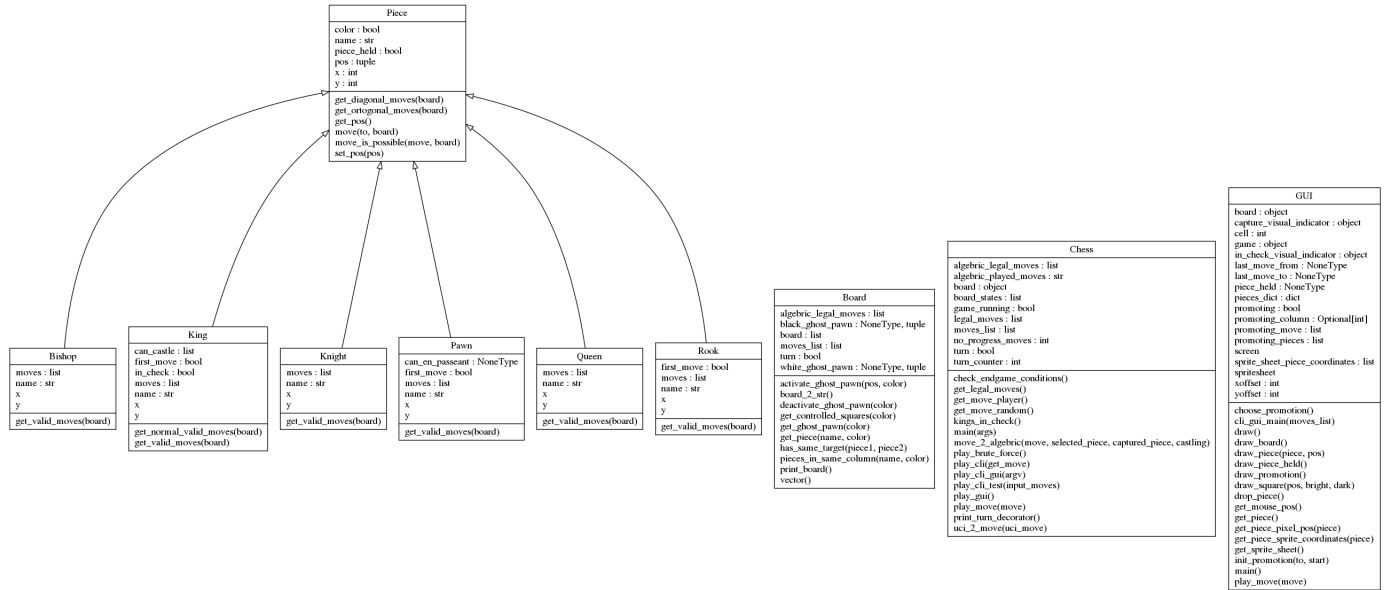# 1 Program Scope

The program should be able to receive as input a chess move in UCI(Universal Chess Interface) format i.e e2e4, and if the movement is valid, output the board state to the user or inform the user the input isn't valid. For this matter, the standard python library is enough address the problem. For debugging purposes, a graphical interface was also required and implemented in pygame, a graphical framework for games.

2 Program project

Piece
color : bool
name : str
piece_held : bool
pos : tuple
x : int
y : int
get_diagonal_moves(board)
get_ortogonal_moves(board)
get_pos()
move(to, board)
move_is_possible(move, board)
set_pos(pos)

Bishop
moves : list
name : str
x
y
get_valid_moves(board)

King
can_castle : list
first_move : bool
in_check : bool
moves : list
name : str
x
y
get_normal_valid_moves(board)
get_valid_moves(board)

Knight
moves : list
name : str
x
y
get_valid_moves(board)

Pawn
can_en_passant : NoneType
first_move : bool
moves : list
name : str
x
y
get_valid_moves(board)

Queen
moves : list
name : str
x
y
get_valid_moves(board)

Rook
first_move : bool
moves : list
name : str
x
y
get_valid_moves(board)

Board
algebric_legal_moves : list
black_ghost_pawn : NoneType, tuple
board : list
moves_list : list
turn : bool
white_ghost_pawn : NoneType, tuple
activate_ghost_pawn(pos, color)
board_2_str()
deactivate_ghost_pawn(color)
get_controlled_squares(color)
get_ghost_pawn(color)
get_piece(name, color)
has_same_target(piece1, piece2)
pieces_in_same_column(name, color)
print_board()
vector()

Chess
algebric_legal_moves : list
algebric_played_moves : str
board : object
board_states : list
game_running : bool
legal_moves : list
moves_list : list
no_progress_moves : int
turn : bool
turn_counter : int
check_endgame_conditions()
get_legal_moves()
get_move_player()
get_move_random()
kings_in_check()
main(args)
move_2_algebric(move, selected_piece, captured_piece, castling)
play_brute_force()
play_cli(get_move)
play_cli_gui(argv)
play_cli_test(input_moves)
play_gui()
play_move(move)
print_turn_decorator()
uci_2_move(uci_move)

GUI
board : object
capture_visual_indicator : object
cell : int
game : object
in_check_visual_indicator : object
last_move_from : NoneType
last_move_to : NoneType
piece_held : NoneType
pieces_dict : dict
promoting : bool
promoting_column : Optional[int]
promoting_move : list
promoting_pieces : list
screen
sprite_sheet_piece_coordinates : list
spritesheet
xoffset : int
yoffset : int
choose_promotion()
cli_gui_main(moves_list)
draw()
draw_board()
draw_piece(piece, pos)
draw_piece_held()
draw_promotion()
draw_square(pos, bright, dark)
drop_piece()
get_mouse_pos()
get_piece()
get_piece_pixel_pos(piece)
get_piece_sprite_coordinates(piece)
get_sprite_sheet()
init_promotion(to, start)
main()
play_move(move)

3 Testing

Number of plies (half-moves)	Number of possible games
1	20
2	400
3	8092
4	197,281
5	4,865,609
6	119,060,324
. . .	. . .
10	69,352,859,712,417

Tabela 1: Shannon's Calculation. Obs: A turn is composed by a white move and a black move. Five plies therefore stands for white playing three times and black two.

For testing the general accuracy of the game, it was used the Shannon Number, which stands for all the possible moves that can be played until a certain ply(half-move). By the limitation of the computer power avaible for our

disposal, and considering that the game was not written in a language nor written in a way for fast computation, we could only check the precision of the game until 5 ply, as we can see by the test log:

```
2022−01−22 00:00:36,742 Result of possible games with 1 ply: 20/20 − OK
2022−01−22 00:00:36,742 Elapsed time in 1 ply: 00h00m00s seconds
2022−01−22 00:00:37,312 Result of possible games with 2 ply: 400/400 − OK
2022−01−22 00:00:37,312 Elapsed time in 2 ply: 00h00m00s seconds
2022−01−22 00:00:52,137 Result of possible games with 3 ply: 8902/8902 − OK
2022−01−22 00:00:52,137 Elapsed time in 3 ply: 00h00m14s seconds
2022−01−22 00:07:11,715 Result of possible games with 4 ply: 197281/197281 − OK
2022−01−22 00:07:11,715 Elapsed time in 4 ply: 00h06m19s seconds
2022−01−22 08:45:00,073 Result of possible games with 5 ply: 4865609/4865609 − OK
2022−01−22 08:45:00,073 Elapsed time in 5 ply: 08h37m48s seconds
```

Although this is a good signal that basic operations are working, in 5 plies we cannot test all the complications that might arise during a chess game.

Depth	Captures	E.P	Castles	Promotions	Checks	Dscry Checks	Dbl Checks	Checkmates
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	34	0	0	0	12	0	0	0
4	1576	0	0	0	469	0	0	8
5	82,719	258	0	0	27,251	6	0	347

Tabela 2: Number of "special" moves by depth accordingly to https://www.chessprogramming.org/Perft_Results

By this table we can see that we need to concentrate our efforts in testing Castle, Promotions, Discovery Checks and Double Checks

Specific tests were also made when debugging certain problems in code. So it was possible to replay a certain game until a specific move and debug it from there, without manually giving the moves as input.

For example, at 5 ply, there can't be a game with a promoted pawn case, therefore we need to make a specific test case for that matter.

```
python3 tests/promotionTest.py
python3 chess.py −guitest g2g4 h7h5 g4h5 g7g6 h5h6 h8h7 f2f3 h7g7 h6h7 f7f6
...
PlayedMoves: 1. g4 h5 2. gxh5 g6 3. h6 Rh7 4. f3 Rg7 5. h7 f6
```
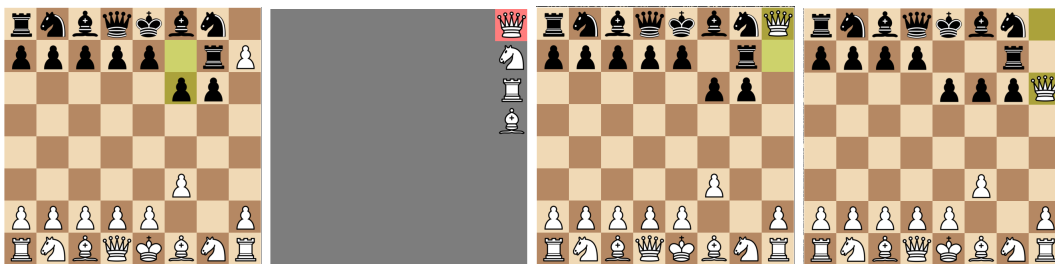

Figura 1: The most left screenshot is the result of the test, and then it was a sequence of screenshots of the user playing and doing the promotion manually.

While developing, the GUI and CLI interface could be behaving differently, having that in mind, we can also do the same test with the CLI if we are in doubt:

```
>> $ python3 tests/promotionTest_cli.py
python3 chess.py −clitest g2g4 h7h5 g4h5 g7g6 h5h6 h8h7 f2f3 h7g7 h6h7 f7f6 h7h8q
e7e6 h8h6
(...)
```

Black's turn to move!
```
********************************
8| r | n | b | q | k | b | n |   |
7| p | p | p | p |   |   | r |   |
6|   |   |   |   | p | p | p | Q |
5|   |   |   |   |   |   |   |   |
4|   |   |   |   |   |   |   |   |
3|   |   |   |   |   | P |   |   |
2| P | P | P | P | P |   |   | P |
1| R | N | B | Q | K | B | N | R |
    a   b   c   d   e   f   g   h
********************************
```

2022−01−16 18:11:02,057 Result of possible games with 1 ply: 20/20 − OK
2022−01−16 18:11:02,057 Elapsed time in 1 ply: 00h00m00s seconds
2022−01−16 18:11:03,073 Result of possible games with 2 ply: 400/400 − OK
2022−01−16 18:11:03,073 Elapsed time in 2 ply: 00h00m01s seconds
2022−01−16 18:11:28,139 Result of possible games with 3 ply: 8902/8902 − OK
2022−01−16 18:11:28,140 Elapsed time in 3 ply: 00h00m25s seconds
2022−01−16 18:20:02,978 Result of possible games with 4 ply: 197281/197281 − OK
2022−01−16 18:20:02,978 Elapsed time in 4 ply: 00h08m34s seconds
2022−01−16 22:00:48,487 Result of possible games with 5 ply: 4865824/4865609 − ERROR
2022−01−16 22:00:48,487 Elapsed time in 5 ply: 03h40m45s seconds
2022−01−16 22:00:48,487 Total Elapsed time: (03h49m46s)