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

КУРСОВОЙ ПРОЕКТ

По дисциплине «Программирование»

Выполнил: Коткин Вадим Иванович

Проверил: Поделенюк Павел Петрович

Санкт-Петербург

2025

Оглавление

[1. Цели и формулировка задачи #](#__RefHeading___1)

2. Разработка классов и UML диаграмма……………………….………#

[3. Результаты работы #](#__RefHeading___2)

[**3.1. Реализация программы с использованием функционального программирования языка Python** #](#__RefHeading___3)

[3.1.1. Ход работы #](#__RefHeading___4)

[3.1.2. Демонстрация работы программы #](#__RefHeading___5)

[3.1.3. Листинг кода #](#__RefHeading___6)

**3.2. Реализация программы с использованием ООП языка**

**Python**……………………………………………………………………….#

3.2.1. Ход работы…………………………………………………...…….#

3.2.2. Демонстрация работы программы………………………………..#

3.2.3. Листинг кода……………………………………………………….#

[ЗАКЛЮЧЕНИЕ #](#__RefHeading___7)

[Список использованных источников #](#__RefHeading___8)

# Цели и формулировка задачи

Цели:

1. Анализ ходов фигуры на шахматной доске.
2. Работа с основами функционального программирования языка Python.
3. Работы с основами ООП языка Python.
4. Разработка классов и UML диаграмма.
5. Работы с пакетами Python.
6. Создание GUI приложения с использованием PySide6.

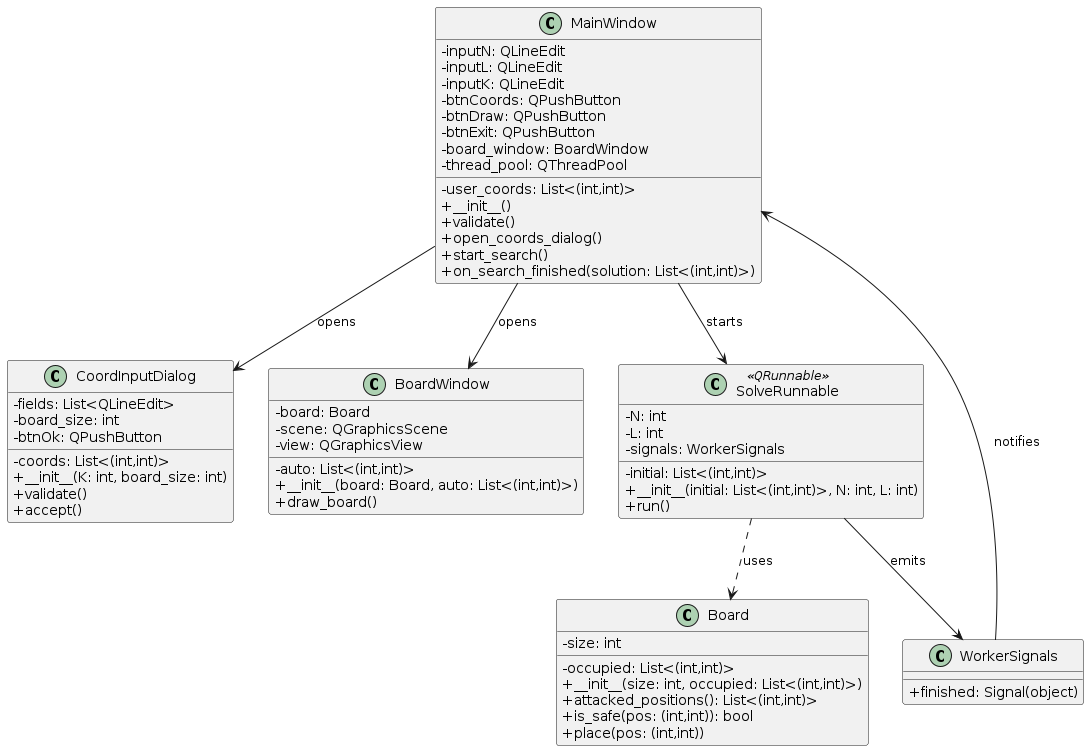
Задача:

Создать приложение на основе кода шахмат прошлого семестра, реализовать его с помощью PySide6.

# Разработка классов и UML диаграмма

# Классы: Board, WorkerSignals, CoordInputDialog, BoardWindow, MainWindow.

UML диаграмма:



1. **Результаты работы**
   1. **Реализация программы с использованием функционального программирования языка Python**
      1. **Ход работы**

1. Инициализация доски (Board)

* + Создается объект доски с указанным размером size (N x N)
  + Принимает начальные координаты занятых позиций occupied (если есть)
  + По умолчанию создается пустая доска (occupied = [])

2. Определение атакованных позиций (attacked\_positions())

* + Для каждой занятой позиции на доске:
    - Проверяются все возможные ходы пони (PONY\_MOVES)
    - Если ход ведет в пределах доски (0 ≤ x, y < size), позиция добавляется в множество атак
  + Возвращается список всех атакованных позиций

3. Проверка безопасности позиции (is\_safe())

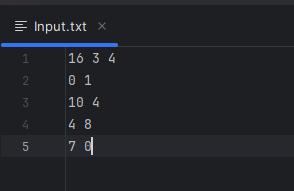
* + Проверяет, не занята ли позиция другим пони
  + Проверяет, не находится ли позиция под атакой других пони:
    - Вычисляются разницы по осям X и Y между текущей и проверяемой позицией
    - Если разницы соответствуют ходу пони [(1,2), (2,1), (1,3), (3,1)], позиция небезопасна

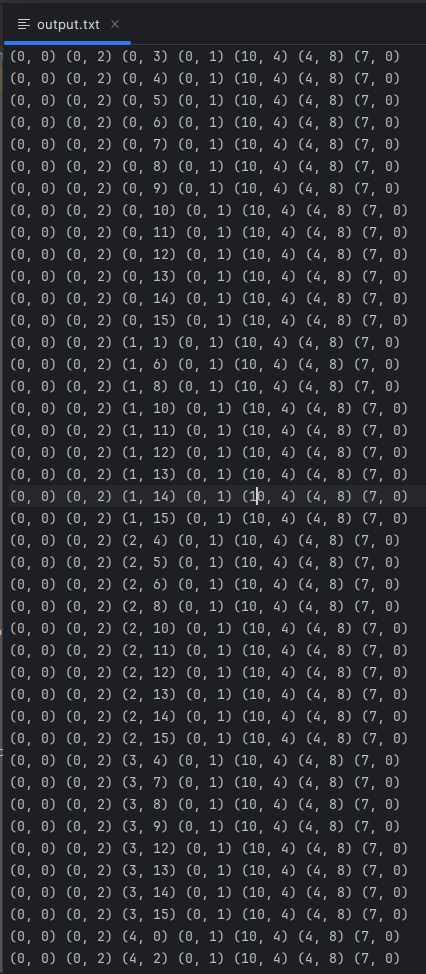
4. Размещение пони (place())

* + Проверяет безопасность позиции через is\_safe()
  + Если позиция безопасна, добавляет координаты в список занятых позиций

5. Поиск решения (find\_one\_solution())

* + Принимает:
    - initial\_coords - начальные позиции пони
    - N - размер доски
    - L - количество дополнительных пони для размещения
  + Использует алгоритм backtracking:
    - Рекурсивно проверяет каждую клетку доски
    - Если клетка безопасна, временно размещает пони и продолжает поиск
    - Если найдено решение (размещены все L пони), возвращает список новых позиций
    - Если решение не найдено, возвращает None
    1. **Демонстрация работы программы**

****

****

* + 1. **Листинг кода**

Coord = Tuple[int, int]  
  
# Ходы пони: конь + верблюд  
PONY\_MOVES = (  
 (1, 2), (2, 1), (-1, 2), (-2, 1), (1, -2), (2, -1), (-1, -2), (-2, -1),  
 (1, 3), (3, 1), (-1, 3), (-3, 1), (1, -3), (3, -1), (-1, -3), (-3, -1)  
)  
  
  
class Board:  
 *"""Класс, представляющий шахматную доску с размещенными пони."""* def \_\_init\_\_(self, size: int, occupied: List[Coord] = None):  
 if occupied is None:  
 occupied = []  
 self.size = size  
 self.occupied = list(occupied)  
  
 def attacked\_positions(self) -> List[Coord]:  
 *"""  
 Возвращает список атакованных позиций на доске.  
  
 :return: Список координат (x, y), находящихся под атакой размещенных пони.  
 """* attacks = set()  
  
 for x0, y0 in self.occupied:  
 for dx, dy in PONY\_MOVES:  
 x, y = x0 + dx, y0 + dy  
  
 if 0 <= x < self.size and 0 <= y < self.size:  
 attacks.add((x, y))  
  
 return list(attacks)  
  
 def is\_safe(self, pos: Coord) -> bool:  
 *"""  
 Проверяет, безопасна ли позиция для размещения нового пони.  
  
 :param pos: Координаты (x, y) для проверки  
  
 :return: True если позиция безопасна, False в противном случае  
  
 """* if pos in self.occupied:  
 return False  
  
 for occ in self.occupied:  
 dx = abs(pos[0] - occ[0])  
 dy = abs(pos[1] - occ[1])  
  
 if (dx, dy) in [(1, 2), (2, 1), (1, 3), (3, 1)]:  
 return False  
  
 return True  
  
 def place(self, pos: Coord):  
 *"""  
 Размещает пони на доске, если позиция безопасна.  
  
 :param pos: Координаты (x, y) для размещения  
  
 """* if self.is\_safe(pos):  
 self.occupied.append(pos)  
  
  
# Бэктрекинг для поиска одного решения  
  
def find\_one\_solution(initial\_coords: List[Coord], N: int, L: int) -> Optional[List[Coord]]:  
 *"""  
 Находит одно решение для размещения L пони на доске NxN с начальными координатами.  
  
 :param initial\_coords: Начальные координаты размещенных пони  
  
 :param N: Размер доски  
  
 :param L: Количество дополнительных пони для размещения  
  
 :return: Список координат дополнительных пони или None, если решение не найдено  
  
 """* solution: List[Coord] = []  
  
 def backtrack(start: int, need: int, occ: List[Coord]) -> bool:  
 if need == 0:  
 solution.extend(occ[len(initial\_coords):])  
 return True  
  
 for i in range(start, N \* N):  
 x, y = divmod(i, N)  
  
 if Board(N, occ).is\_safe((x, y)):  
 occ.append((x, y))  
  
 if backtrack(i + 1, need - 1, occ):  
 return True  
  
 occ.pop()  
  
 return False  
  
 occ\_copy = initial\_coords.copy()  
 if backtrack(0, L, occ\_copy):  
 return solution  
  
 return None

**3.2. Реализация программы с использованием ООП языка**

**Python**

* + 1. **Ход Работы**
       1. Инициализация приложения

Создается главное окно MainWindow с полями ввода:

Размер доски (N)

Количество автоматически размещаемых пони (L)

Количество пользовательских пони (K)

Инициализируется пустой список для хранения координат пользовательских пони

* + - 1. Ввод параметров

Пользователь вводит параметры N, L, K

При вводе K > 0 активируется кнопка "Координаты" для ввода позиций пони

* + - 1. Ввод координат (CoordInputDialog)

При нажатии "Координаты" открывается диалоговое окно:

Создается по K полей ввода (x, y) для каждой фигуры

Реализована валидация:

Проверка формата ввода (два целых числа)

Проверка нахождения в пределах доски (0 ≤ x,y < N)

Проверка, что пони не атакуют друг друга

После успешного ввода координаты сохраняются

* + - 1. Запуск поиска решения

При нажатии "Отрисовать":

Создается задача SolveRunnable для поиска в фоновом потоке

Входные параметры: начальные координаты, размер доски, количество пони для размещения

Кнопка "Отрисовать" временно отключается

* + - 1. Алгоритм поиска (find\_one\_solution)

Используется backtracking для поиска допустимой расстановки:

Рекурсивно проверяются все клетки доски

Для каждой безопасной клетки (is\_safe):

Временно размещается пони

Продолжается поиск для оставшихся пони

Если решение найдено - возвращается список координат

Если нет - откатывается последнее размещение

Результат передается через сигнал WorkerSignals.finished

* + - 1. Отрисовка результата (BoardWindow)

При успешном поиске:

Создается объект доски с начальными и найденными координатами

Открывается новое окно с визуализацией:

Белая сетка - пустые клетки

Красные/розовые - атакованные клетки

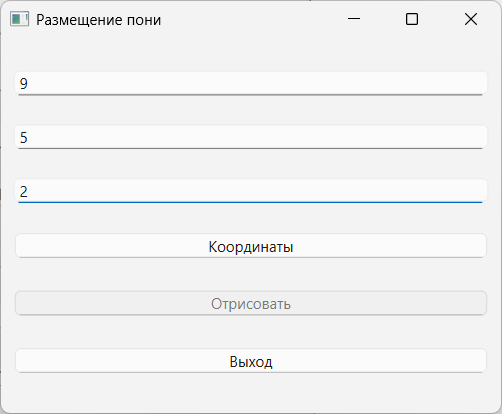
Синие - пользовательские пони

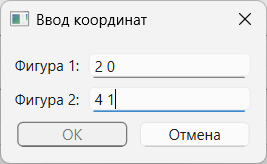
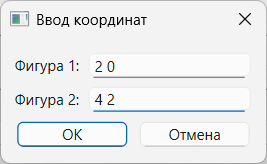
Красные - автоматически размещенные пони

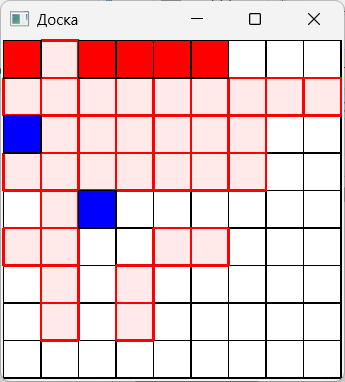
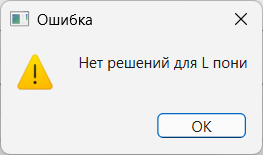
При неудаче - выводится сообщение об ошибке

* + - 1. Прописать условия по заданию.
      2. Привести программу в действие.

**3.2.2 Демонстрация работы программы**



* + 1. **Листинг кода**

import sys  
from typing import List, Tuple, Optional  
  
from PySide6.QtWidgets import (  
 QApplication, QMainWindow, QWidget, QLineEdit, QPushButton,  
 QVBoxLayout, QHBoxLayout, QGraphicsScene, QGraphicsView, QGraphicsRectItem,  
 QMessageBox, QDialog, QLabel  
)  
  
from PySide6.QtCore import Qt, QRunnable, QThreadPool, Signal, QObject  
from PySide6.QtGui import QPen, QColor  
  
Coord = Tuple[int, int]  
  
# Ходы пони: конь + верблюд  
PONY\_MOVES = (  
 (1, 2), (2, 1), (-1, 2), (-2, 1), (1, -2), (2, -1), (-1, -2), (-2, -1),  
 (1, 3), (3, 1), (-1, 3), (-3, 1), (1, -3), (3, -1), (-1, -3), (-3, -1)  
)  
  
  
class Board:  
 *"""Класс, представляющий шахматную доску с размещенными пони."""* def \_\_init\_\_(self, size: int, occupied: List[Coord] = None):  
 if occupied is None:  
 occupied = []  
 self.size = size  
 self.occupied = list(occupied)  
  
 def attacked\_positions(self) -> List[Coord]:  
 *"""  
 Возвращает список атакованных позиций на доске.  
  
 :return: Список координат (x, y), находящихся под атакой размещенных пони.  
 """* attacks = set()  
  
 for x0, y0 in self.occupied:  
 for dx, dy in PONY\_MOVES:  
 x, y = x0 + dx, y0 + dy  
  
 if 0 <= x < self.size and 0 <= y < self.size:  
 attacks.add((x, y))  
  
 return list(attacks)  
  
 def is\_safe(self, pos: Coord) -> bool:  
 *"""  
 Проверяет, безопасна ли позиция для размещения нового пони.  
  
 :param pos: Координаты (x, y) для проверки  
  
 :return: True если позиция безопасна, False в противном случае  
  
 """* if pos in self.occupied:  
 return False  
  
 for occ in self.occupied:  
 dx = abs(pos[0] - occ[0])  
 dy = abs(pos[1] - occ[1])  
  
 if (dx, dy) in [(1, 2), (2, 1), (1, 3), (3, 1)]:  
 return False  
  
 return True  
  
 def place(self, pos: Coord):  
 *"""  
 Размещает пони на доске, если позиция безопасна.  
  
 :param pos: Координаты (x, y) для размещения  
  
 """* if self.is\_safe(pos):  
 self.occupied.append(pos)  
  
  
# Бэктрекинг для поиска одного решения  
  
def find\_one\_solution(initial\_coords: List[Coord], N: int, L: int) -> Optional[List[Coord]]:  
 *"""  
 Находит одно решение для размещения L пони на доске NxN с начальными координатами.  
  
 :param initial\_coords: Начальные координаты размещенных пони  
  
 :param N: Размер доски  
  
 :param L: Количество дополнительных пони для размещения  
  
 :return: Список координат дополнительных пони или None, если решение не найдено  
  
 """* solution: List[Coord] = []  
  
 def backtrack(start: int, need: int, occ: List[Coord]) -> bool:  
 if need == 0:  
 solution.extend(occ[len(initial\_coords):])  
 return True  
  
 for i in range(start, N \* N):  
 x, y = divmod(i, N)  
  
 if Board(N, occ).is\_safe((x, y)):  
 occ.append((x, y))  
  
 if backtrack(i + 1, need - 1, occ):  
 return True  
  
 occ.pop()  
  
 return False  
  
 occ\_copy = initial\_coords.copy()  
 if backtrack(0, L, occ\_copy):  
 return solution  
  
 return None  
  
  
class WorkerSignals(QObject):  
 *"""Сигналы для работы воркера."""* finished = Signal(object)  
  
  
class SolveRunnable(QRunnable):  
 *"""Задача для выполнения поиска решения в отдельном потоке."""* def \_\_init\_\_(self, initial: List[Coord], N: int, L: int):  
 super().\_\_init\_\_()  
  
 self.initial = initial  
 self.N = N  
 self.L = L  
 self.signals = WorkerSignals()  
  
 def run(self):  
 *"""Выполняет поиск решения и испускает сигнал с результатом."""* result = find\_one\_solution(self.initial, self.N, self.L)  
 self.signals.finished.emit(result)  
  
  
class CoordInputDialog(QDialog):  
 *"""Диалоговое окно для ввода координат пони."""* def \_\_init\_\_(self, K: int, board\_size: int, parent=None):  
 super().\_\_init\_\_(parent)  
  
 self.setWindowTitle("Ввод координат")  
 self.setModal(True)  
 self.coords: List[Coord] = []  
 self.fields = []  
 layout = QVBoxLayout()  
  
 for i in range(K):  
 row = QHBoxLayout()  
 label = QLabel(f"Фигура {i + 1}:")  
 line = QLineEdit()  
 line.setPlaceholderText("x y")  
  
 row.addWidget(label)  
 row.addWidget(line)  
  
 layout.addLayout(row)  
 self.fields.append(line)  
 line.textChanged.connect(self.validate)  
  
 btn\_layout = QHBoxLayout()  
 self.btnOk = QPushButton("OK")  
 self.btnOk.setEnabled(False)  
  
 btnCancel = QPushButton("Отмена")  
  
 btn\_layout.addWidget(self.btnOk)  
 btn\_layout.addWidget(btnCancel)  
 layout.addLayout(btn\_layout)  
  
 self.setLayout(layout)  
 self.btnOk.clicked.connect(self.accept)  
 btnCancel.clicked.connect(self.reject)  
  
 self.board\_size = board\_size  
  
 def validate(self):  
 *"""Проверяет валидность введенных координат."""* valid = True  
 tmp: List[Coord] = []  
  
 for line in self.fields:  
 text = line.text().strip()  
 try:  
 x, y = map(int, text.split())  
 if not (0 <= x < self.board\_size and 0 <= y < self.board\_size):  
 valid = False  
  
 tmp.append((x, y))  
 except:  
 valid = False  
  
 if valid:  
 for i in range(len(tmp)):  
 for j in range(i + 1, len(tmp)):  
 dx = abs(tmp[i][0] - tmp[j][0])  
 dy = abs(tmp[i][1] - tmp[j][1])  
  
 if (dx, dy) in [(1, 2), (2, 1), (1, 3), (3, 1)]:  
 valid = False  
 break  
  
 if not valid:  
 break  
  
 self.btnOk.setEnabled(valid)  
  
 def accept(self):  
 *"""Обрабатывает подтверждение ввода координат."""* self.coords = [tuple(map(int, line.text().split())) for line in self.fields]  
 super().accept()  
  
  
class BoardWindow(QMainWindow):  
 *"""Окно для отображения шахматной доски с пони."""* def \_\_init\_\_(self, board: Board, auto: List[Coord]):  
 super().\_\_init\_\_()  
  
 self.setWindowTitle("Доска")  
 self.board = board  
 self.auto = auto  
 self.scene = QGraphicsScene()  
 self.view = QGraphicsView(self.scene)  
 self.setCentralWidget(self.view)  
  
 self.draw\_board()  
  
 def draw\_board(self):  
 *"""Отрисовывает доску с пони и атакованными позициями."""* N = self.board.size  
 cell = 30  
 pen = QPen(Qt.black)  
  
 # рисуем сетку  
 for i in range(N):  
 for j in range(N):  
 rect = QGraphicsRectItem(j \* cell, i \* cell, cell, cell)  
 rect.setPen(pen)  
 rect.setBrush(Qt.white)  
  
 self.scene.addItem(rect)  
  
 # атакованные клетки  
 pen\_attack = QPen(QColor(255, 0, 0), 2)  
 brush\_attack = QColor(255, 200, 200, 100)  
 for x, y in self.board.attacked\_positions():  
 rect = QGraphicsRectItem(y \* cell, x \* cell, cell, cell)  
 rect.setPen(pen\_attack)  
 rect.setBrush(brush\_attack)  
  
 self.scene.addItem(rect)  
  
 # пользовательские пони  
 for x, y in self.board.occupied:  
 rect = QGraphicsRectItem(y \* cell, x \* cell, cell, cell)  
 rect.setBrush(Qt.blue)  
 rect.setPen(pen)  
  
 self.scene.addItem(rect)  
  
 # автоматически расставленные пони  
 for x, y in self.auto:  
 rect = QGraphicsRectItem(y \* cell, x \* cell, cell, cell)  
 rect.setBrush(Qt.red)  
 rect.setPen(pen)  
  
 self.scene.addItem(rect)  
  
  
class MainWindow(QMainWindow):  
 *"""Главное окно приложения для размещения пони."""* def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.setWindowTitle("Размещение пони")  
 self.resize(400, 300)  
 self.user\_coords: List[Coord] = []  
 self.board\_window: Optional[BoardWindow] = None  
 self.thread\_pool = QThreadPool.globalInstance()  
  
 # поля ввода  
 self.inputN = QLineEdit();  
 self.inputL = QLineEdit();  
 self.inputK = QLineEdit()  
 self.inputN.setPlaceholderText("Размер доски N")  
 self.inputL.setPlaceholderText("Автопони L")  
 self.inputK.setPlaceholderText("Польз. пони K")  
  
 # кнопки  
 self.btnCoords = QPushButton("Координаты");  
 self.btnDraw = QPushButton("Отрисовать");  
 self.btnExit = QPushButton("Выход")  
 self.btnCoords.setEnabled(False);  
 self.btnDraw.setEnabled(False)  
  
 # слой  
 layout = QVBoxLayout()  
 for w in [self.inputN, self.inputL, self.inputK, self.btnCoords, self.btnDraw, self.btnExit]:  
 layout.addWidget(w)  
  
 container = QWidget();  
 container.setLayout(layout)  
 self.setCentralWidget(container)  
  
 # сигналы  
 for inp in [self.inputN, self.inputL, self.inputK]:  
 inp.textChanged.connect(self.validate)  
  
 self.btnCoords.clicked.connect(self.open\_coords\_dialog)  
 self.btnDraw.clicked.connect(self.start\_search)  
 self.btnExit.clicked.connect(self.close)  
  
 def validate(self):  
 *"""Проверяет валидность введенных параметров."""* N\_ok = self.inputN.text().isdigit()  
 L\_ok = self.inputL.text().isdigit()  
 K\_ok = self.inputK.text().isdigit()  
  
 if N\_ok and L\_ok and K\_ok:  
 N = int(self.inputN.text());  
 K = int(self.inputK.text())  
  
 self.btnCoords.setEnabled(K > 0)  
 self.btnDraw.setEnabled(K == 0 or len(self.user\_coords) == K)  
 else:  
 self.btnCoords.setEnabled(False)  
 self.btnDraw.setEnabled(False)  
  
 def open\_coords\_dialog(self):  
 *"""Открывает диалог ввода координат пользовательских пони."""* K = int(self.inputK.text());  
 N = int(self.inputN.text())  
 dlg = CoordInputDialog(K, N, self)  
  
 if dlg.exec() == QDialog.Accepted:  
 self.user\_coords = dlg.coords  
  
 self.validate()  
  
 def start\_search(self):  
 *"""Запускает поиск решения в отдельном потоке."""* N = int(self.inputN.text());  
 L = int(self.inputL.text())  
 initial = self.user\_coords.copy()  
 self.btnDraw.setEnabled(False)  
  
 worker = SolveRunnable(initial, N, L)  
 worker.signals.finished.connect(self.on\_search\_finished)  
  
 self.thread\_pool.start(worker)  
  
 def on\_search\_finished(self, solution: Optional[List[Coord]]):  
 *"""  
 Обрабатывает завершение поиска решения.  
  
 :param solution: Найденное решение или None, если решение не найдено  
  
 """* if solution is None:  
 QMessageBox.warning(self, "Ошибка", "Нет решений для L пони")  
 self.btnDraw.setEnabled(True)  
 return  
  
 N = int(self.inputN.text())  
 board = Board(N, self.user\_coords.copy())  
  
 for coord in solution:  
 board.place(coord)  
  
 self.board\_window = BoardWindow(board, solution)  
 self.board\_window.show()  
 self.btnDraw.setEnabled(True)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 mw = MainWindow()  
 mw.show()  
 sys.exit(app.exec())

# ЗАКЛЮЧЕНИЕ

В ходе работы мы написали программу, для работы приложения, в основе которого был использован код «шахматы» прошлого семестра. Более того, была изучена и использована библиотека PySide6, для разработки GUI приложения шахмат.

# Список использованных источников

# <https://www.pythonguis.com/tutorials/pyside6-layouts/>

1. <https://fadeevlecturer.github.io/python_lectures/notebooks/qt/basics.html>
2. https://www.pythoncentral.io/pyside6-how-to-build-gui/