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: Стеценко Александр Алексеевич

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

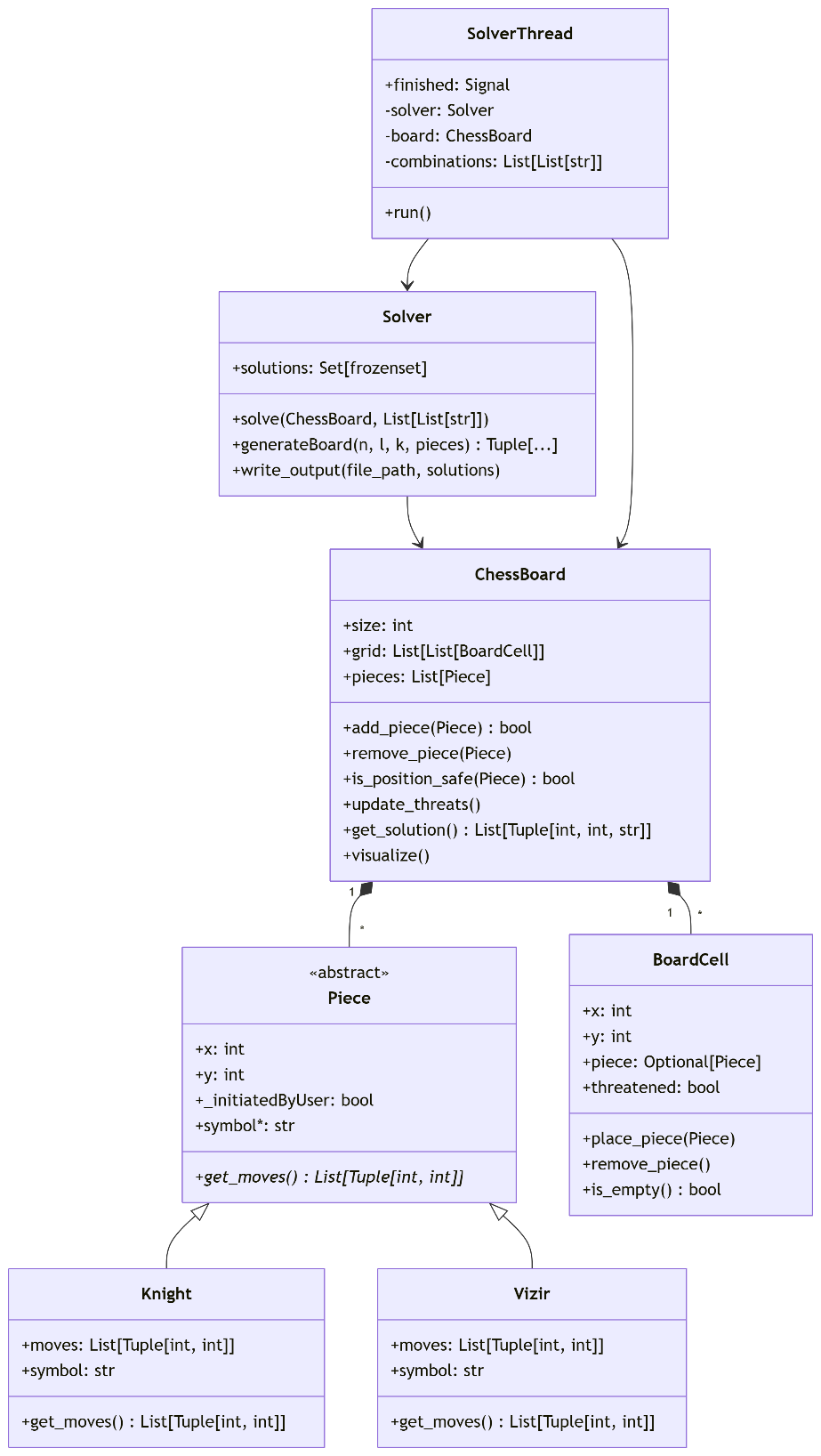
**Цель:**

1. Работа с основами функционального программирования языка Python.
2. Работы с основами ООП языка Python.
3. Разработка классов и UML диаграмма.
4. Работы с пакетами Python.
5. Создание GUI приложения с использованием PySide6.

**Задачи:**

1. Реализовать алгоритм на Python с использованием функционального и ООП подходов.
2. Создать GUI с помощью PySide6 для интерактивного взаимодействия.
3. Построить UML-диаграммы классов.
4. Обеспечить проверку входных данных и многопоточность для отзывчивости интерфейса.

# Разработка классов и UML диаграмма



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

Перед началом работы была формулирована задача: на квадратной шахматной доске размером N × N разместить L фигур (кони и визири) так, чтобы ни одна фигура не находилась под боем других. При этом учитываются комбинации типов фигур (конь, визирь).

**2. Анализ проблемы**

* **Входные данные:**
  + Размер доски (N).
  + Комбинации типов фигур (например, [('v', 'v'), ('v', 'k'), ('k', 'k')]).
  + Координаты уже размещенных фигур.
* **Выходные данные:**
  + Все возможные расположения L фигур, соблюдающие условия задачи.

**3. Методология решения**

* **Первый шаг:** Разработана функция is\_safe, которая проверяет, безопасно ли разместить фигуру (конь или визирь) на заданной клетке, учитывая ее ходы и координаты уже существующих фигур.
* **Второй шаг:** Объявление функции backtrack, которая реализует рекурсивный поиск расстановок. Она перебирает все клетки доски и проверяет возможность безопасного размещения фигуры из заданной комбинации.
  + Если фигура может быть размещена, она добавляется в текущее множество фигур, и вызывается следующий уровень рекурсии.
  + При достижении базового условия (размещены все L фигур) результат сохраняется в список решений.
  + После завершения текущего вызова фигура удаляется, чтобы попробовать другие варианты.
* **Третий шаг:** Обработка комбинаций фигур. Программа принимает комбинации вида [('v', 'v'), ('v', 'k'), ('k', 'k')] и распределяет заданные L фигур между типами, сохраняя корректность их размещения.

**4. Оптимизация алгоритма**

Для обеспечения эффективности работы на досках размером до 20 × 20 были выполнены следующие шаги:

* Уменьшение пространства поиска за счет проверки только безопасных позиций с помощью is\_safe.
* Использование множества для хранения уже размещенных фигур, что упрощает проверку пересечений.
* Прекращение дальнейших вычислений, если промежуточное размещение не может привести к корректному решению.

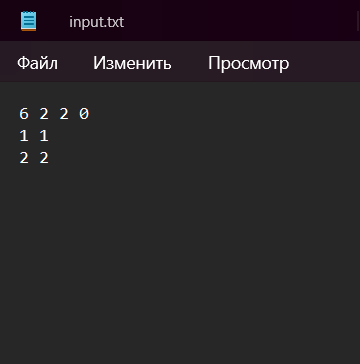
**5. Тестирование и валидация**

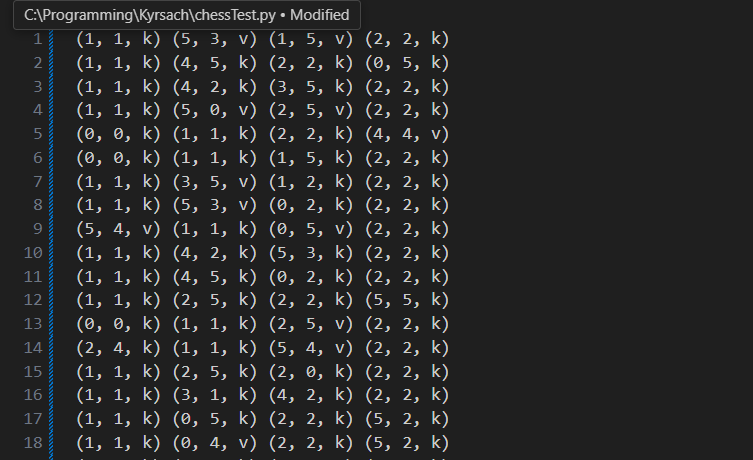
Программа была протестирована на различных входных данных, включая:

* Малые доски (3 × 3, 4 × 4) для проверки базовой функциональности.
* Средние доски (8 × 8, 10 × 10) с разным количеством фигур и комбинациями.
* Большие доски (20 × 20) для оценки производительности алгоритма.

Результаты тестирования показали, что алгоритм корректно находит все возможные комбинации фигур или сообщает об их отсутствии, если расстановка невозможна.

* + 1. **Демонстрация работы программы**





* + 1. **Листинг кода**

1. """
2. Считываем файл с данными в формате: N L K V, где N - размерность поля, L - количество фигур, которые нужно поставить на доску(так как у нас две фигуры, то это число отражает выборку фигур с повторениями)
3. K - уже существующие кони, V - уже существующие визири.
4. Записываем в новый файл все возможнные позиции фигур не под боем
5. """
6. from itertools import combinations\_with\_replacement
7. def add\_piece(x,y,piece\_type,table,n):
8. table[y][x] = piece\_type
9. mark\_position(x,y,piece\_type,table,n)
10. def remove\_piece(x,y,piece\_type,table,n):
11. table[y][x] = '0'
12. unmark\_position(x,y,piece\_type,table,n)
14. def mark\_position(x,y, piece\_type,table,n):
15. """
16. Функция создания на матрице позиций под боем
17. """
18. piece\_moves = {
19. # конь
20. "k":[(-2, -1), (-2, 1), (2, -1), (2, 1),
21. (-1, -2), (-1, 2), (1, -2), (1, 2)],
22. # виверь
23. "v":[(-1, 0), (1, 0),
24. (0, -1), (0, 1)]
25. }
26. for dx,dy in piece\_moves[piece\_type]:
27. nx,ny = x + dx, y + dy
28. if 0 <= x + dx < n and 0 <= y + dy < n:
29. if table[ny][nx] == '0':  # Помечаем только свободные клетки
30. table[ny][nx] = '\*'
31. def unmark\_position(x,y, piece\_type,table,n):
32. """
33. Функция удаления на матрице позиций под боем
34. """
35. piece\_moves = {
36. # конь
37. "k":[(-2, -1), (-2, 1), (2, -1), (2, 1),
38. (-1, -2), (-1, 2), (1, -2), (1, 2)],
39. # виверь
40. "v":[(-2, 0), (-1, 0), (2, 0), (1, 0),
41. (0, -2), (0, 2), (0, -1), (0, 1)]
42. }
43. for dx,dy in piece\_moves[piece\_type]:
44. nx,ny = x + dx, y + dy
45. if 0 <= x + dx < n and 0 <= y + dy < n: table[ny][nx] = '0'
46. def read\_input(file\_path:str) -> list:
47. """
48. Функция чтения и обработки данных из файла input.txt
50. :param
51. file\_path(str): Входное значение с адрессом входного файла
52. :return
53. (list): Возвращает такие переменные, как:
54. n(int): Размерность поля
55. l(int): Количество размещаемых фигур
56. k(int): Количество уже размещенных фигур
57. existing\_piece(list): Список координат всех существующих фигур с названием фигуры
58. permut\_piece(list): Список всех сочетаний из L фигур, таких как конь и визирь
59. """
60. with open(file\_path, 'r') as file:
61. lines = file.readlines()
63. n, l, k, v = map(int, lines[0].strip().split())
64. table = [["0"]\*n for \_ in range(n)]
65. existing\_piece = [list(map(int,line.strip().split())) for line in lines[1:] if line.strip().split()]
66. permut\_piece = list(combinations\_with\_replacement('kv', l))
67. for i in range(k+v):
68. if i < k: existing\_piece[i].append("k")
69. else: existing\_piece[i].append("v")
70. for x,y,f in existing\_piece:
71. add\_piece(x,y,f,table,n)
72. existing\_piece = tuple(tuple(item) for item in existing\_piece)
74. return n, l, k, v, existing\_piece, permut\_piece, table
75. def write\_output(file\_path:str, solutions:tuple) -> None:
76. """
77. Запись результатов в output.txt.
78. :param
79. file\_path(str): Входное значение с адрессом входного файла
80. solutions(str): Список всех уникальных значений координат расстановок фигур
81. :return
82. None
83. """
84. with open(file\_path, 'w') as file:
85. if solutions:
86. for solution in solutions:
87. solution\_str = " ".join("({}, {}, {})".format(x, y, f) for x, y, f in solution)
88. file.write(solution\_str + "\n")
89. else:
90. file.write("no solutions\n")
91. def get\_solution(n,table) -> tuple:
92. solution = []
93. for y in range(n):
94. for x in range(n):
95. if table[y][x] == 'k':
96. solution.append((x,y,'k'))
97. if table[y][x] == 'v':
98. solution.append((x,y,'v'))
99. return solution
100. def is\_safe(x:int, y:int, piece\_type:str, table: tuple) -> bool:
101. """
102. Проверка, безопасно ли поставить фигуру в клетку (x, y).
103. :param
104. x(int) & y(int): Входные значения с координатами фигуры на поле
105. piece\_type(str): Строкое значение, обозначающее тип фигуры: "v" - визирь и "k" - конь
106. table(tuple): Текущая матрица позиций фигур
108. :return
109. (bool): Если фигура не находится под боем в этих координатах, то возвращаем True, иначе False
110. """
111. if table[y][x] != '0':
112. return False  # Клетка уже занята или под боем
114. piece\_moves = {
115. # конь
116. "k":[(-2, -1), (-2, 1), (2, -1), (2, 1),
117. (-1, -2), (-1, 2), (1, -2), (1, 2)],
118. # виверь
119. "v":[(-2, 0), (-1, 0), (2, 0), (1, 0),
120. (0, -2), (0, 2), (0, -1), (0, 1)]
121. }
123. # Проверка: не атакует ли новая фигура кого-то
124. for dx, dy in piece\_moves[piece\_type]:
125. nx, ny = x + dx, y + dy
126. if 0 <= nx < n and 0 <= ny < n:
127. if table[ny][nx] in ('k', 'v'):  # Фигура бьёт существующую
128. return False
129. # Проверка: не атакует ли кто-то новую фигуру
130. for fx in range(n):
131. for fy in range(n):
132. if table[fy][fx] in ('k', 'v'):
133. for dx, dy in piece\_moves[table[fy][fx]]:
134. if (fx + dx == x) and (fy + dy == y):
135. return False
136. # for dx, dy in piece\_moves['k']:
137. #     if 0 <= x + dx < n and 0 <= y + dy < n:
138. #         if table[y + dy][x + dx] == "k": return False
139. # for dx, dy in piece\_moves['v']:
140. #     if 0 <= x + dx < n and 0 <= y + dy < n:
141. #         if table[y + dy][x + dx] == "v": return False
142. return 0 <= x < n and 0 <= y < n
143. def place\_pieces\_with\_combinations(table,combinations:list) -> list:
144. """
145. Поиск всех возможных расстановок дополнительных фигур.
147. :param
148. n(int): Размерность поля
149. existing\_piece(list): Список координат всех существующих фигур с названием фигуры
150. combinations(list): Список всех сочетаний из L фигур, таких как конь и визирь
152. :return
153. solutions(list): Список всех возможных расположений фигур не под боем
154. """
155. solutions = []
156. def backtrack(remaining\_combination:list, table:list) -> None:
157. """
158. Рекурсивная функция для расположения всех фигур в безопастные позиции
160. :param
161. remaining\_combination(list): Оставшиеся фигуры для расположения на поле
162. pieces(list): Список координат всех существующих фигур с названием фигуры
163. :return
164. (None)
165. """
166. if not remaining\_combination:
167. solutions.append(get\_solution(n,table))
168. return
169. piece\_type = remaining\_combination[0]  # Берем первый тип фигуры из комбинации
170. for x in range(n):
171. for y in range(n):
172. if is\_safe(x, y, piece\_type, table):
173. add\_piece(x, y, piece\_type,table,n)
174. backtrack(remaining\_combination[1:], table)
175. remove\_piece(x, y, piece\_type,table,n)
176. # Перебор всех комбинаций
177. for combination in combinations:
178. backtrack(combination, table)
179. return solutions
180. if \_\_name\_\_ == "\_\_main\_\_":
181. # Чтение входных данных
182. n, l, k, v, existing\_pieces, permut\_piece, table = read\_input("input.txt")
183. print(n, l, k, v, existing\_pieces, permut\_piece)
184. for row in table:
185. print(row)
186. # Проверка корректности входных данных
187. assert all(0 <= x < n and 0 <= y < n for x, y, f in existing\_pieces), "Некорректные координаты входных данных."
188. # Поиск решений
189. solutions = place\_pieces\_with\_combinations(table,permut\_piece)
191. # # Избавляемся от повторяющихся позиций
192. solutions = set(tuple(solution) for solution in solutions)
193. # Запись решений в файл
194. write\_output("output.txt", solutions)
195. print(len(solutions))

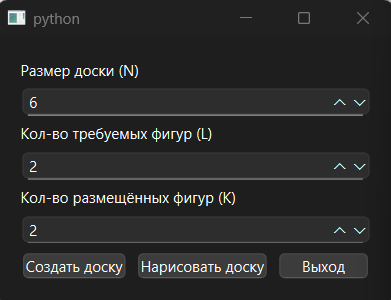
**3.2. Реализация программы с использованием ООП языка**

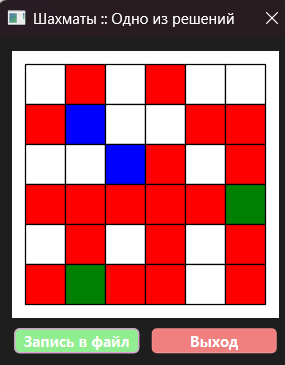
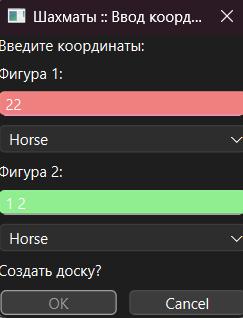
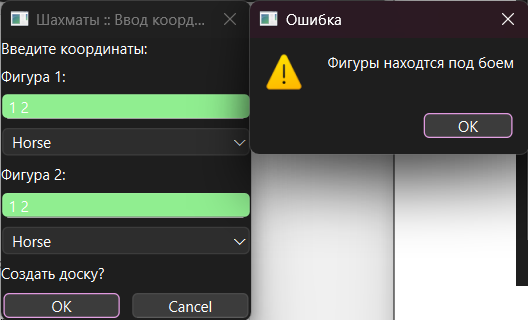
* + 1. **Ход Работы**

**1. Анализ задачи и проектирование структуры**

Перед написанием кода была проведена детальная проработка требований:

* Определены **сущности**: доска (ChessBoard), клетка (BoardCell), фигуры (Piece, Knight, Vizir), основная логика (Solver).
* Выбраны **парадигмы программирования**: ООП для основной логики, функциональный стиль для вспомогательных функций (например, проверки границ).
* Спроектированы **UML-диаграммы** для наглядного представления взаимодействия классов.
  + 1. **Демонстрация работы программы**



* + 1. **Листинг кода**

1. from itertools import combinations\_with\_replacement
2. from typing import List, Tuple, Optional, Set
3. from abc import ABC, abstractmethod
4. # from grafics import \*
5. from PySide6.QtWidgets import (
6. QApplication,
7. QMainWindow,
8. QDialog,
9. QPushButton,
10. QSpinBox,
11. QComboBox,
12. QLabel,
13. QLineEdit,
14. QVBoxLayout,
15. QHBoxLayout,
16. QMessageBox,
17. QWidget,
18. QTableWidget,
19. QTableWidgetItem,
20. QGraphicsRectItem,
21. QGraphicsScene,
22. QGraphicsView,
23. QGraphicsRectItem,
24. QGraphicsSimpleTextItem
25. )
26. from PySide6.QtCore import Qt, QFile, QThread, Signal
27. from PySide6.QtGui import QBrush, QPen, QColor, QFont, QPainter
28. from PySide6.QtUiTools import QUiLoader
29. # import sys
30. from re import fullmatch
31. class InputCoordinatesDialog(QDialog):
32. """
33. Диалоговое окно для ручного ввода координат фигур на шахматной доске.
34. """
35. def \_\_init\_\_(self, num\_figures: int, parent: Optional[QDialog] = None):
36. """
37. Конструктор диалога координат.
38. :param num\_figures (int): Количество фигур для размещения
39. :param parent (Optional[QDialog]): Родительское окно
40. """
41. super().\_\_init\_\_(parent)
42. self.Parent = parent
43. self.num\_figures = num\_figures
44. self.coordinate\_inputs = []  # список QLineEdit
45. self.figure\_selectors = []   # список QComboBox
46. self.resFig = []
47. self.load\_ui()
48. self.create\_input\_fields()
49. self.setWindowTitle("Шахматы :: Ввод координат")
50. @staticmethod
51. def is\_valid\_format(s: str) -> bool:
52. """
53. Проверяет, соответствует ли строка формату "X Y" (два целых числа через пробел).
55. :param s: строка для проверки
56. :return: True, если формат корректный
57. """
58. return bool(fullmatch(r"\d+\s\d+", s.strip()))
59. def load\_ui(self):
60. """
61. Загружает интерфейс из .ui-файла и подключает кнопки.
62. """
63. loader = QUiLoader()
64. file = QFile(r"ui/input\_coordinates.ui")
65. if not file.open(QFile.ReadOnly):
66. QMessageBox.critical(None, "Ошибка", "Не удалось открыть файл UI!")
67. return
68. self.ui = loader.load(file, self)
69. file.close()
70. self.okButton = self.ui.findChild(QPushButton, "okButton")
71. self.cancelButton = self.ui.findChild(QPushButton, "cancelButton")
72. self.mainLayout = self.ui.findChild(QVBoxLayout, "verticalLayout")
73. self.coordinatesLayout = self.ui.findChild(QVBoxLayout, "coordinatesLayout")
74. self.okButton.clicked.connect(self.accept)
75. self.cancelButton.clicked.connect(self.reject)
76. for input in self.coordinate\_inputs:
77. input.textChanged.connect(self.color)
78. self.setLayout(self.mainLayout)
79. def \_\_check\_safe\_position(self, pieces) -> bool:
80. """
81. Проверяет, не угрожают ли фигуры друг другу.
82. :param pieces: Итерация по парам (QLineEdit, QComboBox)
83. :return: True, если позиции безопасны
84. """
85. tempBoard = ChessBoard(self.Parent.boardSizeSpinBox.value())
86. for line\_edit, combo in pieces:
87. x, y = map(int,line\_edit.text().strip().split())
88. type = combo.currentText()
89. piece = Knight(x, y) if type == "Horse" else Vizir(x, y)
90. if not tempBoard.add\_piece(piece):
91. return False
92. return True
93. def accept(self) -> None:
94. """
95. Проверяет валидность позиций и сохраняет их.
96. Если все позиции безопасны — принимает диалог и сохраняет данные в self.resFig.
97. """
98. if self.\_\_check\_safe\_position(zip(self.coordinate\_inputs, self.figure\_selectors)):
100. for line\_edit, combo in zip(self.coordinate\_inputs, self.figure\_selectors):
101. x\_str, y\_str = line\_edit.text().strip().split()
102. self.resFig.append((combo.currentText(), (int(x\_str), int(y\_str))))
103. super().accept()
104. else:
105. QMessageBox.warning(self, "Ошибка", "Фигуры находтся под боем")
106. def create\_input\_fields(self) -> None:
107. """
108. Создаёт поля ввода координат и выпадающие списки для выбора типа фигуры.
109. """
110. for i in range(self.num\_figures):
111. label = QLabel(f"Фигура {i+1}:")
112. line\_edit = QLineEdit()
113. line\_edit.setPlaceholderText("Например: 1 1")
114. line\_edit.setStyleSheet("background-color: lightcoral;")
115. line\_edit.textChanged.connect(self.validate\_all\_inputs)
116. combo = QComboBox()
117. combo.addItems(["Horse", "Vizir"])
118. self.coordinatesLayout.addWidget(label)
119. self.coordinatesLayout.addWidget(line\_edit)
120. self.coordinatesLayout.addWidget(combo)
122. self.coordinate\_inputs.append(line\_edit)
123. self.figure\_selectors.append(combo)
124. def validate\_all\_inputs(self):
125. all\_valid = True
126. for line\_edit in self.coordinate\_inputs:
127. text = line\_edit.text()
128. if self.is\_valid\_format(text):
129. line\_edit.setStyleSheet("background-color: lightgreen;")
130. else:
131. line\_edit.setStyleSheet("background-color: lightcoral;")
132. all\_valid = False
133. self.okButton.setEnabled(all\_valid)
134. def get\_coordinates(self):
135. """Возвращает список введенных координат"""
136. return self.resFig
138. class ModalTable(QDialog):
139. """
140. Диалоговое окно, отображающее одно из решений размещения фигур на шахматной доске.
141. """
142. def \_\_init\_\_(self, table: "ChessBoard", solution: List[Tuple[int, int, str]], Parent=None):
143. """
144. Инициализация диалога, загрузка интерфейса и визуализация переданного решения.
145. :param table: Объект доски (ChessBoard)
146. :param solution: Список кортежей с координатами и типами фигур
147. :param Parent: Родительское окно (обычно MainWindow)
148. """
149. super().\_\_init\_\_(Parent)
150. self.table = table
151. self.solution = solution  # список кортежей: (x, y, 'k' или 'v')
152. self.Parent = Parent
153. self.load\_ui()
154. self.setWindowTitle("Шахматы :: Одно из решений")
155. self.vizualize\_solution()
156. def load\_ui(self):
157. """
158. Загружает интерфейс из файла .ui и находит нужные элементы.
159. """
160. loader = QUiLoader()
161. file = QFile(r"ui/board\_window.ui")
162. if not file.open(QFile.ReadOnly):
163. QMessageBox.critical(self, "Ошибка", "Не удалось открыть файл UI!")
164. return
165. self.ui = loader.load(file, self)
166. file.close()
167. # Основной layout
168. self.mainLayout = self.ui.findChild(QVBoxLayout, "verticalLayout")
169. # Виджет для доски
170. self.boardWidget = self.ui.findChild(QWidget, "boardWidget")
171. # Layout для кнопок
172. self.button\_layout = self.ui.findChild(QHBoxLayout, "buttonLayout")
173. self.saveButton = self.ui.findChild(QPushButton, "saveButton")
174. self.exitButton = self.ui.findChild(QPushButton, "exitButton")
175. self.saveButton.clicked.connect(self.save\_to\_file)
176. self.exitButton.clicked.connect(self.close)
178. self.setLayout(self.mainLayout)
179. def vizualize\_solution(self):
180. """
181. Визуализирует текущее решение на графической сцене с использованием QGraphicsView.
182. """
183. size = len(self.table.grid)
184. scene = QGraphicsScene()
185. view = QGraphicsView()
186. view.setScene(scene)
187. view.setRenderHint(QPainter.Antialiasing)
188. layout = QVBoxLayout(self.boardWidget)
189. layout.addWidget(view)
190. view.setSceneRect(-1,-1, size \* Cell.CELL\_SIZE+2, size \* Cell.CELL\_SIZE+2 )
191. scene.setSceneRect(view.sceneRect())
192. # Добавляем фигуры, которые проинициализированы пользователем
193. for x,y,type in self.solution:
194. if self.table.grid[y][x].is\_empty():
195. piece = Knight(x,y) if type == "k" else Vizir(x,y)
196. self.table.add\_piece(piece)
197. # Визуализация всех ячеек доски
198. for x in range(size):
199. for y in range(size):
200. if not self.table.grid[y][x].is\_empty() and self.table.grid[y][x].piece.\_initiatedByUser:
201. cell = Cell(x,y,2) # Пользовательская фигура
202. elif self.table.grid[y][x].threatened:
203. cell = Cell(x,y,3) # Под угрозой
204. elif self.table.grid[y][x].is\_empty():
205. cell = Cell(x,y,0) # Пустая клетка
206. else:
207. cell = Cell(x,y,1) # Размещенная программой фигура
208. scene.addItem(cell)
209. view.setFixedSize(view.sizeHint())
210. def save\_to\_file(self):
211. """
212. Сохраняет решения в файл через отдельный поток SaveThread.
213. """
214. try:
215. self.save\_thread = SaveThread(self.Parent.solver, self.Parent.solver.solutions, "output.txt")
216. self.save\_thread.finished.connect(self.on\_save\_finished)
217. self.save\_thread.start()
219. except Exception as e:
220. QMessageBox.critical(self, "Ошибка", f"Не удалось сохранить файл: {e}")
221. # Slot
222. def on\_save\_finished(self, filename: str):
223. """
224. Обработчик завершения сохранения файла. Показывает сообщение пользователю.
225. :param filename: Имя файла, в который были сохранены данные
226. """
227. print(len(self.Parent.solver.solutions))
228. QMessageBox.information(
229. self,
230. "Успех",
231. f"Решения сохранены в {filename}"
232. )
234. class SaveThread(QThread):
235. """
236. Поток для асинхронного сохранения решений в файл.
237. Сигналы:
238. finished (str, str): Сигнал, испускаемый после завершения работы.
239. Первый параметр — имя файла.
240. Второй параметр — сообщение об ошибке (пустая строка, если ошибок не было).
241. """
242. finished = Signal(str, str)
244. def \_\_init\_\_(self, solver, solutions: Set[frozenset], filename: str):
245. """
246. Инициализирует поток сохранения.
247. :param solver: Объект Solver, содержащий метод записи в файл.
248. :param solutions: Множество решений для сохранения.
249. :param filename: Имя файла, в который нужно сохранить данные.
250. """
251. super().\_\_init\_\_()
252. self.solver = solver
253. self.solutions = solutions
254. self.filename = filename
256. def run(self):
257. """
258. Выполняет сохранение решений в файл.
259. При успешном завершении отправляет сигнал без ошибки.
260. В случае исключения — сигнал с сообщением об ошибке.
261. """
262. try:
263. self.solver.write\_output(self.filename, self.solutions)
264. self.finished.emit(self.filename, "")
265. except Exception as e:
266. self.finished.emit(self.filename, str(e))
267. class Cell(QGraphicsRectItem):
268. """
269. Класс для отображения одной ячейки шахматного поля в QGraphicsScene.
271. Наследуется от QGraphicsRectItem и отвечает за визуализацию ячейки на сцене.
272. Цвет зависит от значения, переданного при создании.
273. """
274. CELL\_SIZE = 32 # Размер одной клетки в пикселях
275. # Назначим цвет в зависимости от значения в матрице
276. COLOR\_MAP = {
277. 0: QColor("white"),
278. 1: QColor("green"),
279. 2: QColor("blue"),
280. 3: QColor("red")
281. }
282. def \_\_init\_\_(self, x: int, y: int, value: int):
283. """
284. Инициализирует ячейку на заданных координатах и с заданным значением цвета.
285. :param x (int): Координата по горизонтали (в клетках).
286. :param y (int): Координата по вертикали (в клетках).
287. :param value (int): Значение клетки, определяющее цвет заливки (0–3).
288. """
289. super().\_\_init\_\_(0, 0, self.CELL\_SIZE, self.CELL\_SIZE)
290. self.setPos(x \* self.CELL\_SIZE, y \* self.CELL\_SIZE)
291. self.setPen(QPen(QColor("black")))
293. self.setBrush(self.COLOR\_MAP.get(value, QColor("white")))
294. class MainWindow(QMainWindow):
295. """
296. Главное окно приложения, управляющее взаимодействием пользователя с UI и логикой решения задачи.
297. """
298. def \_\_init\_\_(self):
299. """
300. Инициализация главного окна: загрузка UI, настройка кнопок, установка начального состояния.
301. """
302. super().\_\_init\_\_()
303. self.load\_ui()
304. self.board\_created = False
305. self.coords = []
306. # Инициализация кнопок
307. self.createBoardButton.clicked.connect(self.create\_board)
308. self.drawBoardButton.clicked.connect(self.draw\_board)
309. self.exitButton.clicked.connect(self.close)
311. # Изначально кнопка "Нарисовать доску" неактивна
312. self.drawBoardButton.setEnabled(False)
313. self.solver = Solver()
314. # self.thread\_pool = QThreadPool()

317. def load\_ui(self):
318. """
319. Загружает интерфейс из .ui-файла и инициализирует элементы управления.
320. """
321. loader = QUiLoader()
322. file = QFile(r"ui/chess\_ui.ui")
323. file.open(QFile.ReadOnly)
324. self.ui = loader.load(file, self)
325. file.close()
327. # Получаем доступ к элементам UI
328. self.createBoardButton = self.ui.findChild(QPushButton, "createBoardButton")
329. self.drawBoardButton = self.ui.findChild(QPushButton, "drawBoardButton")
330. self.exitButton = self.ui.findChild(QPushButton, "exitButton")
331. self.boardSizeSpinBox = self.ui.findChild(QSpinBox, "boardSizeSpinBox")
332. self.requiredFiguresSpinBox = self.ui.findChild(QSpinBox, "requiredFiguresSpinBox")
333. self.placedFiguresSpinBox = self.ui.findChild(QSpinBox, "placedFiguresSpinBox")
334. # self.figureComboBox = self.ui.findChild(QComboBox, "figureComboBox")
336. # Установка центрального виджета
337. layout = QVBoxLayout()
338. layout.addWidget(self.ui)
339. central\_widget = QWidget()
340. central\_widget.setLayout(layout)
341. self.setCentralWidget(central\_widget)
342. def get\_n\_l\_k\_pieces(self) -> Tuple[int, int, int, List[Tuple[str, Tuple[int, int]]]]:
343. """
344. Возвращает значения с элементов управления: размер доски, число фигур для размещения,
345. количество уже размещенных фигур и их координаты.
346. :return: Кортеж из (n, l, k, координаты фигур)
347. """
348. return self.boardSizeSpinBox.value(), self.requiredFiguresSpinBox.value(), self.placedFiguresSpinBox.value(), self.coords
349. def create\_board(self):
350. """
351. Создает доску и запрашивает координаты уже размещенных фигур,
352. если они есть. Делает кнопку рисования доступной.
353. """
354. try:
355. k = self.placedFiguresSpinBox.value()
356. if k > 0:
357. coords\_menu = InputCoordinatesDialog(k, self)
358. if coords\_menu.exec() == QDialog.Accepted:
359. self.coords = coords\_menu.get\_coordinates()
360. # Если доска успешно создана
361. self.board\_created = True
362. self.drawBoardButton.setEnabled(True)
364. except Exception as e:
365. QMessageBox.critical(self, "Ошибка", f"Не удалось создать доску: {str(e)}")
366. self.drawBoardButton.setEnabled(False)
368. def draw\_board(self):
369. """
370. Отрисовывает доску и запускает поток для поиска решений, если доска уже создана.
371. """
372. if self.board\_created:
373. board, combinations = self.solver.generateBoard(\*self.get\_n\_l\_k\_pieces())
375. self.solver\_thread = SolverThread(self.solver, board, combinations)
376. self.solver\_thread.finished.connect(self.on\_solver\_finished)
377. self.solver\_thread.start()
379. # self.solver.solve(board, combinations)
380. # ModalTable(board, next(iter(self.solver.solutions)), Parent = self).exec()
381. # QMessageBox.information(self, "Информация", "Доска отрисована")
382. else:
383. QMessageBox.warning(self, "Ошибка", "Сначала создайте доску")
384. # Slot
385. def on\_solver\_finished(self, board, solutions):
386. """
387. Обрабатывает завершение потока решения и отображает найденное решение.
388. :param board: Объект доски.
389. :param solutions: Найденные решения в виде множества.
390. """
391. if solutions:
392. ModalTable(board, next(iter(solutions)), self).exec()
393. def closeEvent(self, event):
394. """
395. Перехватывает закрытие окна и завершает все работающие потоки корректно.
396. :param event: Событие закрытия окна.
397. """
398. threads = [
399. getattr(self, attr) for attr in dir(self)
400. if attr.endswith('\_thread') and isinstance(getattr(self, attr), QThread)
401. ]
403. for thread in threads:
404. if thread.isRunning():
405. thread.quit()
406. thread.wait(500)  # Даем 500мс на завершение
408. event.accept()
410. class Piece(ABC):
411. """
412. Базовый класс шахматной фигуры.
413. """
414. def \_\_init\_\_(self, x: int, y: int, initiated = False):
415. """
416. Инициализация фигуры на позиции (x, y).
417. :param x: Координата X на доске.
418. :param y: Координата Y на доске.
419. """
420. self.x = x
421. self.y = y
422. self.\_initiatedByUser = initiated
424. @abstractmethod
425. def get\_moves(self) -> List[Tuple[int, int]]:
426. """
427. Метод, который должен быть реализован в подклассах.
428. Возвращает множество клеток, находящихся под ударом фигуры.
429. :return: Список координат клеток, находящихся под угрозой.
430. """
431. pass
433. @property
434. @abstractmethod
435. def symbol(self) -> str:
436. """:return: Символьное обозначение фигуры"""
437. pass
438. class Knight(Piece):
439. """
440. Класс шахматной фигуры — Конь.
441. """
442. moves = [(-2, -1), (-2, 1), (2, -1), (2, 1),
443. (-1, -2), (-1, 2), (1, -2), (1, 2)]
444. def get\_moves(self):
445. return [(-2, -1), (-2, 1), (2, -1), (2, 1),
446. (-1, -2), (-1, 2), (1, -2), (1, 2)]
448. @property
449. def symbol(self):
450. return 'k'
451. class Vizir(Piece):
452. """
453. Класс шахматной фигуры — Визирь.
454. """
455. moves = [(-1, 0), (1, 0), (0, -1), (0, 1), (-2, 0), (2, 0), (0, -2), (0, 2)]
456. def get\_moves(self):
457. return [(-1, 0), (1, 0), (0, -1), (0, 1), (-2, 0), (2, 0), (0, -2), (0, 2)]
459. @property
460. def symbol(self):
461. return 'v'
462. class BoardCell:
463. """
464. Класс, представляющий одну ячейку шахматной доски.
465. Атрибуты:
466. :param x (int): Координата X (столбец).
467. :param y (int): Координата Y (строка).
468. :param piece (Optional[Piece]): Фигура, размещённая в ячейке, если есть.
469. :param threatened (bool): Флаг, указывающий, находится ли ячейка под угрозой.
470. """
471. def \_\_init\_\_(self, x: int, y: int):
472. self.x = x
473. self.y = y
474. self.piece: Optional[Piece] = None
475. self.threatened = False

478. def place\_piece(self, piece: Piece):
479. self.piece = piece
481. def remove\_piece(self):
482. self.piece = None
484. def is\_empty(self) -> bool:
485. return self.piece is None
487. def \_\_str\_\_(self):
488. return self.piece.symbol if self.piece else '\*' if self.threatened else '0'
489. class ChessBoard:
490. """
491. Класс, представляющий шахматную доску и методы размещения фигур.
492. """
493. def \_\_init\_\_(self, size: int):
494. """
495. Инициализация пустой доски.
496. :param size: Размер доски.
497. """
498. self.size = size
499. self.grid = [[BoardCell(x, y) for x in range(size)] for y in range(size)]
500. self.pieces: List[Piece] = []
502. def add\_piece(self, piece: Piece) -> bool:
503. """
504. Размещает фигуру на доске и обновляет угрозы.
505. :param piece: Экземпляр фигуры.
506. """
507. cell = self.grid[piece.y][piece.x]
509. if not cell.is\_empty() or not self.is\_position\_safe(piece):
510. return False
512. cell.place\_piece(piece)
513. self.pieces.append(piece)
514. self.update\_threats()
515. return True
517. def remove\_piece(self, piece: Piece):
518. """
519. Удаляет фигуру с доски и пересчитывает угрозы.
520. :param piece: Экземпляр фигуры, которую нужно убрать.
521. """
522. cell = self.grid[piece.y][piece.x]
523. if cell.piece == piece:
524. cell.remove\_piece()
525. self.pieces.remove(piece)
526. self.update\_threats()
528. def is\_position\_safe(self, piece: Piece) -> bool:
529. """
530. Проверяет безопасность позиции для фигуры.
531. :param piece: Экземпляр фигуры.
532. """
533. # Проверка выхода за границы
534. if not (0 <= piece.x < self.size and 0 <= piece.y < self.size):
535. return False
537. # Проверка занятости клетки
538. if not self.grid[piece.y][piece.x].is\_empty():
539. return False
540. # Проверка атак от фигуры
541. for dx, dy in piece.get\_moves():
542. nx, ny = piece.x + dx, piece.y + dy
543. if 0 <= nx < self.size and 0 <= ny < self.size:
544. if not self.grid[ny][nx].is\_empty() and self.grid[ny][nx].piece.symbol in ('k', 'v'):
545. return False
546. # Проверка атак от других фигур
547. for other in self.pieces:
548. for dx, dy in other.get\_moves():
549. if other.x + dx == piece.x and other.y + dy == piece.y:
550. return False
552. return True
554. def update\_threats(self):
555. """Обновляет состояние угроз на доске"""
556. # Сброс всех угроз
557. for row in self.grid:
558. for cell in row:
559. cell.threatened = False
561. # Обновление угроз
562. for piece in self.pieces:
563. for dx, dy in piece.get\_moves():
564. nx, ny = piece.x + dx, piece.y + dy
565. if 0 <= nx < self.size and 0 <= ny < self.size:
566. self.grid[ny][nx].threatened = True
568. def get\_solution(self) -> List[Tuple[int, int, str]]:
569. """Возвращает текущее решение"""
570. return sorted([(p.x, p.y, p.symbol) for p in self.pieces],
571. key=lambda item: (item[1], item[0]))
573. def visualize(self):
574. """Визуализирует доску"""
575. for row in self.grid:
576. print(' '.join(str(cell) for cell in row))
577. print()
578. class Solver:
579. """
580. Класс, реализующий решение задачи размещения фигур на шахматной доске
581. с использованием метода полного перебора (backtracking).
582. """
583. def \_\_init\_\_(self):
584. """
585. Инициализация хранилища решений.
586. """
587. self.solutions: Set[frozenset] = set()
588. def solve(self, board: ChessBoard, combinations: List[List[str]]) -> None:
589. """
590. Находит все допустимые расстановки фигур, не угрожающих друг другу.
591. :param board: Объект шахматной доски с уже размещёнными фигурами.
592. :param combinations: Список всех перестановок фигур для размещения, например ['k', 'v', 'v'].
593. """
594. def backtrack(remaining\_pieces: List[str]):
595. """
596. Рекурсивный алгоритм поиска с возвратом.
597. :param remaining\_pieces: Список оставшихся фигур для размещения.
598. """
599. if not remaining\_pieces:
600. self.solutions.add(frozenset(board.get\_solution()))
601. return
603. piece\_type = remaining\_pieces[0]
604. for y in range(board.size):
605. for x in range(board.size):
606. piece = Knight(x, y) if piece\_type == 'k' else Vizir(x, y)
607. if board.add\_piece(piece):
608. backtrack(remaining\_pieces[1:])
609. board.remove\_piece(piece)
611. for combination in combinations:
612. backtrack(combination)
613. @staticmethod
614. def generateBoard(n, l, k, pieces) -> Tuple[int, int, int, int, ChessBoard, List[str]]:
615. """
616. Генерирует шахматную доску и комбинации фигур для размещения.
617. :param n: Размер доски (n x n).
618. :param l: Количество новых фигур, которые нужно разместить.
619. :param k: Количество уже размещённых фигур.
620. :param pieces: Список уже размещённых фигур в формате (тип, (x, y)),
621. где тип — 'Horse' или 'Vizir'.
622. :return: Кортеж, содержащий:
623. - объект доски (ChessBoard) с размещёнными фигурами,
624. - множество всех возможных комбинаций из L фигур ('k' и 'v') для размещения.
625. """
626. board = ChessBoard(n)
627. if k > 0:
628. for i in range(k):
629. type, coords = pieces[i]
630. piece = Knight(\*coords, initiated = True) if type == "Horse" else Vizir(\*coords, initiated = True)
631. board.add\_piece(piece)
632. # Генерация комбинаций фигур для размещения
633. combinations = set(combinations\_with\_replacement('kv', l))
634. return board, combinations
636. @staticmethod
637. def write\_output(file\_path: str, solutions: Set[frozenset]):
638. """
639. Записывает все найденные решения в выходной файл.
640. :param file\_path: Путь к выходному файлу.
641. :param solutions: Множество решений, каждое из которых представлено как frozenset
642. кортежей вида (x, y, тип).
643. """
644. with open(file\_path, 'w') as file:
645. if solutions:
646. for solution in solutions:
647. solution\_str = " ".join(f"({x}, {y}, {f})" for x, y, f in solution)
648. file.write(solution\_str + "\n")
649. else:
650. file.write("no solutions\n")
651. class SolverThread(QThread):
652. """
653. Поток для выполнения поиска решений в фоновом режиме, не блокируя GUI.
654. По завершении испускает сигнал с результатами.
655. """
656. finished = Signal(object,object)
657. # Сигнал, испускаемый по завершении работы потока.
658. # Передаёт: доску и найденные решения.
659. def \_\_init\_\_(self, solver: Solver, board: ChessBoard, combinations:List[List[str]]) -> None:
660. """
661. Инициализация потока с нужными параметрами.
662. :param solver: Объект класса Solver, реализующий логику поиска решений.
663. :param board: Объект шахматной доски (ChessBoard).
664. :param combinations: Список комбинаций фигур для размещения (например: [['k', 'v'], ['v', 'v']] и т.д.).
665. """
666. super().\_\_init\_\_()
667. self.solver = solver
668. self.board = board
669. self.combinations = combinations
671. def run(self) -> None:
672. """
673. Запускает алгоритм поиска решений в отдельном потоке.
674. По завершении испускает сигнал с доской и найденными решениями.
675. """
676. self.solver.solve(self.board, self.combinations)
677. self.finished.emit(self.board, self.solver.solutions)
679. if \_\_name\_\_ == "\_\_main\_\_":
680. app = QApplication([])
681. window = MainWindow()
682. window.show()
683. app.exec()

# ЗАКЛЮЧЕНИЕ

В ходе работы мы написали программу, которая приводит в работу приложение, основанное на коде с шахматами из прошлого семестра. Кроме того, в ходе работы была изучена библиотека PySide6, которая помогла написать программу GUI приложения для шахмат.

**Список использованных источников**

1. Робертс А. "Искусство программирования. Теория и практика". – М.: Диалектика, 2019.
2. Дасгупта С., Пападимитриу Х., Увазани У. "Алгоритмы. Построение и анализ". – М.: Вильямс, 2014.
3. Knuth D. E. "The Art of Computer Programming", Volumes 1-3. – Addison-Wesley, 1997.
4. Официальная документация Python: https://docs.python.org/3