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 №2

По дисциплине «Программирование»

Выполнил: Зорин П. С.

Проверил:

Санкт-Петербург

2024

**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. Работа с пакетами Python
5. Создание GUI приложения с использованием PySide6.

**Формулировка задания**

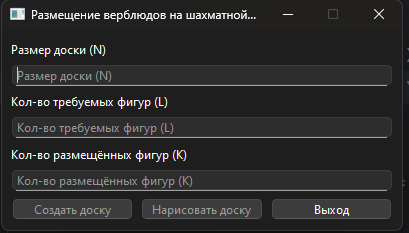
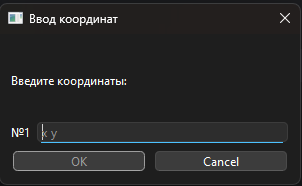
Дана квадратная шахматная доска размером N x N. На доске уже размещено K фигур. Фигуры размещены так, что находятся не под боем друг друга. Необходимо расставить на доске еще L фигур так, чтобы никакая из фигур на доске не находилась под боем любой другой фигуры. Требуется найти одно решение для визуализации и все возможные решения для вывода в файл. Если решение не найдено, то необходимо вывести соответствующее сообщение.

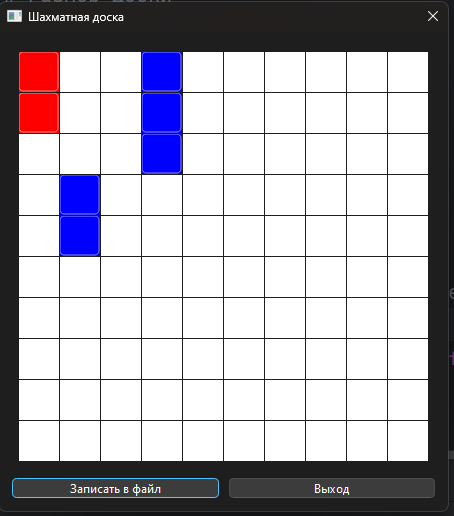
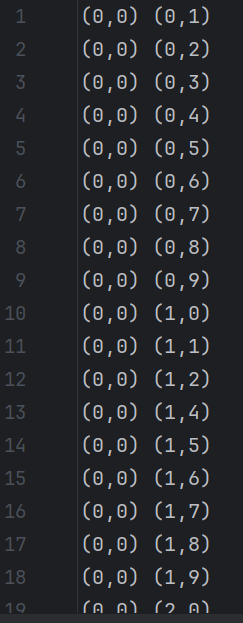
1. Необходимо создать UML диаграмму взаимодействия классов. При необходимости можно делать несколько UML диаграмм. Для диаграммы создается табличка взаимодействия (Имя класса :: Имя базового класса :: Описание). Также для каждого класса создается таблица методов и атрибутов (Методы и атрибуты :: Описание ). 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:

* 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.
* 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-
* Порядок, в котором фигуры перечислены в решении, не имеет значения. Если не было найдено ни одного решения, в файл необходимо записать no solutions.
* Выходные данные на консоль — это доска N\*N, где фигура обозначается #, ее ходы обозначаются \*, а пустые клетки обозначаются 0.

1. Необходимо составить программу с использованием объектно ориентированного программирования (ООП) на языке Python. Каждый класс должен чётко обозначать своё предназначение. Также необходимо разработать интерфейс с использованием пакета PySide6.

**Результаты работы**

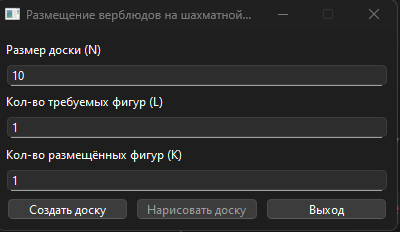
 

Файл output.txt, в котором записаны все уникальные решения.

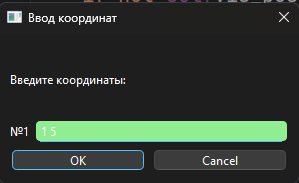
**Ход работы**

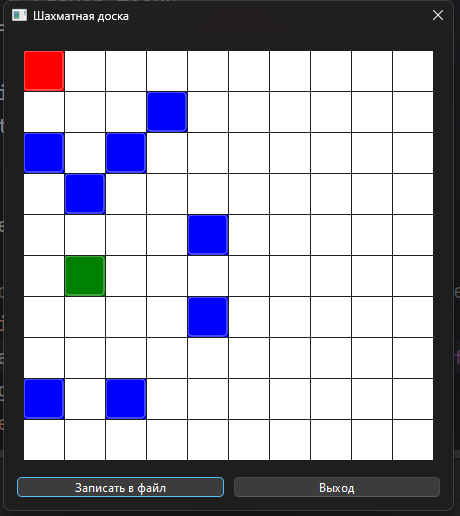
Программа предназначена для размещения определенного числа верблюдов на шахматной доске размером N x N с учетом ограничений. Программа запускает приложение, которое принимает входные данные: N – размер шахматной доски, L – количество фигур, которые надо разместить, K – фигуры, которые уже стоят на доске. Далее открывает диалоговое окно для ввода координат. После рисуется шахматная доска с одним из вариантов расположения верблюдов на ней. Далее выводится количество уникальных решений и все эти решения записываются в текстовый файл output.txt.

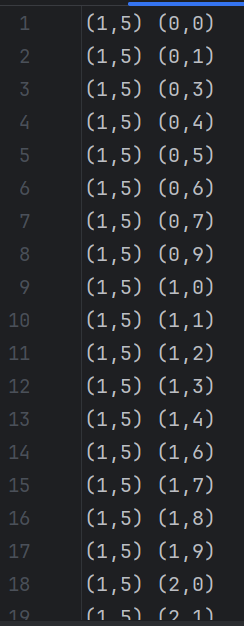
**Демонстрация работы программы**

Ввод данных:  


Ввод координат:



Отрисовка шахматной доски:  


Ввод уникальных решений в текстовый файл:  


**Листинг кода**

import sys  
  
from PySide6.QtCore import QThread, Signal  
from PySide6.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,  
 QHBoxLayout, QLineEdit, QPushButton, QLabel,  
 QDialog, QGridLayout, QMessageBox, QFrame)  
  
  
class Position: # Позиции на доске  
  
 def \_\_init\_\_(self, x, y):  
 self.x = x # Координата X (столбец)  
 self.y = y # Координата Y (Строка)  
  
 def \_\_hash\_\_(self):  
 return hash((self.x, self.y))  
  
 def \_\_eq\_\_(self, other): # Проверка равенства 2ух позиций  
 return isinstance(other, Position) and self.x == other.x and self.y == other.y  
  
  
class CamelPiece: # Верблюд  
 def \_\_init\_\_(self, position):  
 self.position = position # Позиция фигуры на доске  
  
 def get\_attack\_positions(self, board\_size): # Атаковвнные клетки  
 attacks = set()  
 x, y = self.position.x, self.position.y  
  
 # Возможные ходы верблюда  
 moves = [  
 (3, 1), (3, -1), (-3, 1), (-3, -1),  
 (1, 3), (1, -3), (-1, 3), (-1, -3)  
 ]  
  
 for dx, dy in moves:  
 new\_x, new\_y = x + dx, y + dy  
 if 0 <= new\_x < board\_size and 0 <= new\_y < board\_size:  
 attacks.add(Position(new\_x, new\_y))  
  
 return attacks  
  
 def can\_attack(self, other\_position, board\_size): # Проверка, может ли верблюд атаковать позицию  
 return other\_position in self.get\_attack\_positions(board\_size)  
  
  
class ChessBoard: # Шахматная доска  
  
 def \_\_init\_\_(self, size):  
 self.size = size # Размер доски  
 self.placed\_pieces = [] # Список установленных фигут  
  
 def add\_piece(self, position): # Добавление фигур  
 if not self.is\_position\_valid(position):  
 return False  
  
 new\_piece = CamelPiece(position)  
  
 # Проверка на то, может ли новая фигура атаковать уже существующие фигуры  
 for existing\_piece in self.placed\_pieces:  
 if (new\_piece.can\_attack(existing\_piece.position, self.size) or  
 existing\_piece.can\_attack(new\_piece.position, self.size)):  
 return False  
  
 self.placed\_pieces.append(new\_piece)  
 return True  
  
 def is\_position\_valid(self, position): # Проверка не выходит ли фигура за пределы доски  
 return (0 <= position.x < self.size and  
 0 <= position.y < self.size and  
 not self.is\_position\_occupied(position))  
  
 def is\_position\_occupied(self, position): # Проверка не занята ли клетка  
 return any(piece.position == position for piece in self.placed\_pieces)  
  
 def get\_all\_attacked\_positions(self): # Возвращает клетки, которые атакуются  
 attacked = set()  
 for piece in self.placed\_pieces:  
 attacked.update(piece.get\_attack\_positions(self.size))  
 return attacked  
  
 def get\_piece\_positions(self): # Возвращает позиции всех фигур на доске  
 return [piece.position for piece in self.placed\_pieces]  
  
 def clear(self): # Очищает доску  
 self.placed\_pieces.clear()  
  
  
class CamelSolver: # Решение задачи  
 def \_\_init\_\_(self, board\_size, existing\_positions):  
 self.board\_size = board\_size # Размер доски  
 self.existing\_positions = existing\_positions # Существующие позиции  
 self.solutions = [] # Решения  
  
 def solve(self, additional\_pieces): # Поиск решений  
 # Создаем доску с уже размещенными фигурами  
 board = ChessBoard(self.board\_size)  
 for pos in self.existing\_positions:  
 board.add\_piece(pos)  
  
 # Свободные клетки  
 free\_positions = []  
 attacked\_positions = board.get\_all\_attacked\_positions()  
  
 for x in range(self.board\_size):  
 for y in range(self.board\_size):  
 pos = Position(x, y)  
 if (not board.is\_position\_occupied(pos) and  
 pos not in attacked\_positions):  
 free\_positions.append(pos)  
  
 # Находим все возможные комбинации для размещения дополнительных фигур  
 self.solutions = []  
 if additional\_pieces == 0:  
 self.solutions.append(self.existing\_positions.copy())  
 else:  
 self.\_find\_solutions(board.placed\_pieces.copy(), free\_positions, additional\_pieces)  
  
 return self.solutions  
  
 def \_find\_solutions(self, current\_pieces, available\_positions, remaining): # рекурсивно находим решения  
 if remaining == 0:  
 all\_positions = [piece.position for piece in current\_pieces]  
 self.solutions.append(all\_positions)  
 return  
  
 if len(available\_positions) < remaining:  
 return # Недостаточно клеток  
  
 # Возможные клетки для следующей фигуры  
 for i in range(len(available\_positions) - remaining + 1):  
 pos = available\_positions[i]  
  
 new\_piece = CamelPiece(pos)  
  
 # Проверка можно ли разместить фигуру  
 can\_place = True  
 for existing\_piece in current\_pieces:  
 if (new\_piece.can\_attack(existing\_piece.position, self.board\_size) or  
 existing\_piece.can\_attack(new\_piece.position, self.board\_size)):  
 can\_place = False  
 break  
  
 if can\_place: # Размещение фигуры  
 new\_pieces = current\_pieces + [new\_piece]  
  
 # Обновляем список доступных клеток  
 new\_available = []  
 attacked\_by\_new = new\_piece.get\_attack\_positions(self.board\_size)  
  
 for j in range(i + 1, len(available\_positions)):  
 candidate\_pos = available\_positions[j]  
 if candidate\_pos not in attacked\_by\_new:  
 # Проверка, что фигура не атакует новую фигуру  
 candidate\_piece = CamelPiece(candidate\_pos)  
 if not candidate\_piece.can\_attack(pos, self.board\_size):  
 new\_available.append(candidate\_pos)  
  
 self.\_find\_solutions(new\_pieces, new\_available, remaining - 1)  
  
  
class SolverWorker(QThread): # Решение в отдельном потоке  
  
 solution\_found = Signal(list) # Найденные решения  
  
 def \_\_init\_\_(self, board\_size, existing\_positions, additional\_pieces):  
 super().\_\_init\_\_()  
 self.board\_size = board\_size  
 self.existing\_positions = existing\_positions  
 self.additional\_pieces = additional\_pieces  
  
 def run(self): # Решение задачи в отдельном потоке  
 solver = CamelSolver(self.board\_size, self.existing\_positions)  
 solutions = solver.solve(self.additional\_pieces)  
 self.solution\_found.emit(solutions)  
  
  
class CoordinateInputDialog(QDialog): # Ввод координат  
  
 def \_\_init\_\_(self, parent, num\_pieces, board\_size):  
 super().\_\_init\_\_(parent)  
 self.num\_pieces = num\_pieces # Кол-во вводимых фигур  
 self.board\_size = board\_size # Размер доски  
 self.coordinate\_inputs = [] # Поля ввода координат  
 self.positions = []  
  
 self.setWindowTitle("Ввод координат")  
 self.setModal(True)  
 self.setFixedSize(300, 150)  
  
 self.setup\_ui()  
 self.update\_ok\_button\_state()  
  
 def setup\_ui(self): # Интерфейс  
 layout = QVBoxLayout()  
  
 # Заголовок  
 title\_label = QLabel("Введите координаты:")  
 layout.addWidget(title\_label)  
  
 # Поля ввода координат  
 for i in range(self.num\_pieces):  
 coord\_layout = QHBoxLayout()  
 label = QLabel(f"№{i + 1}")  
 coord\_input = QLineEdit()  
 coord\_input.setPlaceholderText("x y")  
 coord\_input.textChanged.connect(self.validate\_coordinates)  
  
 coord\_layout.addWidget(label)  
 coord\_layout.addWidget(coord\_input)  
 layout.addLayout(coord\_layout)  
  
 self.coordinate\_inputs.append(coord\_input)  
  
 # Кнопки  
 button\_layout = QHBoxLayout()  
 self.ok\_button = QPushButton("OK")  
 self.cancel\_button = QPushButton("Cancel")  
  
 self.ok\_button.clicked.connect(self.accept\_coordinates)  
 self.cancel\_button.clicked.connect(self.reject)  
  
 button\_layout.addWidget(self.ok\_button)  
 button\_layout.addWidget(self.cancel\_button)  
 layout.addLayout(button\_layout)  
  
 self.setLayout(layout)  
  
 def validate\_coordinates(self): # Проверка введенных координат с подсветкой ошибок  
 positions = []  
 all\_valid = True  
  
 # Проверяем все поля ввода  
 for i, input\_field in enumerate(self.coordinate\_inputs):  
 text = input\_field.text().strip()  
  
 if not text:  
 input\_field.setStyleSheet("")  
 all\_valid = False  
 continue  
  
 try:  
 parts = text.split()  
 if len(parts) != 2:  
 raise ValueError("Неверные координаты")  
  
 x, y = int(parts[0]), int(parts[1])  
  
 if not (0 <= x < self.board\_size and 0 <= y < self.board\_size):  
 raise ValueError(f"Координаты должны быть от 0 до {self.board\_size - 1}")  
  
 positions.append(Position(x, y))  
 # Правильный ввод - зеленая подсветка  
 input\_field.setStyleSheet("background-color: lightgreen;")  
  
 except ValueError:  
 # Неправильный ввод - красная подсветка  
 input\_field.setStyleSheet("background-color: lightcoral;")  
 all\_valid = False  
  
 # Допольнительная проверка для заполнения полей  
 if all\_valid and len(positions) == self.num\_pieces:  
 # Проверка уникальность позиций  
 if len(set(positions)) != len(positions):  
 # Если есть дублирующиеся позиции, подсвечиваем все поля красным  
 for input\_field in self.coordinate\_inputs:  
 if input\_field.text().strip():  
 input\_field.setStyleSheet("background-color: lightcoral;")  
 all\_valid = False  
 else:  
 # Проверяем, что фигуры не атакуют друг друга  
 board = ChessBoard(self.board\_size)  
 for pos in positions:  
 if not board.add\_piece(pos):  
 # Если фигуры атакуют друг друга, подсвечиваем все поля красным  
 for input\_field in self.coordinate\_inputs:  
 if input\_field.text().strip():  
 input\_field.setStyleSheet("background-color: lightcoral;")  
 all\_valid = False  
 break  
  
 if all\_valid and len(positions) == self.num\_pieces:  
 self.positions = positions  
 else:  
 self.positions = []  
  
 self.update\_ok\_button\_state() # обновление кнопки OK  
  
 def update\_ok\_button\_state(self): # обновление кнопки OK  
 all\_filled = all(input\_field.text().strip() for input\_field in self.coordinate\_inputs)  
 valid\_positions = len(self.positions) == self.num\_pieces  
 self.ok\_button.setEnabled(all\_filled and valid\_positions)  
  
 def accept\_coordinates(self): # Принимает координаты при OK  
 if self.positions:  
 self.accept()  
  
 def get\_positions(self): # Возвращает позиции  
 return self.positions  
  
  
class BoardDisplayWindow(QDialog): # Отображение шахматной доски  
  
 def \_\_init\_\_(self, parent, board\_size, user\_positions, solution\_positions, all\_solutions):  
 super().\_\_init\_\_(parent)  
 self.board\_size = board\_size  
 self.user\_positions = user\_positions  
 self.solution\_positions = solution\_positions  
 self.all\_solutions = all\_solutions  
 self.setWindowTitle("Шахматная доска")  
 self.setModal(True)  
 self.setup\_ui()  
  
 def setup\_ui(self): # Интерфейс окна шахматной доски  
 layout = QVBoxLayout()  
  
 # Создаем сетку для доски  
 board\_widget = QWidget()  
 board\_layout = QGridLayout()  
 board\_layout.setSpacing(1)  
  
 # Создаем клетки доски  
 for y in range(self.board\_size):  
 for x in range(self.board\_size):  
 cell = QFrame()  
 cell.setFixedSize(40, 40)  
 cell.setFrameStyle(QFrame.Box)  
 pos = Position(x, y)  
  
 # Цвета клеток  
 if pos in self.user\_positions:  
 cell.setStyleSheet("background-color: green;") # Уже стоящие фигуры  
 elif pos in self.solution\_positions:  
 cell.setStyleSheet("background-color: red;") # Найденные фигуры  
 elif self.is\_attacked\_position(pos):  
 cell.setStyleSheet("background-color: blue;") # Атакованные позиции  
 else:  
 cell.setStyleSheet("background-color: white;") # Пустые клетки  
  
 board\_layout.addWidget(cell, y, x)  
  
 board\_widget.setLayout(board\_layout)  
 layout.addWidget(board\_widget)  
  
 # Кнопочки  
 button\_layout = QHBoxLayout()  
  
 save\_button = QPushButton("Записать в файл")  
 close\_button = QPushButton("Выход")  
  
 save\_button.clicked.connect(self.save\_to\_file)  
 close\_button.clicked.connect(self.accept)  
  
 button\_layout.addWidget(save\_button)  
 button\_layout.addWidget(close\_button)  
 layout.addLayout(button\_layout)  
  
 self.setLayout(layout)  
  
 def is\_attacked\_position(self, position): # Проверка, атакована ли клетка  
 all\_pieces = self.user\_positions + self.solution\_positions  
  
 for piece\_pos in all\_pieces:  
 piece = CamelPiece(piece\_pos)  
 if piece.can\_attack(position, self.board\_size):  
 return True  
  
 return False  
  
 def save\_to\_file(self): # Сохранение решений в файл  
 try:  
 with open("output.txt", "w", encoding="utf-8") as f:  
 if self.all\_solutions:  
 for solution in self.all\_solutions:  
 line = " ".join(f"({pos.x},{pos.y})" for pos in solution)  
 f.write(line + "\n")  
 else:  
 f.write("no solutions\n")  
  
 QMessageBox.information(self, "Успех", "Данные записаны в файл output.txt")  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Ошибка записи в файл: {str(e)}")  
  
  
class MainWindow(QMainWindow): # Главное окно  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Размещение верблюдов на шахматной доске")  
 self.setFixedSize(400, 200)  
 self.user\_positions = [] # Существующие решения  
 self.solutions = [] # Найденные решения  
 self.setup\_ui()  
 self.update\_button\_states()  
  
 def setup\_ui(self): # Настройка главного окна  
 central\_widget = QWidget()  
 self.setCentralWidget(central\_widget)  
 layout = QVBoxLayout()  
  
 # Поля ввода  
 self.size\_input = QLineEdit()  
 self.size\_input.setPlaceholderText("Размер доски (N)")  
 self.size\_input.textChanged.connect(self.update\_button\_states)  
  
 self.additional\_input = QLineEdit()  
 self.additional\_input.setPlaceholderText("Кол-во требуемых фигур (L)")  
 self.additional\_input.textChanged.connect(self.update\_button\_states)  
  
 self.existing\_input = QLineEdit()  
 self.existing\_input.setPlaceholderText("Кол-во размещённых фигур (K)")  
 self.existing\_input.textChanged.connect(self.update\_button\_states)  
  
 layout.addWidget(QLabel("Размер доски (N)"))  
 layout.addWidget(self.size\_input)  
 layout.addWidget(QLabel("Кол-во требуемых фигур (L)"))  
 layout.addWidget(self.additional\_input)  
 layout.addWidget(QLabel("Кол-во размещённых фигур (K)"))  
 layout.addWidget(self.existing\_input)  
  
 # Кнопочки  
 button\_layout = QHBoxLayout()  
 self.coord\_button = QPushButton("Создать доску")  
 self.draw\_button = QPushButton("Нарисовать доску")  
 self.exit\_button = QPushButton("Выход")  
 self.coord\_button.clicked.connect(self.open\_coordinate\_input)  
 self.draw\_button.clicked.connect(self.solve\_and\_display)  
 self.exit\_button.clicked.connect(self.close)  
 button\_layout.addWidget(self.coord\_button)  
 button\_layout.addWidget(self.draw\_button)  
 button\_layout.addWidget(self.exit\_button)  
 layout.addLayout(button\_layout)  
 central\_widget.setLayout(layout)  
  
 def update\_button\_states(self): # Обновляет кнопки  
 try:  
 size\_text = self.size\_input.text().strip()  
 additional\_text = self.additional\_input.text().strip()  
 existing\_text = self.existing\_input.text().strip()  
  
 # Проверка, что все поля заполнены  
 all\_filled = size\_text and additional\_text and existing\_text  
 if all\_filled:  
 size = int(size\_text)  
 additional = int(additional\_text)  
 existing = int(existing\_text)  
  
 # Проверка допустимости значений  
 valid\_values = (1 <= size <= 20 and additional >= 0 and existing >= 0)  
 if valid\_values:  
 self.coord\_button.setEnabled(True)  
  
 # Условия активации кнопки отрисовки  
 if existing == 0:  
 self.draw\_button.setEnabled(True)  
 else:  
 self.draw\_button.setEnabled(len(self.user\_positions) == existing)  
 else:  
 self.coord\_button.setEnabled(False)  
 self.draw\_button.setEnabled(False)  
 else:  
 self.coord\_button.setEnabled(False)  
 self.draw\_button.setEnabled(False)  
  
 except ValueError:  
 self.coord\_button.setEnabled(False)  
 self.draw\_button.setEnabled(False)  
  
 def open\_coordinate\_input(self): # Открывает окно ввода координат  
 try:  
 existing = int(self.existing\_input.text().strip())  
 size = int(self.size\_input.text().strip())  
  
 if existing == 0:  
 self.user\_positions = []  
 self.update\_button\_states()  
 return  
  
 dialog = CoordinateInputDialog(self, existing, size)  
 if dialog.exec() == QDialog.Accepted:  
 self.user\_positions = dialog.get\_positions()  
 self.update\_button\_states()  
  
 except ValueError:  
 QMessageBox.critical(self, "Ошибка", "Неверные входные данные")  
  
 def solve\_and\_display(self): # Решение задачи и отображение результата  
 try:  
 size = int(self.size\_input.text().strip())  
 additional = int(self.additional\_input.text().strip())  
 existing = int(self.existing\_input.text().strip())  
  
 # Проверка соответствия количества введенных координат  
 if len(self.user\_positions) != existing:  
 QMessageBox.critical(self, "Ошибка",  
 f"Введено {len(self.user\_positions)} координат, а должно быть {existing}")  
 return  
  
 # Поток решения задачи  
 self.solver\_thread = SolverWorker(size, self.user\_positions, additional)  
 self.solver\_thread.solution\_found.connect(self.on\_solution\_found)  
 self.solver\_thread.start()  
 except ValueError:  
 QMessageBox.critical(self, "Ошибка", "Неверные входные данные")  
  
 def on\_solution\_found(self, solutions): # Обработка решения  
 self.solutions = solutions  
  
 if not solutions:  
 QMessageBox.information(self, "Результат", "Решение не найдено")  
 return  
  
 # Первое решение для отображения  
 first\_solution = solutions[0]  
 additional\_positions = [pos for pos in first\_solution if pos not in self.user\_positions]  
  
 # Окно отображения шахматной доски  
 try:  
 size = int(self.size\_input.text().strip())  
 board\_window = BoardDisplayWindow(  
 self,  
 size,  
 self.user\_positions,  
 additional\_positions,  
 solutions  
 )  
 board\_window.exec()  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Ошибка отображения доски: {str(e)}")  
  
  
def main(): # Запуск приложения  
 app = QApplication(sys.argv)  
 window = MainWindow()  
 window.show()  
 sys.exit(app.exec())  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Заключение**

**Список используемых источников**