигурами. Он также выводит количество найденных решений и отображает доску с помощью методов класса InputDialog и ChessLogic.



+ 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

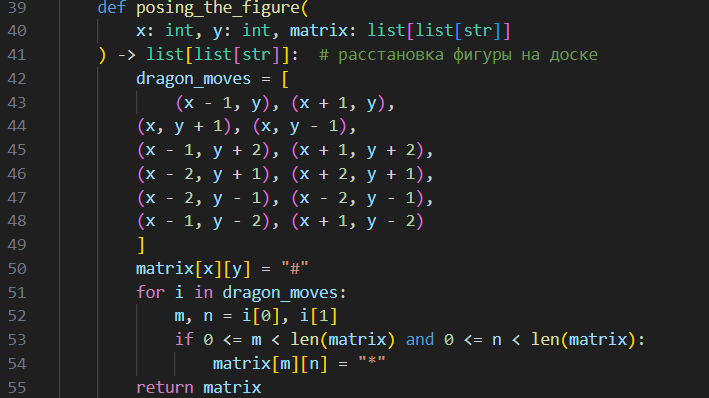
## Реализация программы с использованием функционального программирования

### Ход работы

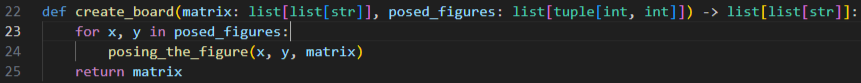
Была написана функция, которая создает доску в виде матрицы:



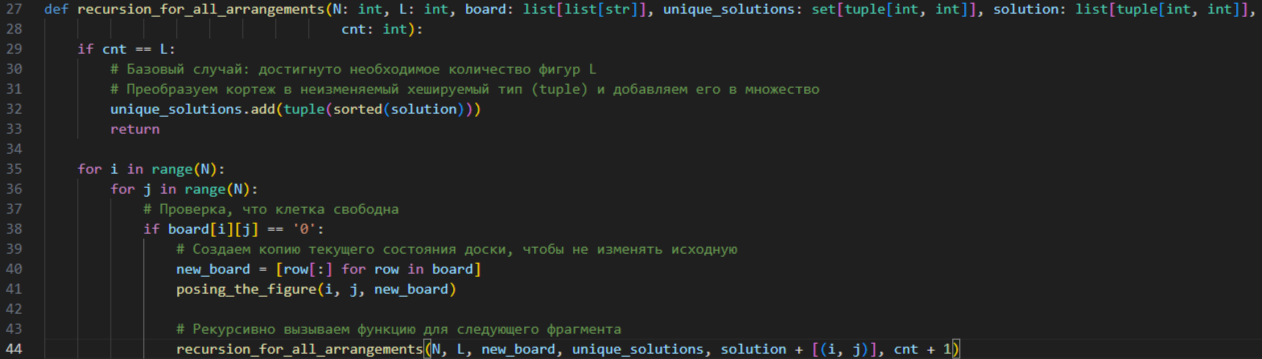
Также была создана функция, которая ставит на доску фигуру (#) и обозначает клетки, которые эта фигура бьет (\*):



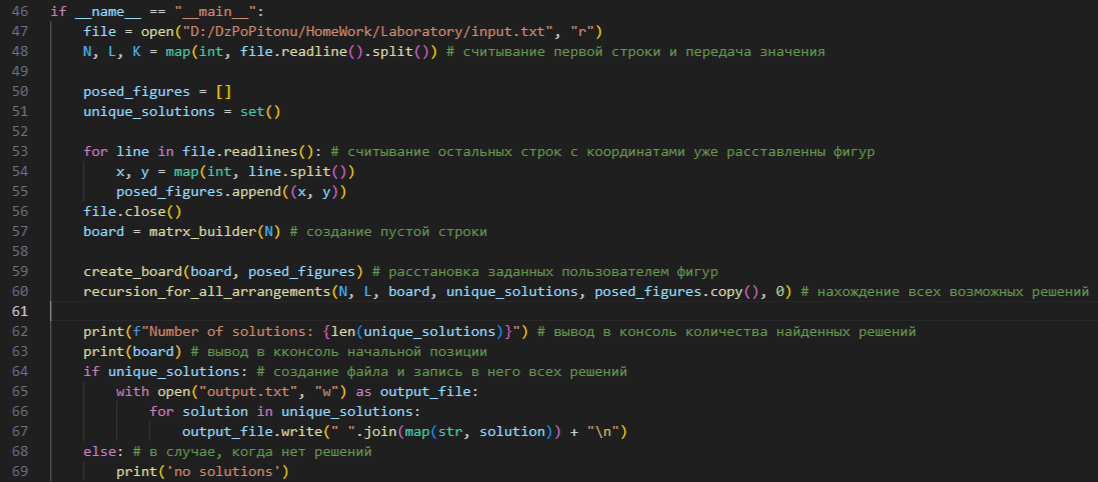
Была создана функция для создания начальной позиции, учитывая расставленные фигуры:



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



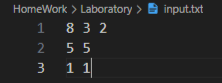
Основная часть кода с считыванием с input.txt, созданием output.txt, использованием вышеуказанных функций и выводом в консоль количества решений и начальной позиции:



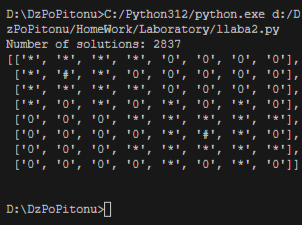
### Демонстрация работы программы

**Когда решения есть**

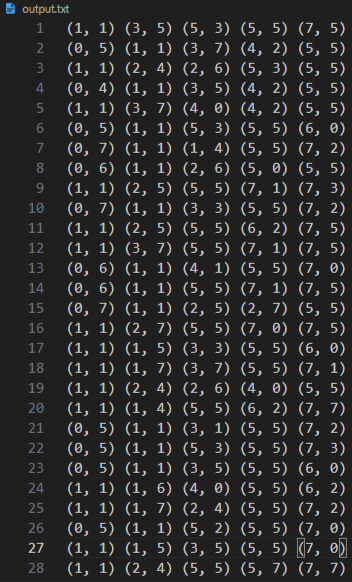
Input.txt



Консоль

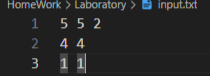


Часть файла Output.txt

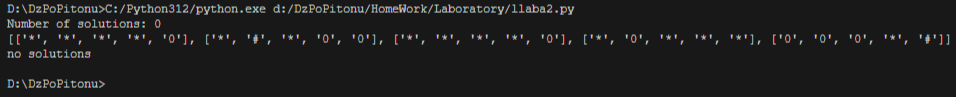


**Когда решений нет**

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 ChessLogic, который содержит статические методы для работы с доской.

class ChessLogic:

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

    @staticmethod

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

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

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

@staticmethod

    def piece\_moves(x: int, y: int) -> set[tuple[int, int]]: # координаты, которые бьет фигура

        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

posing\_the\_figure(x, y, matrix): Расставляет фигуру на доске matrix в клетке (x, y) и отмечает возможные ходы фигуры.

@staticmethod

    def posing\_the\_figure(x: int, y: int, matrix: list[list[str]]) -> list[list[str]]: # расстановка фигуры на доске

        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

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

@staticmethod

    def create\_board(

        matrix: list[list[str]], posed\_figures: list[tuple[int, int]]

    ) -> list[list[str]]: # создание доски с расстановкой фигур

        board = matrix

        for x, y in posed\_figures:

            board = ChessLogic.posing\_the\_figure(x, y, board)

        return board

Метод recursion\_for\_all\_arrangements класса ChessLogic 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.

 @staticmethod

    def recursion\_for\_all\_arrangements(

        N: int,

        L: int,

        solutions: set[frozenset],

        posed\_figures: set[tuple[int, int]],

        x: int,

        y: int,

        count: int,

    ):  # находит все возможные расстановки фигур

        if count == L:

            solutions.add(frozenset(posed\_figures))

            return

        for i in range(x, N):

            for j in range(y if i == x else 0, N):

                if (i, j) not in posed\_figures:

                    new\_posed\_figures = posed\_figures | {(i, j)}

                    ChessLogic.recursion\_for\_all\_arrangements(

                        N, L, solutions, new\_posed\_figures, i, j + 1, count + 1

                    )

is\_valid\_position(posed\_figures, N): Метод проверяет корректность расстановки фигур posed\_figures на доске размера N x N.

Проверка осуществляется на основе того, что фигуры не находятся под ударом друг от друга согласно правилам шахмат.

@staticmethod

    def is\_valid\_position(

        posed\_figures: set[tuple[int, int]], N: int

    ) -> bool:  # проверяет корректность растановок фигур

        for x, y in posed\_figures:

            for move in ChessLogic.piece\_moves(x, y):

                if move in posed\_figures:

                    return False

        return True

Класс CoordinateInputeDialog предоставляет диалоговое окно для ввода координат фигуры.

\_\_init\_\_(self, figure\_number: int, N: int): Инициализирует диалоговое окно с меткой и полем ввода для координат фигуры figure\_number.

get\_coordinates(self) -> tuple[int, int]: Возвращает введенные пользователем координаты в виде кортежа (x, y).

class CoordinateInputDialog(QDialog):

    def \_\_init\_\_(self, figure\_number: int, N: int): # окно с вводом координат фигур

        super().\_\_init\_\_()

        self.setWindowTitle(f"Ввод координат для фигуры {figure\_number}")

        self.layout = QVBoxLayout()

        self.coord\_input = QLineEdit()

        self.coord\_label = QLabel(

            f"Введите координаты для фигуры {figure\_number} (x y):"

        )

        self.layout.addWidget(self.coord\_label)

        self.layout.addWidget(self.coord\_input)

        self.ok\_button = QPushButton("OK")

        self.ok\_button.clicked.connect(self.accept)

        self.layout.addWidget(self.ok\_button)

        self.setLayout(self.layout)

    def get\_coordinates(self) -> tuple[int, int]:  # возвращает введенные координаты

        try:

            x, y = map(int, self.coord\_input.text().split())

            return x, y

        except ValueError:

            return None, None

InputDialog управляет вводом параметров (размер доски N, количество фигур L, и количество известных фигур K) и расстановкой фигур на доске.

\_init\_\_(self, N: int, L: int, K: int): Инициализирует окно для ввода параметров и координат известных фигур.

validate\_and\_generate\_board(self) -> bool: Проверяет корректность введенных параметров и генерирует доску с фигурами.

class InputDialog(QDialog):

    def \_\_init\_\_(self, N: int, L: int, K: int): # расстановка фигур

        super().\_\_init\_\_()

        self.setWindowTitle("Ввод координат")

        self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.layout = QVBoxLayout()

        self.N = N

        self.L = L

        self.K = K

        self.coords\_inputs = []

        for i in range(K):

            coord\_dialog = CoordinateInputDialog(i + 1, N)

            if coord\_dialog.exec() == QDialog.Accepted:

                x, y = coord\_dialog.get\_coordinates()

                if 0 <= x < N and 0 <= y < N:

                    self.coords\_inputs.append((x, y))

        self.generate\_button = QPushButton("Сгенерировать доску")

        self.generate\_button.clicked.connect(self.validate\_and\_generate\_board)

        self.layout.addWidget(self.generate\_button)

        self.board\_layout = QGridLayout()

        self.layout.addLayout(self.board\_layout)

        self.solutions\_label = QLabel("")

        self.layout.addWidget(self.solutions\_label)

        spacer = QSpacerItem(20, 40, QSizePolicy.Minimum, QSizePolicy.Expanding)

        self.layout.addItem(spacer)

        self.setLayout(self.layout)

    def validate\_and\_generate\_board(self): # проверка расстановки фигур и генерация доски

        posed\_figures = set(self.coords\_inputs)

        if not ChessLogic.is\_valid\_position(posed\_figures, self.N):

            self.display\_error("Некорректное расположение фигур")

            return False

        self.generate\_board()

        return True

generate\_board(self): Генерирует доску и находит все возможные решения для расстановки фигур.

def generate\_board(self): # генерация доски с решением

        posed\_figures = set(self.coords\_inputs)

        solutions = set()

        ChessLogic.recursion\_for\_all\_arrangements(

            self.N, self.L, solutions, posed\_figures, 0, 0, 0

        )

        self.display\_board(

            ChessLogic.matrx\_builder(self.N),

            next(iter(solutions)) if solutions else None,

        )

        self.solutions\_label.setText(f"Количество решений: {len(solutions)}")

display\_error(self, message): Выводит диалоговое окно с сообщением об ошибке.

display\_board(self, matrix, solution): Отображает доску с расстановкой фигур и ее решениями.

 def display\_error(self, message: str): # вызывает окно с ошибкой

        error\_dialog = QDialog(self)

        error\_dialog.setWindowTitle("Ошибка")

        error\_layout = QVBoxLayout()

        error\_label = QLabel(message)

        error\_layout.addWidget(error\_label)

        error\_dialog.setLayout(error\_layout)

        error\_dialog.exec()

    def display\_board(self, matrix: list[list[str]], solution: set[tuple[int, int]]):  # отображает доску с расстановкой фигур

        for i in range(self.board\_layout.count()):

            self.board\_layout.itemAt(i).widget().deleteLater()

        if solution:

            board = ChessLogic.create\_board(matrix, solution)

            for i, row in enumerate(board):

                for j, cell in enumerate(row):

                    label = QLabel()

                    color = self.get\_color\_for\_cell(cell, i, j)

                    label.setStyleSheet(

                        f"background-color: {color}; border: 1px solid black; min-width: 15px; min-height: 15px;"

                    )

                    label.setAlignment(Qt.AlignCenter)

                    label.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

                    self.board\_layout.addWidget(label, i, j)

        else:

            self.display\_error("Решений не найдено")

get\_preplaced\_coordinates(self) возвращает координаты расставленных фигур.

def get\_preplaced\_coordinates(

        self,

    ) -> set[tuple[int, int]]:  # координаты расставленных изначально фигур

        return set(self.coords\_inputs)

get\_color\_for\_cell(self, cell) отвечает за окрашивание клеток.

def get\_color\_for\_cell(self, cell: str, x: int, y: int) -> str: # расскрашивание клеток

        if cell == "#":

            return "blue"

        elif cell == "\*":

            return "red"

        else:

            return "white"

Class MainWindow - главное окно приложения, отображает интерфейс для ввода параметров и управляет их обработкой.

class MainWindow(QMainWindow):

    def \_\_init\_\_(self): # главное окно приложения

        super().\_\_init\_\_()

        self.setWindowTitle("Расстановка фигур")

        self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.central\_widget = QWidget()

        self.layout = QVBoxLayout()

        self.n\_input = QLineEdit()

        self.l\_input = QLineEdit()

        self.k\_input = QLineEdit()

        self.n\_label = QLabel("Размер доски:")

        self.l\_label = QLabel("Количество фигур:")

        self.k\_label = QLabel("Количество раставленных фигур:")

        self.layout.addWidget(self.n\_label)

        self.layout.addWidget(self.n\_input)

        self.layout.addWidget(self.l\_label)

        self.layout.addWidget(self.l\_input)

        self.layout.addWidget(self.k\_label)

        self.layout.addWidget(self.k\_input)

        self.generate\_button = QPushButton("Далее")

        self.generate\_button.clicked.connect(self.open\_input\_dialog)

        self.layout.addWidget(self.generate\_button)

        self.central\_widget.setLayout(self.layout)

        self.setCentralWidget(self.central\_widget)

    def open\_input\_dialog(self): # ввод координат, проверка и запуск

        try:

            N = int(self.n\_input.text())

            L = int(self.l\_input.text())

            K = int(self.k\_input.text())

            if N <= 0 or L <= 0 or K < 0 or K >= N\*\*2 or L + K > N\*\*2:

                raise ValueError

            dialog = InputDialog(N, L, K)

            dialog.exec()

        except ValueError:

            self.display\_error(

                "Некорректный ввод.

            )

    def display\_error(self, message: str): # вывод ошибки

        error\_dialog = QDialog(self)

        error\_dialog.setWindowTitle("Ошибка")

        error\_layout = QVBoxLayout()

        error\_label = QLabel(message)

        error\_layout.addWidget(error\_label)

        error\_dialog.setLayout(error\_layout)

        error\_dialog.exec()

Запуск приложения.

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

    app = QApplication(sys.argv)

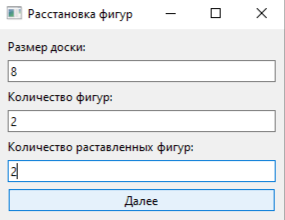
    window = MainWindow()

    window.show()

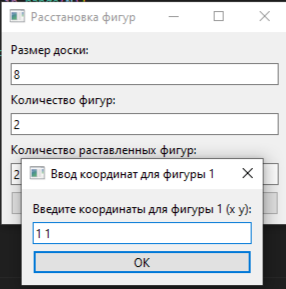
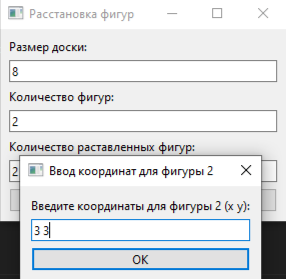
    sys.exit(app.exec())

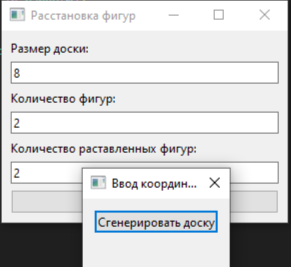
### Демонстрация работы программы

Ввод данных пользователя(размер доски, количество фигур)



Ввод координат уже расставленных фигур



Получение результата



### Листинг кода

import sys

from PySide6.QtWidgets import (

    QApplication,

    QMainWindow,

    QWidget,

    QVBoxLayout,

    QPushButton,

    QLineEdit,

    QLabel,

    QGridLayout,

    QDialog,

    QSizePolicy,

    QSpacerItem,

    QProgressDialog,

)

from PySide6.QtCore import Qt

class ChessLogic:

    @staticmethod

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

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

    @staticmethod

    def piece\_moves(x: int, y: int) -> set[tuple[int, int]]: # координаты, которые бьет фигура

        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

    @staticmethod

    def posing\_the\_figure(x: int, y: int, matrix: list[list[str]]) -> list[list[str]]: # расстановка фигуры на доске

        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

    @staticmethod

    def create\_board(

        matrix: list[list[str]], posed\_figures: list[tuple[int, int]]

    ) -> list[list[str]]: # создание доски с расстановкой фигур

        board = matrix

        for x, y in posed\_figures:

            board = ChessLogic.posing\_the\_figure(x, y, board)

        return board

    @staticmethod

    def recursion\_for\_all\_arrangements(

        N: int,

        L: int,

        solutions: set[frozenset],

        posed\_figures: set[tuple[int, int]],

        x: int,

        y: int,

        count: int,

    ):  # находит все возможные расстановки фигур

        if count == L:

            solutions.add(frozenset(posed\_figures))

            return

        for i in range(x, N):

            for j in range(y if i == x else 0, N):

                if (i, j) not in posed\_figures:

                    new\_posed\_figures = posed\_figures | {(i, j)}

                    ChessLogic.recursion\_for\_all\_arrangements(

                        N, L, solutions, new\_posed\_figures, i, j + 1, count + 1

                    )

    @staticmethod

    def is\_valid\_position(

        posed\_figures: set[tuple[int, int]], N: int

    ) -> bool:  # проверяет корректность растановок фигур

        for x, y in posed\_figures:

            for move in ChessLogic.piece\_moves(x, y):

                if move in posed\_figures:

                    return False

        return True

class CoordinateInputDialog(QDialog):

    def \_\_init\_\_(self, figure\_number: int, N: int): # окно с вводом координат фигур

        super().\_\_init\_\_()

        self.setWindowTitle(f"Ввод координат для фигуры {figure\_number}")

        self.layout = QVBoxLayout()

        self.coord\_input = QLineEdit()

        self.coord\_label = QLabel(

            f"Введите координаты для фигуры {figure\_number} (x y):"

        )

        self.layout.addWidget(self.coord\_label)

        self.layout.addWidget(self.coord\_input)

        self.ok\_button = QPushButton("OK")

        self.ok\_button.clicked.connect(self.accept)

        self.layout.addWidget(self.ok\_button)

        self.setLayout(self.layout)

    def get\_coordinates(self) -> tuple[int, int]:  # возвращает введенные координаты

        try:

            x, y = map(int, self.coord\_input.text().split())

            return x, y

        except ValueError:

            return None, None

class InputDialog(QDialog):

    def \_\_init\_\_(self, N: int, L: int, K: int): # расстановка фигур

        super().\_\_init\_\_()

        self.setWindowTitle("Ввод координат")

        self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.layout = QVBoxLayout()

        self.N = N

        self.L = L

        self.K = K

        self.coords\_inputs = []

        for i in range(K):

            coord\_dialog = CoordinateInputDialog(i + 1, N)

            if coord\_dialog.exec() == QDialog.Accepted:

                x, y = coord\_dialog.get\_coordinates()

                if 0 <= x < N and 0 <= y < N:

                    self.coords\_inputs.append((x, y))

        self.generate\_button = QPushButton("Сгенерировать доску")

        self.generate\_button.clicked.connect(self.validate\_and\_generate\_board)

        self.layout.addWidget(self.generate\_button)

        self.board\_layout = QGridLayout()

        self.layout.addLayout(self.board\_layout)

        self.solutions\_label = QLabel("")

        self.layout.addWidget(self.solutions\_label)

        spacer = QSpacerItem(20, 40, QSizePolicy.Minimum, QSizePolicy.Expanding)

        self.layout.addItem(spacer)

        self.setLayout(self.layout)

    def validate\_and\_generate\_board(self): # проверка расстановки фигур и генерация доски

        posed\_figures = set(self.coords\_inputs)

        if not ChessLogic.is\_valid\_position(posed\_figures, self.N):

            self.display\_error("Некорректное расположение фигур")

            return False

        self.generate\_board()

        return True

    def generate\_board(self): # генерация доски с решением

        posed\_figures = set(self.coords\_inputs)

        solutions = set()

        ChessLogic.recursion\_for\_all\_arrangements(

            self.N, self.L, solutions, posed\_figures, 0, 0, 0

        )

        self.display\_board(

            ChessLogic.matrx\_builder(self.N),

            next(iter(solutions)) if solutions else None,

        )

        self.solutions\_label.setText(f"Количество решений: {len(solutions)}")

    def display\_error(self, message: str): # вызывает окно с ошибкой

        error\_dialog = QDialog(self)

        error\_dialog.setWindowTitle("Ошибка")

        error\_layout = QVBoxLayout()

        error\_label = QLabel(message)

        error\_layout.addWidget(error\_label)

        error\_dialog.setLayout(error\_layout)

        error\_dialog.exec()

    def display\_board(self, matrix: list[list[str]], solution: set[tuple[int, int]]):  # отображает доску с расстановкой фигур

        for i in range(self.board\_layout.count()):

            self.board\_layout.itemAt(i).widget().deleteLater()

        if solution:

            board = ChessLogic.create\_board(matrix, solution)

            for i, row in enumerate(board):

                for j, cell in enumerate(row):

                    label = QLabel()

                    color = self.get\_color\_for\_cell(cell, i, j)

                    label.setStyleSheet(

                        f"background-color: {color}; border: 1px solid black; min-width: 15px; min-height: 15px;"

                    )

                    label.setAlignment(Qt.AlignCenter)

                    label.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

                    self.board\_layout.addWidget(label, i, j)

        else:

            self.display\_error("Решений не найдено")

    def get\_preplaced\_coordinates(

        self,

    ) -> set[tuple[int, int]]:  # координаты расставленных изначально фигур

        return set(self.coords\_inputs)

    def get\_color\_for\_cell(self, cell: str, x: int, y: int) -> str: # расскрашивание клеток

        if cell == "#":

            return "blue"

        elif cell == "\*":

            return "red"

        else:

            return "white"

class MainWindow(QMainWindow):

    def \_\_init\_\_(self): # главное окно приложения

        super().\_\_init\_\_()

        self.setWindowTitle("Расстановка фигур")

        self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.central\_widget = QWidget()

        self.layout = QVBoxLayout()

        self.n\_input = QLineEdit()

        self.l\_input = QLineEdit()

        self.k\_input = QLineEdit()

        self.n\_label = QLabel("Размер доски:")

        self.l\_label = QLabel("Количество фигур:")

        self.k\_label = QLabel("Количество раставленных фигур:")

        self.layout.addWidget(self.n\_label)

        self.layout.addWidget(self.n\_input)

        self.layout.addWidget(self.l\_label)

        self.layout.addWidget(self.l\_input)

        self.layout.addWidget(self.k\_label)

        self.layout.addWidget(self.k\_input)

        self.generate\_button = QPushButton("Далее")

        self.generate\_button.clicked.connect(self.open\_input\_dialog)

        self.layout.addWidget(self.generate\_button)

        self.central\_widget.setLayout(self.layout)

        self.setCentralWidget(self.central\_widget)

    def open\_input\_dialog(self): # ввод координат, проверка и запуск

        try:

            N = int(self.n\_input.text())

            L = int(self.l\_input.text())

            K = int(self.k\_input.text())

            if N <= 0 or L <= 0 or K < 0 or K >= N\*\*2 or L + K > N\*\*2:

                raise ValueError

            dialog = InputDialog(N, L, K)

            dialog.exec()

        except ValueError:

            self.display\_error(

                "Некорректный ввод"

            )

    def display\_error(self, message: str): # вывод ошибки

        error\_dialog = QDialog(self)

        error\_dialog.setWindowTitle("Ошибка")

        error\_layout = QVBoxLayout()

        error\_label = QLabel(message)

        error\_layout.addWidget(error\_label)

        error\_dialog.setLayout(error\_layout)

        error\_dialog.exec()

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.

Были проведены тесты с различными входными данными, включая доски размером до 20х20. Программа успешно справляется с поставленной задачей, обеспечивая оптимальное распределение фигур и выводя доску. Количество найденных решений выводится под доской.

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.