САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(СПбГМТУ)

Работа по дисциплине

**“Программирование”**

Тип работы: Курсовой проект

Выполнил: Студент 1 курса Факультета Цифровых Промышленных Технологий группы 20121 Хохлов Д.Р.

Преподаватель:   
Кафедра Киберфизических систем

Санкт-Петербург   
2024

**Оглавление**

[Цель работы: 3](#_Toc170561059)

[Формулировка задачи: 3](#_Toc170561060)

[UML диаграмма 3](#_Toc170561061)

[Демонстрация работы программы: 3](#_Toc170561062)

[Листинг кода 3](#_Toc170561063)

[Файл gui.py: 3](#_Toc170561064)

[Файл chess.py: 3](#_Toc170561065)

Цель работы: Анализ ходов фигуры на шахматной доске

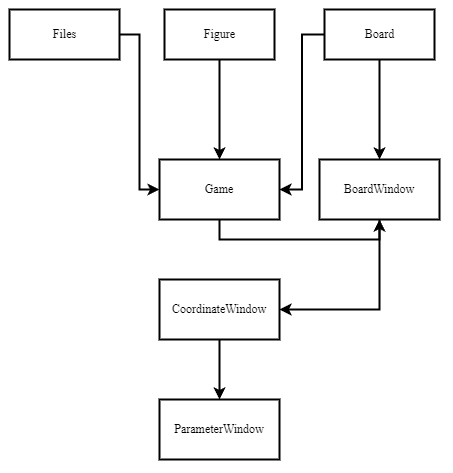
## Формулировка задачи:

Целью данной работы является разработка программы для визуализации размещения фигур на шахматной доске и поиска возможных решений с использованием функционального и объектно-ориентированного программирования (ООП) на языке Python. Программа должна предоставить интерфейс для ввода параметров доски и координат фигур, визуализировать доску с фигурами и ходами, а также сохранять найденные решения в файл.

## UML диаграмма

**В UML-схеме указаны взаимосвязи между классами, а также основные атрибуты и методы классов.**

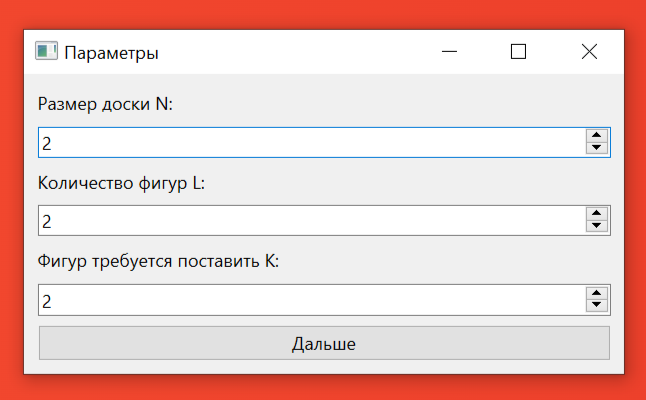
|  |  |  |
| --- | --- | --- |
| **Class** | **Attribute** | **Methods** |
| **Game** | * files: Files * N: int * L: int * K: int * board: Board * steps\_list: List[Tuple[int, int]] * all\_steps: List[List[Tuple[int, int]]] * console\_display: bool | * \_\_init\_\_(self, console\_display=False) * initialize\_game(self, N, L, K, figure\_coordinates) * find\_steps(self, L, board, steps, row=0, col=-1) * get\_all\_steps(self) * display\_info(self, start\_time) |
| **Files** | output\_file\_name: str | * \_\_init\_\_(self, output\_file\_name="output.txt") * set\_data(self, all\_steps) |
| **Figure** | * row: int * col: int *  moves: List[Tuple[int, int]] | \_\_init\_\_(self, row, col)   make\_move(self, board) |
| Board | * N: int *  board: List[List[str]] | * \_\_init\_\_(self, N: int) * clear(self) * \_\_getitem\_\_(self, pos) * \_\_setitem\_\_(self, pos, value) * display(self) |
| **ParameterWindow (QMainWindow)** |  central\_widget: QWidget   layout: QVBoxLayout   label\_n: QLabel   edit\_n: QSpinBox   label\_l: QLabel   edit\_l: QSpinBox   label\_k: QLabel   edit\_k: QSpinBox   button\_next: QPushButton   coordinate\_window: CoordinateWindow |  \_\_init\_\_(self)   * open\_coordinate\_window(self) |
| CoordinateWindow (QMainWindow) |  N: int   L: int   K: int   central\_widget: QWidget   layout: QVBoxLayout   figure\_inputs: List[QLineEdit]   label\_figures: QLabel   button\_next: QPushButton   figure\_coordinates: List[Tuple[int, int]]   board\_window: BoardWindow |  \_\_init\_\_(self, N:int, K:int, L:int)   open\_board\_window(self) |
| BoardWindow (QMainWindow) |  N: int   K: int   L: int   figure\_coordinates: List[Tuple[int, int]]   board: Board   steps\_list: List[Tuple[int, int]]   figure\_moves: List[Tuple[int, int]]   central\_widget: QWidget   layout: QVBoxLayout   scene: QGraphicsScene   * view: QGraphicsView * button\_solve: QPushButton * button\_save: QPushButton *  all\_steps: List[List[Tuple[int, int]]] |  \_\_init\_\_(self, N:int, K:int, L:int, figure\_coordinates)   draw\_board(self)   solve(self)   draw\_solution(self, solution)   save\_to\_file(self) |



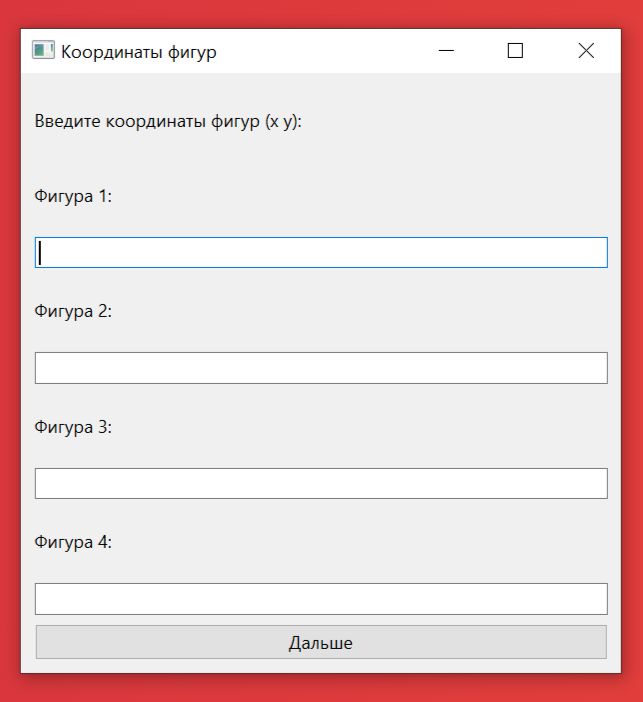
## Демонстрация работы программы:

Окно ввода параметров:

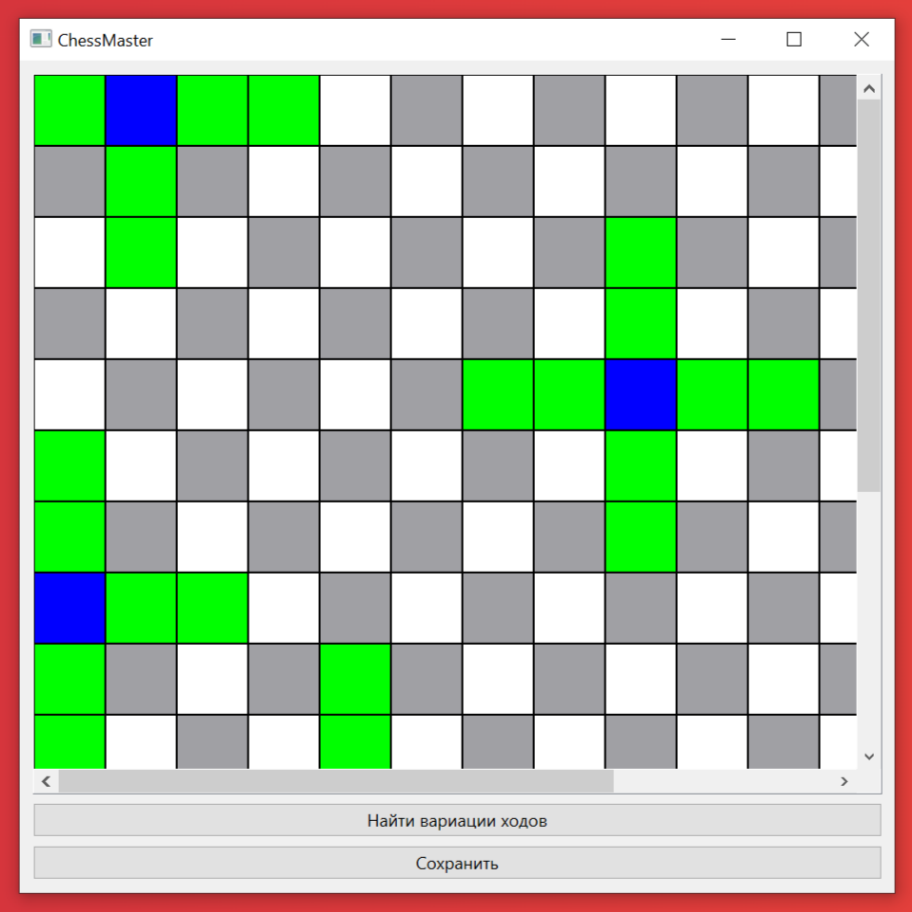
* N – размер доски (NxN, 2=< N <= 20)
* L – Количество фигур для расстановки (2 =< L <= 20)
* K – Количество фигур от пользователя (2 =< L <= 20)



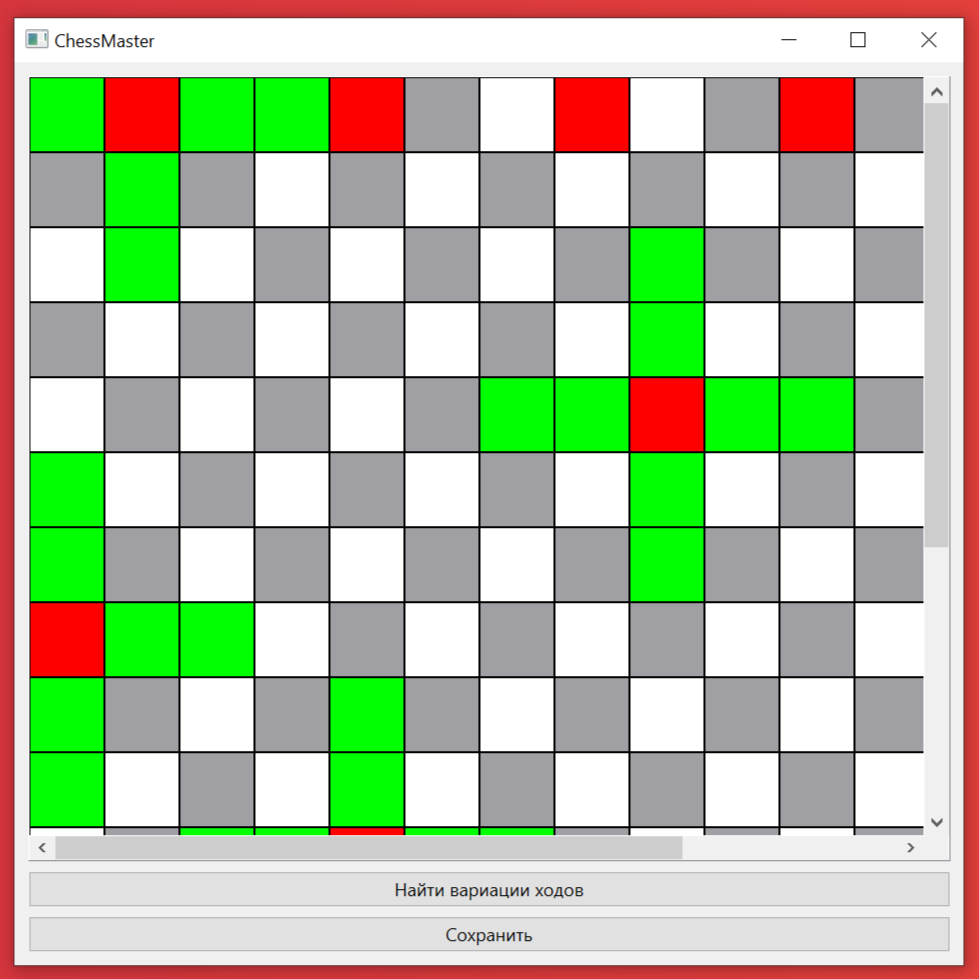
После нажатия кнопки “Дальше”, появится окно расстановки фигур по координатам, количество строк ввода равняется “K”, которое ввели в прошлом окне. Координаты в поле вводятся через пробел (x y)



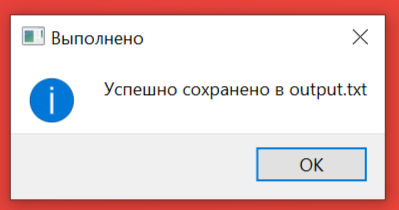
После правильного ввода координат и нажатия на кнопку “Дальше”, откроется новое окно, с шахматной доской, где на нужных полях (по координатам) установлены фигуры.



С помощью нажатия на кнопку “Найти вариации ходов” программа начнёт расчёты всех возможных вариантов, после чего шахматная доска обновится:



Все расчёты можно сохранить нажатием на кнопку “Сохранить”



## Листинг кода

В проекте присутствуют 2 основных файла:

* **gui.py** – файл отвечает за графику
* **chess.py** – файл отвечает за функционал

### Файл gui.py:

import sys  
from PySide6.QtWidgets import QApplication, QMainWindow, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QGraphicsView, QGraphicsScene, QGraphicsRectItem, QMessageBox, QSpinBox  
from PySide6.QtCore import Qt  
from chess import \*  
class ParameterWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Параметры")  
 self.setGeometry(100, 100, 400, 200)  
  
 self.central\_widget = QWidget()  
 self.setCentralWidget(self.central\_widget)  
  
 self.layout = QVBoxLayout()  
 self.central\_widget.setLayout(self.layout)  
  
 self.label\_n = QLabel("Размер доски N:")  
 self.edit\_n = QSpinBox()  
 self.edit\_n.setRange(2,20)  
 self.label\_l = QLabel("Количество фигур L:")  
 self.edit\_l = QSpinBox()  
 self.edit\_l.setRange(2, 20)  
 self.label\_k = QLabel("Фигур требуется поставить K:")  
 self.edit\_k = QSpinBox()  
 self.edit\_k.setRange(2, 20)  
  
 self.button\_next = QPushButton("Дальше")  
 self.button\_next.clicked.connect(self.open\_coordinate\_window)  
  
 self.layout.addWidget(self.label\_n)  
 self.layout.addWidget(self.edit\_n)  
 self.layout.addWidget(self.label\_l)  
 self.layout.addWidget(self.edit\_l)  
 self.layout.addWidget(self.label\_k)  
 self.layout.addWidget(self.edit\_k)  
  
 self.layout.addWidget(self.button\_next)  
  
 def open\_coordinate\_window(self):  
 try:  
 N = int(self.edit\_n.text())  
 K = int(self.edit\_k.text())  
 L = int(self.edit\_l.text())  
 if N <= 0 or K < 0 or L < 0:  
 raise ValueError("Invalid input: N must be positive, K and L must be non-negative")  
 except ValueError as e:  
 QMessageBox.critical(self, "Ошибка", str(e))  
 return  
  
 self.coordinate\_window = CoordinateWindow(N, K, L)  
 self.coordinate\_window.show()  
 self.close()  
  
class CoordinateWindow(QMainWindow):  
 def \_\_init\_\_(self, N:int, K:int, L:int):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Координаты фигур")  
 self.setGeometry(100, 100, 400, 400)  
  
 self.N = N  
 self.L = L  
 self.K = K  
  
 self.central\_widget = QWidget()  
 self.setCentralWidget(self.central\_widget)  
  
 self.layout = QVBoxLayout()  
 self.central\_widget.setLayout(self.layout)  
  
 self.figure\_inputs = []  
  
 self.label\_figures = QLabel("Введите координаты фигур (x y):")  
 self.layout.addWidget(self.label\_figures)  
  
 for i in range(K):  
 label = QLabel(f"Фигура {i + 1}:")  
 edit = QLineEdit()  
 self.figure\_inputs.append(edit)  
 self.layout.addWidget(label)  
 self.layout.addWidget(edit)  
  
 self.button\_next = QPushButton("Дальше")  
 self.button\_next.clicked.connect(self.open\_board\_window)  
 self.layout.addWidget(self.button\_next)  
  
 def open\_board\_window(self):  
 self.figure\_coordinates = []  
 try:  
 for edit in self.figure\_inputs:  
 x, y = map(int, edit.text().split())  
 if not (0 <= x < self.N and 0 <= y < self.N):  
 raise ValueError(f"Coordinates ({x}, {y}) are out of board bounds")  
 self.figure\_coordinates.append((x, y))  
 except ValueError as e:  
 QMessageBox.critical(self, "Ошибка", str(e))  
 return  
  
 self.board\_window = BoardWindow(self.N, self.K, self.L, self.figure\_coordinates)  
 self.board\_window.show()  
 self.close()  
  
class BoardWindow(QMainWindow):  
 def \_\_init\_\_(self, N:int, K:int, L:int, figure\_coordinates):  
 super().\_\_init\_\_()  
 self.setWindowTitle("ChessMaster")  
 self.setGeometry(100, 100, 600, 600)  
  
 self.N = N  
 self.K = K  
 self.L = L  
 self.figure\_coordinates = figure\_coordinates  
  
 self.board = Board(N)  
 self.board.clear()  
 self.steps\_list = []  
 self.figure\_moves = []  
  
 self.central\_widget = QWidget()  
 self.setCentralWidget(self.central\_widget)  
  
 self.layout = QVBoxLayout()  
 self.central\_widget.setLayout(self.layout)  
  
 self.scene = QGraphicsScene()  
 self.view = QGraphicsView(self.scene)  
 self.layout.addWidget(self.view)  
  
 self.button\_solve = QPushButton("Найти вариации ходов")  
 self.button\_solve.clicked.connect(self.solve)  
 self.layout.addWidget(self.button\_solve)  
  
 self.button\_save = QPushButton("Сохранить")  
 self.button\_save.clicked.connect(self.save\_to\_file)  
 self.layout.addWidget(self.button\_save)  
  
 self.draw\_board()  
  
 def draw\_board(self):  
 cell\_size = 50  
 for row in range(self.N):  
 for col in range(self.N):  
 color = Qt.white if (row + col) % 2 == 0 else Qt.gray  
 rect\_item = QGraphicsRectItem(col \* cell\_size, row \* cell\_size, cell\_size, cell\_size)  
 rect\_item.setBrush(color)  
 self.scene.addItem(rect\_item)  
  
 for x, y in self.figure\_coordinates:  
 figure = Figure(x, y)  
 figure.make\_move(self.board)  
 rect\_item = QGraphicsRectItem(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size)  
 rect\_item.setBrush(Qt.blue)  
 self.scene.addItem(rect\_item)  
  
 for row in range(self.N):  
 for col in range(self.N):  
 if self.board[row, col] == "\*":  
 rect\_item = QGraphicsRectItem(col \* cell\_size, row \* cell\_size, cell\_size, cell\_size)  
 rect\_item.setBrush(Qt.green)  
 self.scene.addItem(rect\_item)  
  
 def solve(self):  
 game = Game()  
 game.initialize\_game(self.N, self.L, self.K, self.figure\_coordinates)  
 self.all\_steps = game.get\_all\_steps()  
  
 if not self.all\_steps:  
 QMessageBox.information(self, "Без вариантов(", "Варианты не найдены")  
 else:  
 self.draw\_solution(self.all\_steps[0])  
  
 def draw\_solution(self, solution):  
 cell\_size = 50  
 for x, y in solution:  
 rect\_item = QGraphicsRectItem(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size)  
 rect\_item.setBrush(Qt.red)  
 self.scene.addItem(rect\_item)  
  
 def save\_to\_file(self):  
 if hasattr(self, 'all\_steps'):  
 files = Files()  
 files.set\_data(self.all\_steps)  
 QMessageBox.information(self, "Выполнено", "Успешно сохранено в output.txt")  
 else:  
 QMessageBox.warning(self, "Без вариантов(", "Нет вариантов для сохранения")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 main\_window = ParameterWindow()  
 main\_window.show()  
 sys.exit(app.exec())

### Файл chess.py:

import time  
  
  
  
  
class Board:  
 def \_\_init\_\_(self, N: int):  
 self.N = N  
 self.board = [["0"] \* N for \_ in range(N)]  
  
 def clear(self):  
 self.board = [["0"] \* self.N for \_ in range(self.N)]  
  
 def \_\_getitem\_\_(self, pos):  
 row, col = pos  
 return self.board[row][col]  
  
 def \_\_setitem\_\_(self, pos, value):  
 row, col = pos  
 self.board[row][col] = value  
  
 def display(self):  
 for row in self.board:  
 print(" ".join(row))  
  
class Figure:  
 def \_\_init\_\_(self, row, col):  
 self.row = row  
 self.col = col  
 self.moves = [(row, col - 1), (row, col - 2),  
 (row, col + 1), (row, col + 2),  
 (row - 1, col), (row - 2, col),  
 (row + 1, col), (row + 2, col)]  
  
 def make\_move(self, board):  
 board[self.row, self.col] = "#"  
 for row\_index, col\_index in self.moves:  
 if 0 <= row\_index < board.N and 0 <= col\_index < board.N and board[row\_index, col\_index] != "#":  
 board[row\_index, col\_index] = "\*"  
  
class Files:  
 def \_\_init\_\_(self, output\_file\_name="output.txt"):  
 self.output\_file\_name = output\_file\_name  
 def set\_data(self, all\_steps):  
 with open(self.output\_file\_name, "w") as output\_file:  
 if not all\_steps:  
 output\_file.write("no steps")  
 else:  
 for step in all\_steps:  
 output\_file.write(" ".join(map(str, step)) + "\n")  
  
  
  
class Game:  
 def \_\_init\_\_(self, console\_display=False):  
 self.files = Files()  
 self.N = 0  
 self.L = 0  
 self.K = 0  
 self.board = None  
 self.steps\_list = []  
 self.all\_steps = []  
 self.console\_display = console\_display  
  
 def initialize\_game(self, N, L, K, figure\_coordinates):  
 self.N = N  
 self.L = L  
 self.K = K  
 self.board = Board(N)  
 self.board.clear()  
 self.steps\_list = figure\_coordinates  
  
 for row, col in figure\_coordinates:  
 figure = Figure(row, col)  
 figure.make\_move(self.board)  
  
 def find\_steps(self, L, board, steps, row=0, col=-1):  
 if L == 0:  
 self.all\_steps.append(steps[:])  
 return  
  
 for i in range(row, board.N):  
 for j in range(col + 1 if i == row else 0, board.N):  
 if board[i, j] == "0":  
 new\_board = Board(board.N)  
 new\_board.board = [row[:] for row in board.board]  
 figure = Figure(i, j)  
 figure.make\_move(new\_board)  
 steps.append((i, j))  
 self.find\_steps(L - 1, new\_board, steps, i, j)  
 steps.pop()  
  
 def get\_all\_steps(self):  
 self.all\_steps = []  
 self.find\_steps(self.L, self.board, self.steps\_list[:])  
 return self.all\_steps  
  
 def display\_info(self, start\_time):  
 if self.console\_display:  
 print("#------Результат------#")  
 print(f"Всего решений: {len(self.all\_steps)}")  
 end\_time = time.time()  
 print(f"Время выполнения программы: {round(end\_time - start\_time, 1)}s")