

Министерство науки и высшего образования Российской Федерации

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
ITMO University**

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

По дисциплине Программирование

Тема работы Реализация приложения для игры в шахматы

Обучающийся Савальский Матвей Иванович

Факультет факультет инфокоммуникационных технологий

Группа К3120

Направление подготовки 11.03.02 Инфокоммуникационные технологии и
системы связи

Образовательная программа Программирование в
инфокоммуникационных системах

Обучающийся

(дата)

(подпись)

Савальский М.И.
(Ф.И.О.)

Руководитель

(дата)

(подпись)

Казанова П.П.
(Ф.И.О.)

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ	3
Предварительный анализ.....	4
1 Основная часть	5
1.1 Классы и их основные функции.....	5
1.1.1 GameMixin.....	5
1.1.2 Game	10
1.1.3 Player.....	12
1.1.4 Figure.....	12
1.1.5 Terminal.....	13
1.1.6 ChessApp.....	17
1.1.7 StartProgramm	23
ЗАКЛЮЧЕНИЕ	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	25

ВВЕДЕНИЕ

В данном документе представлен отчет о выполнении домашнего задания по дисциплине Программирование за 2 семестр.

Задание: Создать программное обеспечение системы обработки данных: «Программа для игры в шахматы», обладающая GUI (написанным в ООП стиле). Программа должна быть устойчива к ошибкам, а также соответствовать своей предметной области.

ПРЕДВАРИТЕЛЬНЫЙ АНАЛИЗ

В данном разделе представлен анализ предметной области и требований к разрабатываемой системе.

Поскольку система нацелена на рядового пользователя, она должна иметь интуитивно-понятный интерфейс и защиту от ввода данных неправильного формата. Данные будут сохраняться в БД в папке, в которой находится исполняемый код приложения в расширении .py. Приложение исполняется в терминале компьютера с установленным на него языком Python версии 3.10 и выше и наличием соответствующих библиотек.

Для начала необходимо разобраться со структурой классов в ООП приложении, ниже представлена концептуальная диаграмма классов(сущностей) данного приложения с пояснениями (Рисунок 1).

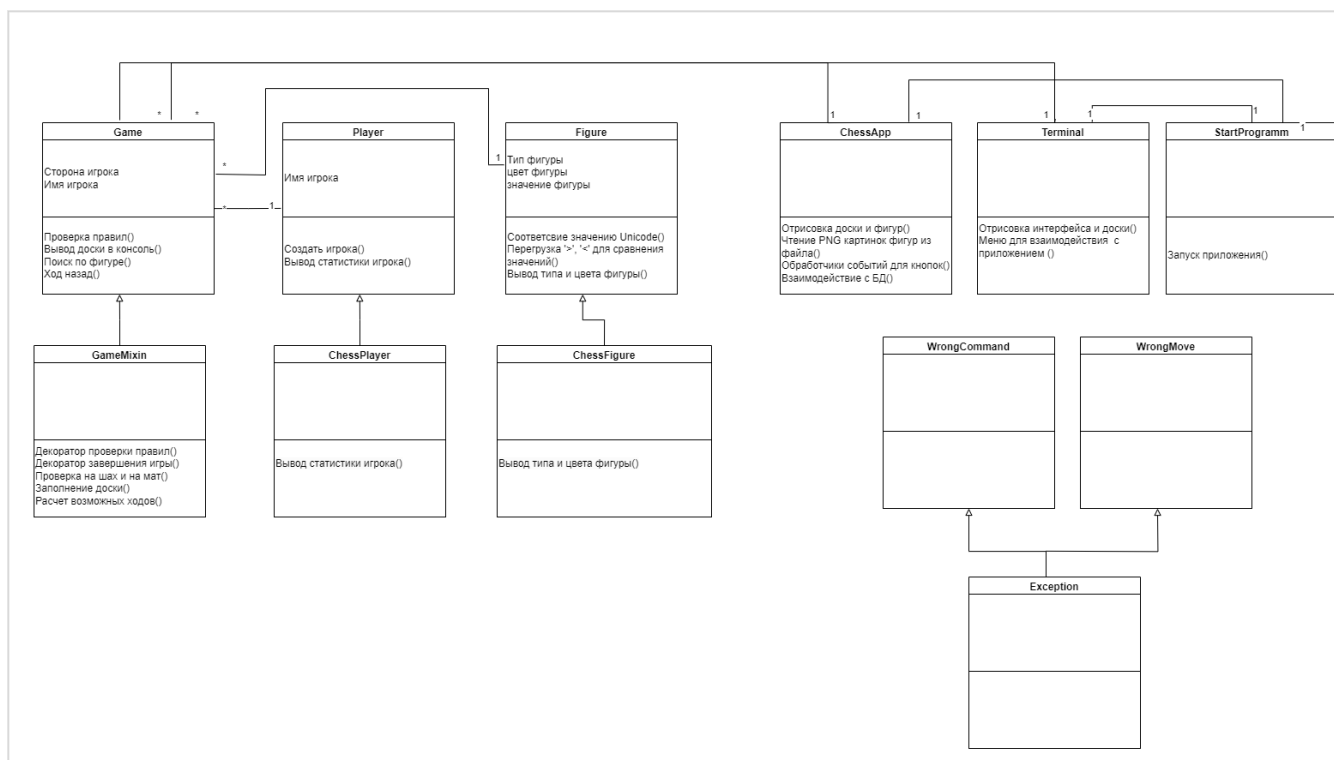


Рисунок 1 — Диаграмма классов

Пояснения:

Классы WrongCommand и WrongMove - классы исключения, для соответствующих ошибок при вводе пользователя.

1 Основная часть

1.1 Классы и их основные функции

В данной главе приведены практические реализации важных методов классов в диаграмме классов на рисунке 1.

1.1.1 GameMixin

Начнем с класса GameMixin, который собирает в себе основные методы для построения процесса игры в шахматы, такие как заполнение доски, проверка на шах или мат, а также правила хода разных фигур в зависимости от их позиции, цвета и типа. Рассмотрим его функцию заполнения доски(Рисунок 1.1).

```
132
133 def set_all_figures(self, reverse = False):
134     colour = ['white', 'black']
135     if reverse:
136         colour = colour[::-1]
137     for t in [6,1]:
138         for i in range(8):
139             if t == 6:
140                 self.board[t][i] = Figure('pawn', colour[1])
141             else:
142                 self.board[t][i] = Figure('pawn', colour[0])
143
144     for t in [0,7]:
145         for i in range(0,5):
146             if t == 0:
147                 if i == 0:
148                     self.board[t][i] = Figure('rook', colour[0])
149                     self.board[t][7-i] = Figure('rook', colour[0])
150                 elif i == 1:
151                     self.board[t][i] = Figure('knight', colour[0])
152                     self.board[t][7-i] = Figure('knight', colour[0])
153                 elif i == 2:
154                     self.board[t][i] = Figure('bishop', colour[0])
155                     self.board[t][7-i] = Figure('bishop', colour[0])
156                 elif i == 3:
157                     self.board[t][i] = Figure('queen', colour[0])
158                     self.board[t][7-i] = Figure('king', colour[0])
159             else:
160                 if i == 0:
161                     self.board[t][i] = Figure('rook', colour[1])
162                     self.board[t][7-i] = Figure('rook', colour[1])
163                 elif i == 1:
164                     self.board[t][i] = Figure('knight', colour[1])
165                     self.board[t][7-i] = Figure('knight', colour[1])
166                 elif i == 2:
167                     self.board[t][i] = Figure('bishop', colour[1])
168                     self.board[t][7-i] = Figure('bishop', colour[1])
169                 elif i == 3:
170                     self.board[t][i] = Figure('queen', colour[1])
171                     self.board[t][7-i] = Figure('king', colour[1])
172                 if colour[1] != 'white':
173                     self.board[t][i], self.board[t][7-i] = self.board[t][7-i], self.board[t][i]
174                     self.board[7-t][i], self.board[7-t][7-i] = self.board[7-t][7-i], self.board[7-t][i]
175
176     return
```

Рисунок 1.1 — Функция заполнения доски

Пояснения: board - атрибут объекта класса Game, он представляет из себя двумерный массив 8 на 8, заполненный либо элементом класса Figure() отвечающим за тип и цвет фигуры, или '_', символизирующем о том, что на данной клетке не находится никакая фигура.

Далее перейдем к функции просчета всевозможных ходов для некоторой фигуры на поле (Рисунки 1.2,1.3,1.4).

```

177
178 def possible_moves(self, pos, qtype = ''):
179     if qtype == '':
180         type = self.board[pos[0]][pos[1]].type
181     else:
182         type = qtype
183     color = self.board[pos[0]][pos[1]].color
184     av_moves = []
185     av_moves_c = []
186     try:
187         match type:
188             case 'pawn':
189                 if color == 'white' and self.side == 'w' or color == 'black' and self.side == 'b':
190                     d = 1
191                 else:
192                     d = -1
193                 if pos[0] == 1 or pos[0] == 6:
194                     step = 2
195                 else:
196                     step = 1
197
198                 if self.get_figure([pos[0] - 1*d, pos[1]]) == '_':
199                     av_moves.append([pos[0] - 1*d, pos[1]])
200                 if self.get_figure([pos[0] - step*d, pos[1]]) == '_' and step == 2:
201                     av_moves.append([pos[0] - 2*d, pos[1]])
202
203                 if -1 < pos[1] - 1 < 7:
204                     if self.get_figure([pos[0]-1*d, pos[1] - 1]) != '_' and self.get_figure([pos[0]-1*d, pos[1] - 1]).color != color :
205                         av_moves.append([pos[0]-1*d, pos[1] - 1])
206                 if -1 < pos[1] + 1 < 7:
207                     if self.get_figure([pos[0]-1*d, pos[1] + 1]) != '_' and self.get_figure([pos[0]-1*d, pos[1] + 1]).color != color :
208                         av_moves.append([pos[0]-1*d, pos[1] + 1])
209
210             case 'knight':
211                 for s in [-1,1]:
212                     for i in [1,2]:
213                         if 0 <= pos[0] + s*i <= 7 and 0 <= pos[1] + s*(3-i) <= 7:
214                             if self.get_figure([pos[0] + s*i, pos[1] + s*(3-i)]) == '_' or self.get_figure([pos[0] + s*i, pos[1] + s*(3-i)]).color != color :
215                                 av_moves.append([pos[0] + s*i, pos[1] + s*(3-i)])
216                         if 0 <= pos[0] + s*i <= 7 and 0 <= pos[1] - s*(3-i) <= 7:
217                             if self.get_figure([pos[0] + s*i, pos[1] - s*(3-i)]) == '_' or self.get_figure([pos[0] + s*i, pos[1] - s*(3-i)]).color != color :
218                                 av_moves.append([pos[0] + s*i, pos[1] - s*(3-i)])
219

```

Рисунок 1.2 — Функция расчета ходов (1)

```

218         av_moves.append([pos[0] + s*i, pos[1] - s*(3-i)])
219
220     case 'rook':
221         for d in [1,-1]:
222             x, y = pos[0], pos[1]
223             while 0 < x < 8 and d == -1 or -1 < x < 7 and d == 1:
224                 x = x + d
225                 if self.get_figure([x,pos[1]]) == '_':
226                     av_moves.append([x,pos[1]])
227             else:
228                 if self.get_figure([x,pos[1]]).color != color:
229                     av_moves.append([x,pos[1]])
230             break
231
232             while 0 < y < 8 and d == -1 or -1 < y < 7 and d == 1:
233                 y = y + d
234                 if self.get_figure([pos[0],y]) == '_':
235                     av_moves.append([pos[0],y])
236             else:
237                 if self.get_figure([pos[0],y]).color != color:
238                     av_moves.append([pos[0],y])
239             break
240
241     case 'bishop':
242         x, y = pos[0], pos[1]
243         for j in [-1,1]:
244             flag1, flag2 = 1, 1
245             for i in range(1,8):
246                 if flag1:
247                     if -1 < (x + j*i) < 8 and -1 < (y + j*i) < 8:
248                         if self.get_figure([x + j*i, y + j*i]) == '_':
249                             av_moves.append([x + j*i, y + j*i])
250                     else:
251                         if self.get_figure([x + j*i, y + j*i]).color != color:
252                             av_moves.append([x + j*i, y + j*i])
253                         flag1 = 0
254                 if flag2:
255                     if -1 < (x - j*i) < 8 and -1 < (y + j*i) < 8:
256                         if self.get_figure([x - j*i, y + j*i]) == '_':
257                             av_moves.append([x - j*i, y + j*i])
258                     else:
259                         if self.get_figure([x - j*i, y + j*i]).color != color:
260                             av_moves.append([x - j*i, y + j*i])
261                         flag2 = 0

```

Рисунок 1.3 — Функция расчета ходов (2)

```

262
263     case 'king':
264         x, y = pos[0], pos[1]
265         for i in [-1,0,1]:
266             for j in [-1,0,1]:
267                 if i == j == 0:
268                     continue
269                 if -1 < (x+i) < 8 and -1 < (y+j) < 8:
270                     if self.get_figure([x+i,y+j]) == '_' or self.get_figure([x+i,y+j]).color != color:
271                         av_moves.append([x+i,y+j])
272
273     case 'queen':
274         av_moves += (self.possible_moves(pos,qtype = 'bishop'))
275         av_moves += (self.possible_moves(pos,qtype = 'rook'))
276
277     except Exception as e:
278         print('possible_moves_error:',e)
279         return []
280
281     for i in av_moves:
282         if 8 > i[0] > -1 and 8 > i[1] > -1:
283             av_moves_c.append(i)
284     return av_moves_c
285

```

Рисунок 1.4 — Функция расчета ходов (3)

Пояснения: В данном классе происходит обыкновенный switch case сценарий в зависимости от типа фигуры и её цвета. На выходе функция выдает массив из возможных клеток, куда можно поставить ту или иную фигуру, в определенный момент игры(т.е. в зависимости от того как стоят другие фигуры).

Далее рассмотрим функцию `is_check`, которая отвечает за проверку, того, стоит ли шах королю некоторой стороны (Рисунок 1.5). Суть её работы: пройтись по всем вражеским фигурам и определить, возможен ли ход на клетку короля, если да, то присутствует шах. Функция использует многопоточность, таким образом, фигуры, которые могут поставить шах королю разбиваются на две группы и проверяются в отдельных потоках.

```
93
94
95     def is_check(self,pos):
96         try:
97             lock = threading.Lock()
98             color = self.get_figure(pos).color
99             self.all_moves = []
100             def get_all_moves(n,m):
101                 for i in range(8):
102                     for j in range(n,m):
103                         if self.board[i][j] != '_' and self.board[i][j].color != color:
104                             with lock:
105                                 self.all_moves += self.possible_moves([i,j])
106             return
107
108             threads = [Thread(target=get_all_moves,args = i) for i in [(0,4), (4,8)]]
109             for thread in threads:
110                 thread.start()
111             for thread in threads:
112                 thread.join()
113
114             if pos in self.all_moves:
115                 return 1 #check
116             else:
117                 return 0 #no check
118
```

Рисунок 1.5 — Функция `is_check`

А также тесно связанную с ней и дополняющую её функцию `checkmate`, которая проверяет является ли поставленный шах матом (Рисунок 1.6). Её суть заключается в проверке всевозможных ходов стороны, которая находится под шахом, и если любой ход не меняет того, что король под шахом, то это значит, что был поставлен мат. Также принцип есть на диаграмме (Рисунок 1.7).


```

64
65 # Before every move if check to the 'white' king
66 # DO:
67 # 1. For all figures('white') one by one make every possible move and check the position with is_check again.
68 # 2. If all is_checks are positive then its mate to the 'white' side
69 # 3. If one of the checks is negative, then it's just a check to the 'white' side
70 def checkmate(self, pos): # Pos is position to the side that is BEING CHECKMATED
71     current_board = Game.c_board(self.board)
72     color = self.get_figure(pos).color
73
74     for x in range(8):
75         for y in range(8):
76             if self.board[x][y] != '_' and self.board[x][y].color == color:
77                 f_moves = self.possible_moves([x,y])
78                 for move in f_moves:
79                     self.board[x][y], self.board[move[0]][move[1]] = '_', self.board[x][y]
80                     if not self.is_check(self.get_position('king',color)[0]):
81                         self.board = GameMixin.copy_board(current_board)
82                         return False #No checkmate
83                     self.board = GameMixin.copy_board(current_board)
84     return True # Checkmate

```

Рисунок 1.6 — Функция checkmate

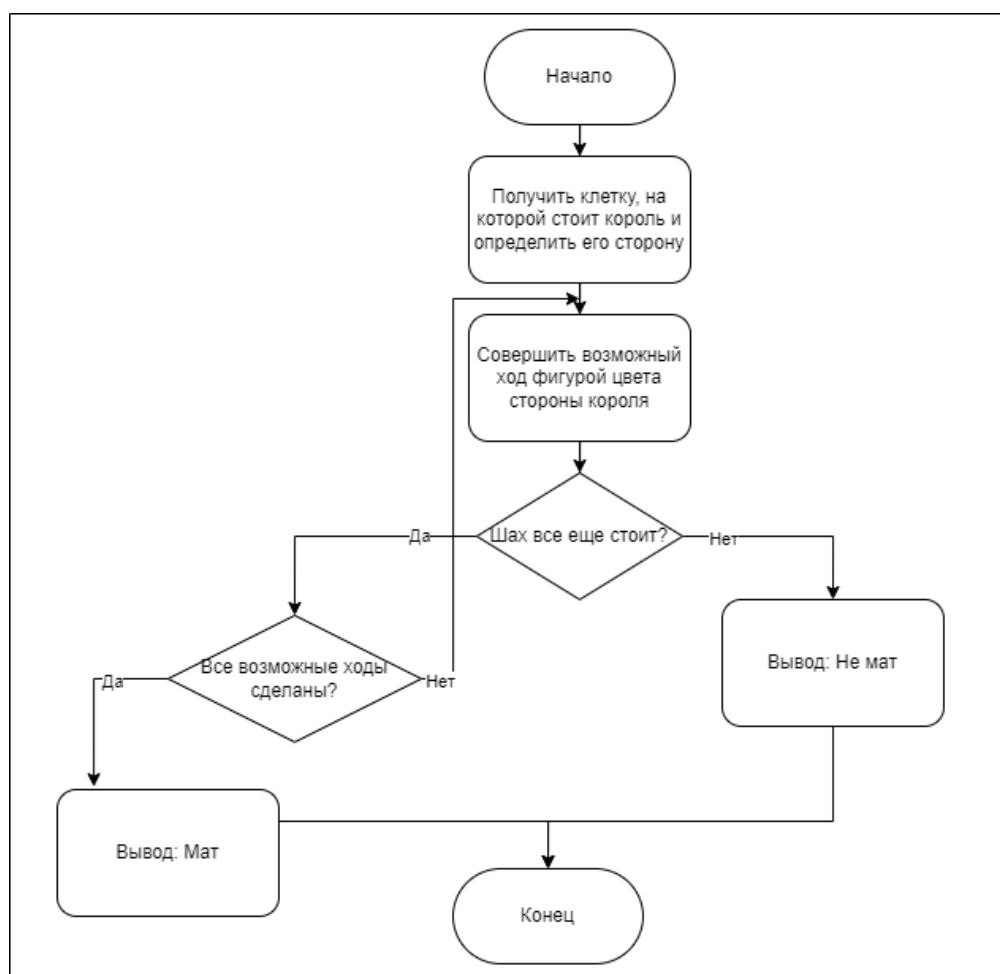


Рисунок 1.7 — Функция checkmate, диаграмма

1.1.2 Game

Класс Game наследуется от GameMixin и окончательно определяет правила и процесс игры в шахматы своими методами. Рассмотрим его атрибуты (Рисунок 1.8).

```
304
305 class Game(GameMixin):
306     def __init__(self, side='w', player=Player('Default')):
307         self.board = [['_'] * 8 for _ in range(8)]
308         self.time_start = time.time()
309         self.move_num = 0
310         self.side = side
311         self.player = player
312         self.history = []
313         self.g_history = []
314
```

Рисунок 1.8 — Атрибуты класса Game

При создании объекта класса Game обязательно должна быть выбрана сторона игрока(side) и его имя(player). Также с течением игры ведется учет времени, прошедшей с начала(time_start) и число сделанных ходов(move_num), для чередования ходов двух сторон. Также добавляется массив history для учета всех состояний доски, чтобы если что, можно было откатиться на несколько шагов назад (это допускается, так как приложение не имеет мультиплеера и соответственно, эта функция выступает как помощник в анализе партий), а также g_history, статистику по партиям и после загружающая ее на БД.

Рассмотрим его 2 основные функции rules (Рисунок 1.9) и move (Рисунок 1.10), т.е. функции проверки легитимности хода игрока, а также сам ход.

```

314
315     def rules(self, pos1, pos2):
316         try:
317             if self.board[pos1[0]][pos1[1]] == '_':
318                 raise WrongMove('Moving an empty spot')
319             if (self.move_num % 2 == 0 and self.board[pos1[0]][pos1[1]].color == 'black' or
320                 self.move_num % 2 == 1 and self.board[pos1[0]][pos1[1]].color == 'white'):
321                 raise WrongMove("Not your turn")
322
323             if pos2 not in self.possible_moves(pos1):
324                 raise WrongMove('illegal move')
325             king_color = 'white' if self.move_num % 2 == 0 else 'black'
326             cur_board = Game.c_board(self.board)
327
328             self.board[pos1[0]][pos1[1]], self.board[pos2[0]][pos2[1]] = '_', self.board[pos1[0]][pos1[1]]
329             if self.is_check(self.get_position('king', king_color)[0]):
330                 self.board = cur_board
331                 raise WrongMove('Moving under check')
332
333             self.history.append(cur_board)
334             self.move_num += 1
335
336             king_color = 'white' if self.move_num % 2 == 0 else 'black' # checking if the last move made a check
337             king_pos = self.get_position('king', king_color)[0]
338             if self.is_check(king_pos):
339                 if self.checkmate(king_pos):
340                     self.board = Game.c_board(cur_board)
341                     return -1
342                 else:
343                     print(king_color, 'check!')
344             self.board = Game.c_board(cur_board)
345             return 1
346
347         except Exception as e:
348             print('You made an illegal move: ', e)
349
350     return 0
351

```

Рисунок 1.9 — Функция rules

```

368
369     @GameMixin.check_rules
370     def move(self, pos1, pos2):
371         if self.board[pos1[0]][pos1[1]] != '_' and pos1 != pos2:
372             self.board[pos2[0]][pos2[1]] = self.board[pos1[0]][pos1[1]]
373             self.board[pos1[0]][pos1[1]] = '_'
374         return
375

```

Рисунок 1.10 — Функция move

Функции связаны друг с другом, так как при вызове функции move, её декоратор check_rules (Рисунок 1.11) проводит проверку легитимности хода, через rules.

```

51
52     def check_rules(func):
53         def wrapper_move(self,*args):
54             res = self.rules(*args)
55             if res == 1:
56                 func(self,*args) # Making a move
57                 print(f'Move from {args[0]} to {args[1]} is valid')
58             elif res == 0:
59                 print(f'Move from {args[0]} to {args[1]} invalid')
60             else:
61                 func(self,*args)
62                 self.finish_game('Checkmate!')
63         return wrapper_move
64

```

Рисунок 1.11 — Функция check_rules

1.1.3 Player

Класс, определяющий игрока (Рисунок 1.12). Обладает атрибутом, содержащим имя игрока.

```

292
293
294     class Player(ChessPlayer):
295         def __init__(self,name):
296             self.name = name
297
298         def create_player(name):
299             return Player(name)
300
301         def get_name(self):
302             return self.name
303

```

Рисунок 1.12 — Класс Player

1.1.4 Figure

Класс, определяющий фигуры (Рисунок 1.13). Атрибутом являются тип, цвет и значение(вес) фигуры. Добавлен overload оператора сравнения, чтобы объекты класса фигур можно было сравнить друг с другом по весу, для дальнейшего анализа партии. Элементами массива table являются

unicode коды для символов шахмат, с их помощью в командной строке можно отобразить следующее (Рисунок 1.14)

```

405
406 class Figure(ChessFigure):
407     table = {'black':{'pawn': '\u2659' , 'rook': '\u2656' , 'bishop': '\u2657' , 'king': '\u2654' , 'queen': '\u2655' , 'knight': '\u2658' },\
408             'white': {'pawn': '\u265F' , 'rook': '\u265C' , 'bishop': '\u265D' , 'king': '\u265A' , 'queen': '\u265B' , 'knight': '\u265E' }}
409
410     def __init__(self,type,color,value=0):
411         self.type = type
412         self.color = color
413         self.value = value
414         self.sname = color+'-'+type
415
416     def __str__(self):
417         return Figure.table[self.color][self.type]
418
419     def __lt__(self, other):
420         if self.value < other.value:
421             return True
422         return False
423
424     def __gt__(self,other):
425         if self.value > other.value:
426             return True
427         return False
428
429     def show(self):
430         print(self.type, self.color)
431

```

Рисунок 1.13 — Класс Figure

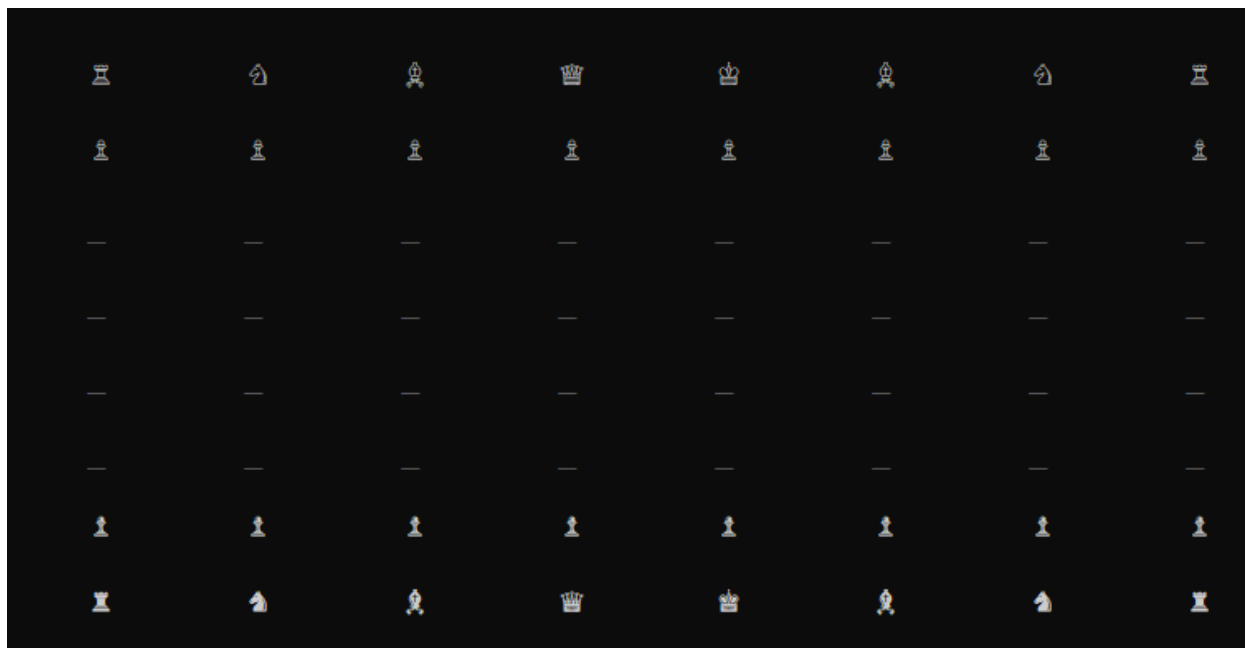


Рисунок 1.14 — Unicode символы шахматных фигур

1.1.5 Terminal

Класс, обеспечивающий интерфейс с пользователем в терминале, а также взаимодействие с БД. Рассмотрим самую важную функцию, содержащую в себе все остальные - `command_window` (Рисунки 1.15,1.16,1.17).

```

452     async def command_window(self):
453         async def handle_new_game():
454             flag = False
455             txt_in = input('Choose side: w (white) or b (black) >>> ')
456             if txt_in != 'w' and txt_in != 'b':
457                 raise WrongCommand('Choose the right side')
458             if txt_in == 'w':
459                 flag = True
460             self.game = Game(side = txt_in, player = self.chosen_player)
461             self.game.set_all_figures(reverse = flag)
462             self.game.show_game()
463
464         async def handle_player():
465             try:
466                 with sqlite3.connect('chess.db') as conn:
467                     cur = conn.cursor()
468                     cur.execute("CREATE TABLE IF NOT EXISTS Player (Player_id INTEGER PRIMARY KEY, Player_name TEXT)")
469                     cur.execute("SELECT Player_name FROM Player")
470                     players_db = cur.fetchall()
471                     print(*players_db, sep = '\n')
472                     print('Choose your player account, or create a new one (enter </create <player_nickname> > )')
473                     txt_in = input('>>>').split()
474                     if '/create' in txt_in and len(txt_in) == 2:
475                         for t in players_db:
476                             if t[0] == txt_in[1]:
477                                 raise WrongCommand('Player already exists')
478                             self.chosen_player = Player(txt_in[1])
479                             cur.execute("INSERT INTO Player(Player_name) VALUES(?)", (self.chosen_player.name,))
480                             elif txt_in[0] in [i[0] for i in players_db]:
481                                 self.chosen_player = Player(txt_in[0])
482                             else:
483                                 raise WrongCommand('Incorrect name for a player')
484
485             except Exception as e:
486                 print('Database connection error: ', e)
487
488         async def handle_move():
489             moves = input('Enter your move >> ').split()
490             if len(moves) != 2:
491                 raise WrongCommand('Wrong amount of arguments for the move')
492             self.game.move(GameMixin.translate_to_pos(moves[0]), GameMixin.translate_to_pos(moves[1]))
493             self.game.show_game()
494

```

Рисунок 1.15 — Функция command_window(1)

```

487
488     async def handle_move():
489         moves = input('Enter your move >> ').split()
490         if len(moves) != 2:
491             raise WrongCommand('Wrong amount of arguments for the move')
492         self.game.move(GameMixin.translate_to_pos(moves[0]), GameMixin.translate_to_pos(moves[1]))
493         self.game.show_game()
494
495     async def handle_retract_move():
496         self.game.retract_move()
497         self.game.show_game()
498
499     async def handle_history():
500         try:
501             with sqlite3.connect('chess.db') as conn:
502                 cur = conn.cursor()
503                 cur.execute("CREATE TABLE IF NOT EXISTS Games (GameID INTEGER PRIMARY KEY, Player_name TEXT, Result TEXT, Game_duration TEXT, Game_time TEXT)")
504                 cur.execute("SELECT * FROM Games")
505                 res = cur.fetchall()
506                 print('GameID Player_name Result(win) Game_duration(seconds) Game_time'.replace(' ', '\t\t'))
507                 for game_res in res:
508                     print('\t\t'.join(str(x) for x in game_res))
509         except Exception as e:
510             print('Database connection error: ', e)
511
512     menu_num = 1
513     self.chosen_player = Player('Default')
514     while 1:
515         try:
516             self.show_menu(menu_num)
517             txt_in = input('>>>')
518             command_num = int(txt_in)
519             if menu_num == 1:
520                 if command_num not in self.menu_1:
521                     raise WrongCommand('Wrong command number')
522                 if command_num == 1:
523                     await handle_new_game()
524                     menu_num = 2
525                 elif command_num == 2:
526                     await handle_history()
527                 elif command_num == 3:
528                     await handle_player()
529                 elif command_num == 4:
530                     await asyncio.sleep(1)
531                 elif command_num == 5:
532                     return #Console exit

```

Рисунок 1.16 — Функция command_window(2)

```

511
512     menu_num = 1
513     self.chosen_player = Player('Default')
514     while 1:
515         try:
516             self.show_menu(menu_num)
517             txt_in = input('>>>')
518             command_num = int(txt_in)
519             if menu_num == 1:
520                 if command_num not in self.menu_1:
521                     raise WrongCommand('Wrong command number')
522                 if command_num == 1:
523                     await handle_new_game()
524                     menu_num = 2
525                 elif command_num == 2:
526                     await handle_history()
527                 elif command_num == 3:
528                     await handle_player()
529                 elif command_num == 4:
530                     await asyncio.sleep(1)
531                 elif command_num == 5:
532                     return #Console exit
533
534             elif menu_num == 2:
535                 if command_num not in self.menu_2:
536                     raise WrongCommand('Wrong command number')
537                 elif command_num == 1:
538                     await handle_move()
539                 elif command_num == 2:
540                     self.game.finish_game()
541                     menu_num = 1
542                 elif command_num == 3:
543                     self.game.show_game()
544                 elif command_num == 4:
545                     await handle_retract_move()
546                 elif command_num == 5:
547                     menu_num = 1
548
549         except Exception as e:
550             print('Exception:',e)
551

```

Рисунок 1.17 — Функция command_window(3)

По своей сути, command_window запускает в терминале меню взаимодействия программы с пользователем. Внутри нее самой в отдельные функции вынесены части программы, отвечающие за действие, которое будет выбрано, после того, как это действие будет выбрано пользователем в соответствующем меню. Примеры интерфейса в терминале (Рисунок 1.18,1.19). Асинхронность функции обусловлена тем, что это добавляет возможность запускать несколько объектов типа терминал и переключаться между ними (пользователем), не закрывая другую.

```

Game instance: <Classes.Terminal object at 0x0000014A917A6550>
Player: Default
-----MENU-----
1 New game
2 History
3 Player
4 Change game instance
5 Quit
>>>

```

Рисунок 1.18 — Пример интерфейса в терминале(1)

```

Game instance: <Classes.Terminal object at 0x0000014A917A6550>
Player: Default
-----MENU-----
1 New game
2 History
3 Player
4 Change game instance
5 Quit
>>>1
Choose side: w (white) or b (black) >>> w

8      ♖      ♜      ♝      ♚      ♗      ♞      ♟      ♠
7      ♙      ♙      ♙      ♙      ♙      ♙      ♙      ♙
6      —      —      —      —      —      —      —      —
5      —      —      —      —      —      —      —      —
4      —      —      —      —      —      —      —      —
3      —      —      —      —      —      —      —      —
2      ♟      ♟      ♟      ♟      ♟      ♟      ♟      ♟
1      ♠      ♟      ♞      ♚      ♗      ♝      ♜      ♖
      a      b      c      d      e      f      g      h

Game instance: <Classes.Terminal object at 0x0000014A917A6550>
-----GAME MENU-----
1 Move
2 Surrender
3 Show game
4 Retract move
5 Back to the menu
>>>_

```

Рисунок 1.19 — Пример интерфейса в терминале(2)

1.1.6 ChessApp

Класс, реализующий GUI для программы при помощи модуля tkinter. По своей сути, является оберткой для объектов класса Game, как и Terminal, только реализует это в диалоговом окне, а не в окне консоли. Рассмотрим наиболее важные функции.

Как не странно, важнейшей частью является отрисовка интерфейса, она релизована в init. На рисунке 1.20 представлена наиболее важная ее часть, загрузка png картинок для иконок шахматных фигур. Остальная часть не так интересна, поскольку определяет меню для главного окна.

```
572
573     def __init__(self, master):
574         self.game = Game()
575         self.game.set_all_figures(reverse=True)
576         self.master = master
577         self.master.minsize(width=400, height=430)
578         self.master.maxsize(width=400, height=430)
579         # C://Users//sta30//Downloads//Учеба//PROG//
580         self.master.title("Chess App")
581         self.board_canvas = tk.Canvas(master, width=400, height=400, bg = 'black')
582         self.board_canvas.pack()
583         self.selected_square = None
584         self.lock = 0
585
586     def read_imgs():
587         piece = ['rook', 'king', 'queen', 'knight', 'bishop', 'pawn']
588         d = 'Chess_img//'
589         b = 'black-'; w = 'white-'
590         orig = []
591         for c1 in ['g', 'w']:
592             for c2 in [b, w]:
593                 orig += [(c1+c2+x, Image.open(d + c1 + c2 + x + '.png')) for x in piece]
594         resized = []
595         for name, img in orig:
596             resized += [(name, ImageTk.PhotoImage(img.resize((32, 32))))]
597         self.imgs = dict(resized)
598     read_imgs()
599
```

Рисунок 1.20 — Часть init

На вход подается master = tk.Tk(), на нем строится весь интерфейс. в read_imgs читаются png иконки для шахматных фигур, после они уменьшаются до нужного размера и записываются в словарь, для дальнейшего заполнения ими отрисованной доски.

set_board - функция отрисовки и заполнения доски (Рисунок 1.21)

```

552 class ChessApp():
553
554     def set_board(self):
555         if self.board_canvas:
556             self.board_canvas.delete('all')
557         self.table_id = [[0 for y in range(8)] for x in range(8)]
558         for i in range(8):
559             color = "white" if i % 2 == 0 else "gray"
560             for j in range(8):
561                 x0 = j * 50
562                 y0 = i * 50
563                 x1 = x0 + 50
564                 y1 = y0 + 50
565                 square_id = self.board_canvas.create_rectangle(x0, y0, x1, y1, fill=color)
566                 self.table_id[i][j] = square_id
567                 self.board_canvas.tag_bind(square_id, "<Button-1>", Lambda event, square_id=square_id, square_id_pos = i*8+j+1: self.select_square(square_id,square_id_pos))
568                 if self.game.board[i][j] != '.':
569                     pic_id = self.board_canvas.create_image(j*50+25,i*50+25,image = self.ims[color[0]+self.game.board[i][j].sname])
570                     self.board_canvas.tag_bind(pic_id, "<Button-1>", Lambda event,square_id = square_id,square_id_pos = i*8+j+1: self.select_square(square_id,square_id_pos))
571                 color = "white" if color == "gray" else "gray"

```

Рисунок 1.21 — Функция set_board

Она проходит по всем возможным клеткам шахматной доски и отрисовывает соответствующие фигуры, стоящие на этих клетках. В качестве элементов доски служат элементы атрибута game.board (двумерный массив), game - объект класса Game (т.е. game - объект содержащий состояние партии, а функция всего лишь считывает его атрибут board и на соответствующие клетки отрисовывает иконки соотв. фигур). Также, к отрисованным фигурам привязывается event на нажатие левой кнопки мыши при помощи canvas.tag_bind. Таким образом получаем кликабельную доску (Рисунок 1.22).

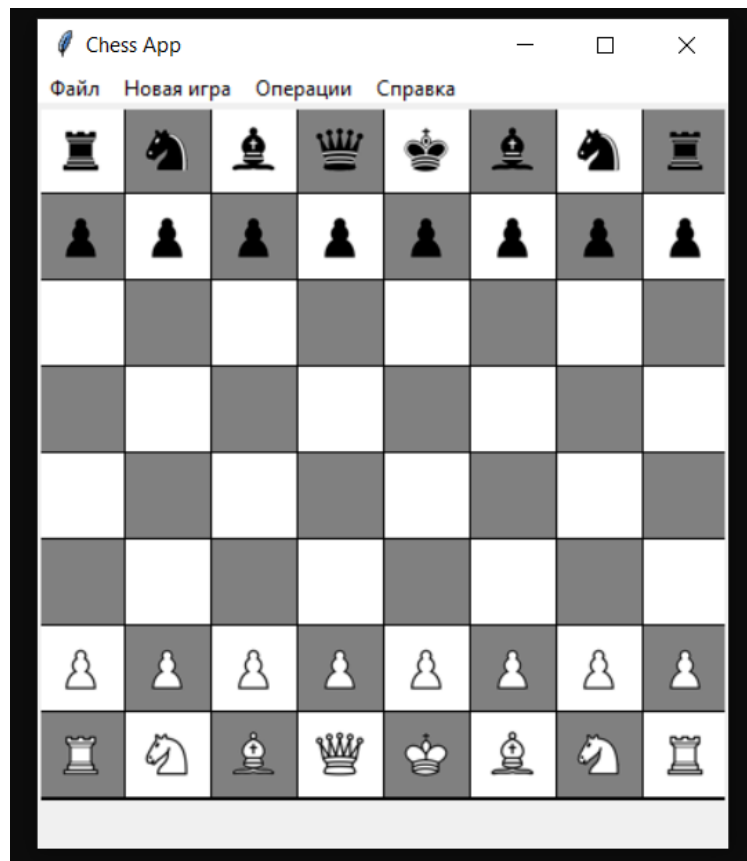


Рисунок 1.22 — Отрисованная доска

Функция `event`, которая вызывается при нажатии на клетку доски запускает функцию `select_square` (Рисунок 1.23). Эта функция подсвечивает выбранную клетку красным, а также возможные ходы из этой клетки (зеленым цветом). Также внизу окна отражается выбранная клетка. Пример(Рисунок 1.24).

```

641 def select_square(self, square_id, square_id_pos):
642     if not self.lock:
643         color = 'white' if self.game.move_num%2 == 0 else 'black'
644         if self.selected_square:
645             self.game.move(self.last_move, ChessApp.convert(square_id_pos))
646             self.set_board()
647             color = 'white' if self.game.move_num%2 == 0 else 'black'
648             king_pos = self.game.get_position('king', color)[0]
649             if self.game.checkmate(king_pos):
650                 self.line.configure(text=f'Checkmate! Game finished in {round(time.time() - self.game.time_start)} seconds')
651                 self.lock = 1
652                 return
653             self.selected_square = None
654         else:
655             self.selected_square = square_id
656             self.last_move = ChessApp.convert(square_id_pos)
657             self.board_canvas.itemconfig(self.selected_square, outline="red", width=2)
658             color = 'white' if self.game.move_num%2 == 0 else 'black'
659             if self.game.board[self.last_move[0]][self.last_move[1]] != '_' and self.game.board[self.last_move[0]][self.last_move[1]].color == color :
660                 moves = self.game.possible_moves(self.last_move)
661                 for m1, m2 in moves:
662                     self.board_canvas.itemconfig(self.table_id[m1][m2], outline="green", width=2)
663             self.line.configure(text=f'Выбрана клетка: {Game.translate_to_str(ChessApp.convert(square_id_pos))}')
664
665

```

Рисунок 1.23 — Функция `select_square`

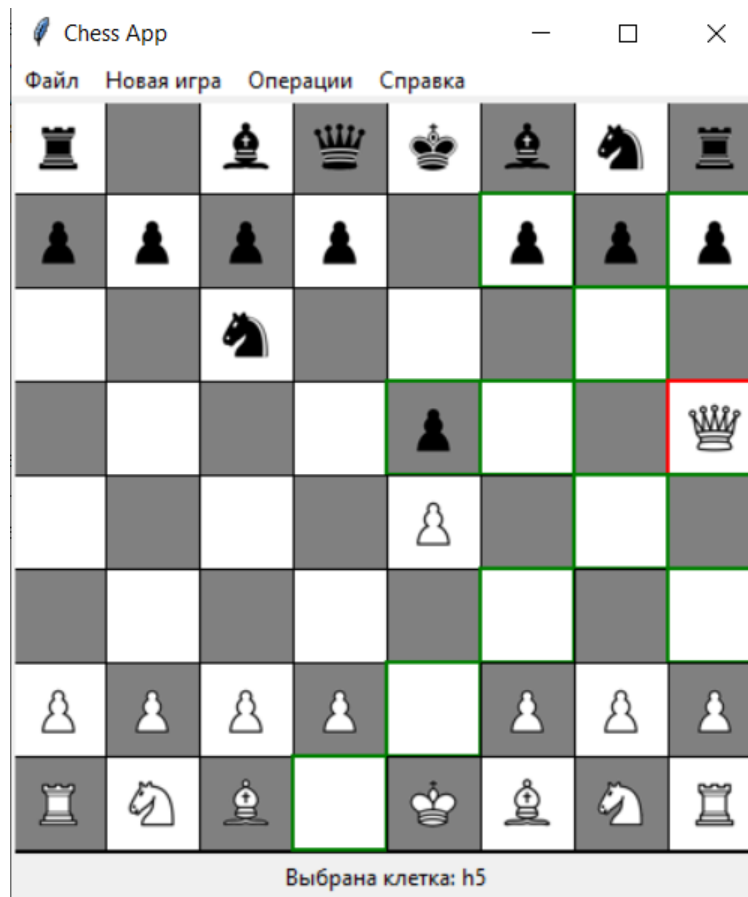


Рисунок 1.24 — Пример работы select_square

За взаимодействие с БД отвечают `get_history` (Рисунок 1.25) и `get_players` (Рисунок 1.26). `get_history` открывает новое окно, в котором можно посмотреть историю всех предыдущих матчей, включая те, что были проведены в текущую сессию (до закрытия программы). `get_players` предоставляет пользователю возможность выбрать аккаунт, на котором он будет играть (это отразится в истории матчей). Примеры того, как это выглядит (Рисунок 1.27, 1.28)

```

684
685     def get_history(self):
686         try:
687             with sqlite3.connect('chess.db') as conn:
688                 cur = conn.cursor()
689                 cur.execute("CREATE TABLE IF NOT EXISTS Games (GameID INTEGER PRIMARY KEY, Player_name TEXT, Result TEXT, Game_duration TEXT, Game_time TEXT)")
690                 cur.execute("SELECT * FROM Games")
691                 res = cur.fetchall()
692                 table = PrettyTable()
693                 table.field_names = 'GameID Player_name Result(win) Game_duration(seconds) Game_time'.split()
694                 for i in res:
695                     table.add_row(i)
696
697             except Exception as e:
698                 print('Database connection error: ', e)
699
700             top = tk.Toplevel(self.master)
701             top.title("Game history")
702             top.minsize(width=550, height=400)
703             canvas = tk.Canvas(top, scrollregion=(-1000, -1000, 10000, 10000))
704             canvas.create_text(270, 370, text = table)
705
706             scrollbar = tk.Scrollbar(top, orient = 'vertical')
707             scrollbar.pack(side='right', fill='y')
708             scrollbar.config(command=canvas.yview)
709
710             canvas.config(yscrollcommand=scrollbar.set)
711             canvas.pack(side='left', fill='both', expand=True)
712

```

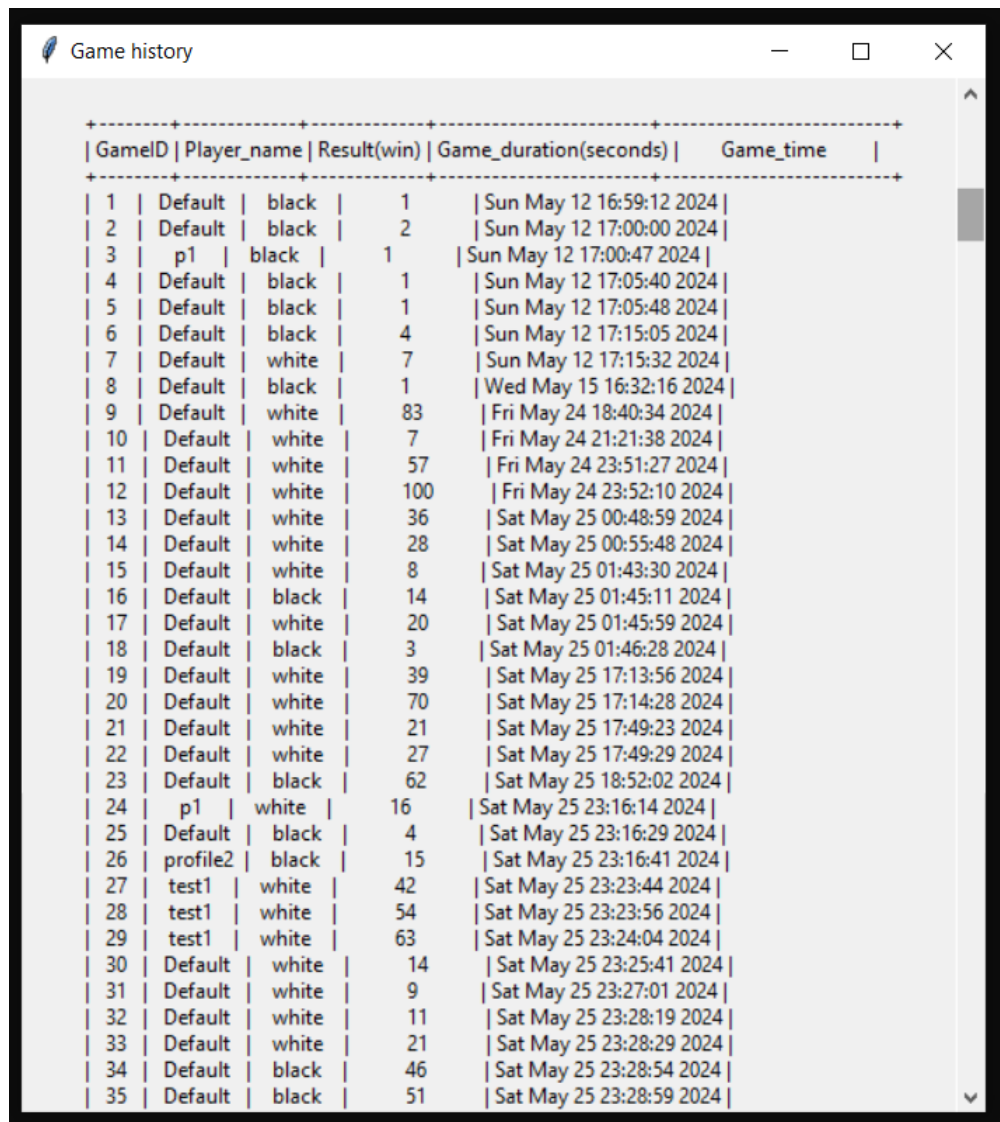
Рисунок 1.25 — Функция get_history

```

712
713     def get_players(self):
714         try:
715             with sqlite3.connect('chess.db') as conn:
716                 cur = conn.cursor()
717                 cur.execute("CREATE TABLE IF NOT EXISTS Player (Player_id INTEGER PRIMARY KEY, Player_name TEXT)")
718                 cur.execute("SELECT Player_name FROM Player")
719                 players_db = cur.fetchall()
720             except Exception as e:
721                 print('Database connection error: ', e)
722
723             top = tk.Toplevel(self.master)
724             top.minsize (width = 200, height = 100)
725             top.title("Player List")
726
727             var = tk.StringVar(value = self.game.player.name)
728             def set_player():
729                 self.game.player = Player(var.get())
730             for player in players_db:
731                 tk.Radiobutton(top, text=player[0], value = player[0], variable = var, command = set_player).pack()
732

```

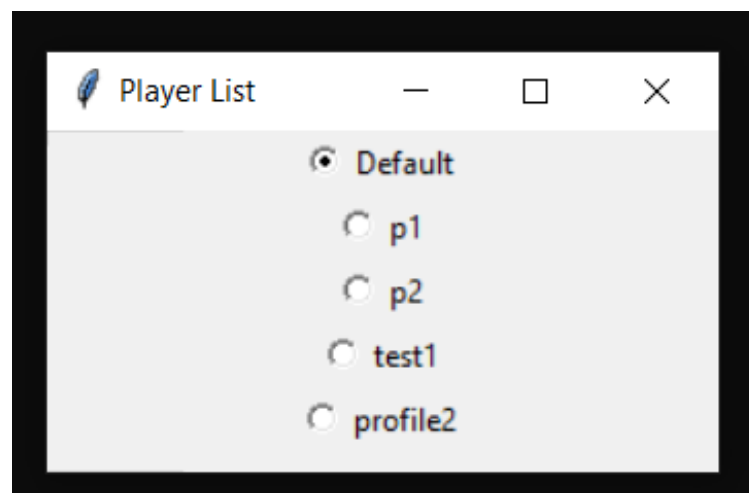
Рисунок 1.26 — Функция get_players



The screenshot shows a window titled "Game history" with a table of game records. The table has five columns: GameID, Player_name, Result(win), Game_duration(seconds), and Game_time. The records are numbered 1 through 35, showing various players and game durations.

GameID	Player_name	Result(win)	Game_duration(seconds)	Game_time
1	Default	black	1	Sun May 12 16:59:12 2024
2	Default	black	2	Sun May 12 17:00:00 2024
3	p1	black	1	Sun May 12 17:00:47 2024
4	Default	black	1	Sun May 12 17:05:40 2024
5	Default	black	1	Sun May 12 17:05:48 2024
6	Default	black	4	Sun May 12 17:15:05 2024
7	Default	white	7	Sun May 12 17:15:32 2024
8	Default	black	1	Wed May 15 16:32:16 2024
9	Default	white	83	Fri May 24 18:40:34 2024
10	Default	white	7	Fri May 24 21:21:38 2024
11	Default	white	57	Fri May 24 23:51:27 2024
12	Default	white	100	Fri May 24 23:52:10 2024
13	Default	white	36	Sat May 25 00:48:59 2024
14	Default	white	28	Sat May 25 00:55:48 2024
15	Default	white	8	Sat May 25 01:43:30 2024
16	Default	black	14	Sat May 25 01:45:11 2024
17	Default	white	20	Sat May 25 01:45:59 2024
18	Default	black	3	Sat May 25 01:46:28 2024
19	Default	white	39	Sat May 25 17:13:56 2024
20	Default	white	70	Sat May 25 17:14:28 2024
21	Default	white	21	Sat May 25 17:49:23 2024
22	Default	white	27	Sat May 25 17:49:29 2024
23	Default	black	62	Sat May 25 18:52:02 2024
24	p1	white	16	Sat May 25 23:16:14 2024
25	Default	black	4	Sat May 25 23:16:29 2024
26	profile2	black	15	Sat May 25 23:16:41 2024
27	test1	white	42	Sat May 25 23:23:44 2024
28	test1	white	54	Sat May 25 23:23:56 2024
29	test1	white	63	Sat May 25 23:24:04 2024
30	Default	white	14	Sat May 25 23:25:41 2024
31	Default	white	9	Sat May 25 23:27:01 2024
32	Default	white	11	Sat May 25 23:28:19 2024
33	Default	white	21	Sat May 25 23:28:29 2024
34	Default	black	46	Sat May 25 23:28:54 2024
35	Default	black	51	Sat May 25 23:28:59 2024

Рисунок 1.27 — Пример get_history



The screenshot shows a window titled "Player List" with a list of players. Each player name is preceded by a radio button. The "Default" option is selected.

- ☒ Default
- ☐ p1
- ☐ p2
- ☐ test1
- ☐ profile2

Рисунок 1.28 — Пример get_players

1.1.7 StartProgramm

Класс StartProgramm(Рисунок 1.29). Его инициализация, запускает терминал, в котором предлагается выбрать играть ли в терминале или с графическим интерфейсом. Он создан для того, чтобы после того как пользователь импортировал себе в файл все вышеперечисленные классы, включая StartProgramm, ему бы оставалось лишь прописать StartProgramm() для запуска приложения. Т.е. класс создан сугубо ради удобства запуска. В нем также можно указать сколько версий одновременно вы хотите запустить (для консольной версии, то что закомментировано).

```
741
742 class StartProgramm():
743     def __init__(self):
744         while 1:
745             txt = input('Введите\n1 для игры в консоли\n2 для запуска с граф. интерфейсом\n>>> ')
746             try:
747                 if int(txt) == 1 or int(txt) == 2:
748                     break
749                 else:
750                     raise Exception
751             except:
752                 print('Ошибка ввода, выберите снова ')
753
754         if int(txt) == 1:
755             t = Terminal()
756             # # t2 = Terminal()
757             async def main():
758                 task1 = asyncio.create_task(t.command_window())
759                 # # task2 = asyncio.create_task(t2.command_window())
760                 await task1
761                 # # await task2
762             asyncio.run(main())
763
764         else:
765             root = tk.Tk()
766             app = ChessApp(root)
767             root.mainloop()
768
```

Рисунок 1.29 — StartProgramm

ЗАКЛЮЧЕНИЕ

В результате выполнения домашнего задания было создано приложение для игры в шахматы с поддержкой графического интерфейса, написанного с помощью модуля tkinter, все функции которого работают в соответствии с заданием к работе. Приложение обладает отловом ошибок для любых ошибок пользователя, как при взаимодействии с консольной версией, так и с диалоговым окном. В процессе выполнения работы был получен опыт проектирования структуры ООП приложений и их интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Код ДЗ // Google Drive : Облачное хранилище файлов .URL:
[https://drive.google.com/drive/folders/14re6r9UR3kA00U_
odMRxeBq4QqNCJ2SU?usp=sharing](https://drive.google.com/drive/folders/14re6r9UR3kA00U_odMRxeBq4QqNCJ2SU?usp=sharing) (дата обращения 26.05.2024)