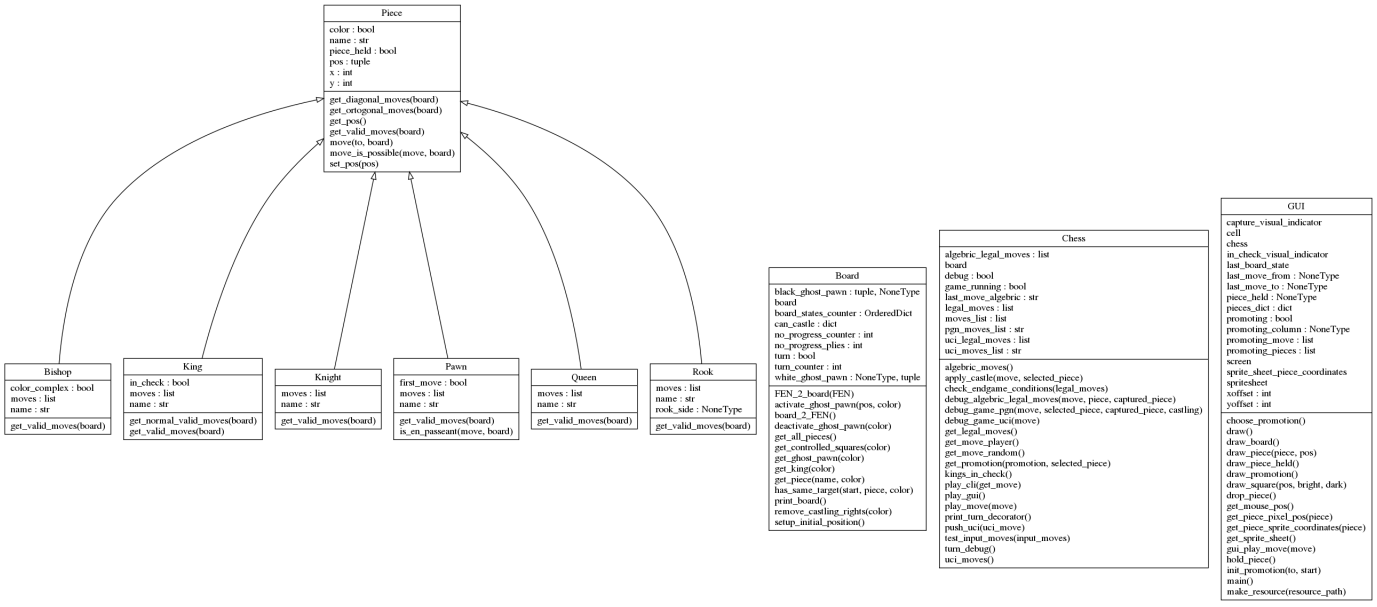
1 Program Scope

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. Also for debugging and testing purposes, it was used the program *pgn-extract* to convert PGN game notation to UCI notation.

2 Program project

The project is constitued by four modules that contains in itself their respective major class: The Piece, Board, Chess and GUI.

1. The Pieces module contains the Piece class, that is inherited by all the chess pieces, and specify how to get from each piece their own set of possible moves.
2. The Board module contains the Board class that is used to save all information relative to board state, such as pieces positions, castling rights, number of turns, en passeant possibility, etc.
3. The Chess module contains the Chess class that is used to process the Board information and create legal moves from which the player can chose to play.
4. The GUI module uses the Board and Chess classes to play the game in a graphical interface mode.



3 Testing

Three approaches were made to certify the accuracy of the game:

1) Shannon's Calculations

2) Brute force complex positions and comparing with a proof table

3) Specific tests to test draw conditions

3.1 Shannon's Number

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 basic operations accuracy, 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.

3.2 Brute forcing complex positions

One way to test them, is loading "complex positions", which contain it its possibilities, castling, promotion, discovery checks and double checks, and brute forcing their possible moves, and comparing the results with a proof table. Using the positions recommended in `https://www.chessprogramming.org/Perft_Results`, we get the following result from 5 "complex positions":

```
>>> $ python3 -m tests.test_brute.py (Running from top module)
2022-01-24 00:03:34,134 ----------------------------------------------------
2022-01-24 00:03:34,134 Initiating move generation test on depth: 2
2022-01-24 00:03:34,268 Result of possible games with 1 ply: 48/48 - OK
2022-01-24 00:03:34,268 Elapsed time in 1 ply: 00h00m00s seconds
2022-01-24 00:03:40,177 Result of possible games with 2 ply: 2039/2039 - OK
2022-01-24 00:03:40,177 Elapsed time in 2 ply: 00h00m05s seconds
2022-01-24 00:03:40,177 Total Elapsed time: (00h00m06s)
2022-01-24 00:03:40,177 ----------------------------------------------------
2022-01-24 00:03:40,177 ----------------------------------------------------
2022-01-24 00:03:40,177 Initiating move generation test on depth: 3
2022-01-24 00:03:40,195 Result of possible games with 1 ply: 14/14 - OK
2022-01-24 00:03:40,195 Elapsed time in 1 ply: 00h00m00s seconds
2022-01-24 00:03:40,454 Result of possible games with 2 ply: 191/191 - OK
2022-01-24 00:03:40,454 Elapsed time in 2 ply: 00h00m00s seconds
2022-01-24 00:03:44,350 Result of possible games with 3 ply: 2812/2812 - OK
2022-01-24 00:03:44,351 Elapsed time in 3 ply: 00h00m03s seconds
2022-01-24 00:03:44,351 Total Elapsed time: (00h00m04s)
2022-01-24 00:03:44,351 ----------------------------------------------------
2022-01-24 00:03:44,358 ----------------------------------------------------
2022-01-24 00:03:44,359 Initiating move generation test on depth: 3
2022-01-24 00:03:44,382 Result of possible games with 1 ply: 6/6 - OK
2022-01-24 00:03:44,382 Elapsed time in 1 ply: 00h00m00s seconds
2022-01-24 00:03:45,135 Result of possible games with 2 ply: 264/264 - OK
2022-01-24 00:03:45,135 Elapsed time in 2 ply: 00h00m00s seconds
2022-01-24 00:04:12,876 Result of possible games with 3 ply: 9467/9467 - OK
2022-01-24 00:04:12,876 Elapsed time in 3 ply: 00h00m27s seconds
2022-01-24 00:04:12,876 Total Elapsed time: (00h00m28s)
2022-01-24 00:04:12,876 ----------------------------------------------------
2022-01-24 00:04:12,881 ----------------------------------------------------
2022-01-24 00:04:12,881 Initiating move generation test on depth: 3
2022-01-24 00:04:12,997 Result of possible games with 1 ply: 44/44 - OK
2022-01-24 00:04:12,997 Elapsed time in 1 ply: 00h00m00s seconds
2022-01-24 00:04:17,258 Result of possible games with 2 ply: 1486/1486 - OK
2022-01-24 00:04:17,258 Elapsed time in 2 ply: 00h00m04s seconds
2022-01-24 00:07:10,379 Result of possible games with 3 ply: 62379/62379 - OK
2022-01-24 00:07:10,379 Elapsed time in 3 ply: 00h02m53s seconds
2022-01-24 00:07:10,379 Total Elapsed time: (00h02m57s)
2022-01-24 01:06:43,597 ----------------------------------------------------
2022-01-24 01:06:43,598 Initiating move generation test on depth: 3
2022-01-24 01:06:43,724 Result of possible games with 1 ply: 46/46 - OK
2022-01-24 01:06:43,724 Elapsed time in 1 ply: 00h00m00s seconds
2022-01-24 01:06:49,460 Result of possible games with 2 ply: 2079/2079 - OK
2022-01-24 01:06:49,460 Elapsed time in 2 ply: 00h00m05s seconds
2022-01-24 01:11:14,405 Result of possible games with 3 ply: 89890/89890 - OK
2022-01-24 01:11:14,423 Elapsed time in 3 ply: 00h04m24s seconds
2022-01-24 01:11:14,424 Total Elapsed time: (00h04m30s)
2022-01-24 01:11:14,424 ----------------------------------------------------
```

By doing these tests, we can be relativery certain that the "special moves" are working as desired.

3.3 Specific tests for Draws

Because Draws take so long to happen, they very rarely appear in brute forcing tests. Because of that, we made specific tests, that have this logic:

```
movedraw50 = [" d2d4 g8f6 c2c4 g7g6 b1c3 f8g7 e2e4 d7d6 g1f3 e8g8 f1e2 e7e5 e1g1
```

```
b8c6 d4d5 c6e7 f3d2 a7a5 a1b1 f6d7 a2a3 f7f5 b2b4 g8h8 f2f3 e7g8 d1c2 g8f6 c3b5
a5b4 a3b4 f6h5 g2g3 d7f6 c4c5 c8d7 b1b3 h5g3 h2g3 f6h5 f3f4 e5f4 c5c6 b7c6 d5c6
h5g3 b3g3 f4g3 c6d7 g3g2 f1f3 d8d7 c1b2 f5e4 f3f8 a8f8 b2g7 d7g7 c2e4 g7f6 d2f3
f6f4 e4e7 f8f7 e7e6 f7f6 e6e8 f6f8 e8e7 f8f7 e7e6 f7f6 e6b3 g6g5 b5c7 g5g4 c7d5
f4c1 b3d1 c1d1 e2d1 f6f5 d5e3 f5f4 f3e1 f4b4 d1g4 h7h5 g4f3 d6d5 e3g2 h5h4 e1d3
b4a4 g2f4 h8g7 g1g2 g7f6 f3d5 a4a5 d5c6 a5a6 c6b7 a6a3 b7e4 a3a4 e4d5 a4a5 d5c6
a5a6 c6f3 f6g5 f3b7 a6a1 b7c8 a1a4 g2f3 a4c4 c8d7 g5f6 f3g4 c4d4 d7c6 d4d8 g4h4
d8g8 c6e4 g8g1 f4h5 f6e6 h5g3 e6f6 h4g4 g1a1 e4d5 a1a5 d5f3 a5a1 g4f4 f6e6 d3c5
e6d6 g3e4 d6e7 f4e5 a1f1 f3g4 f1g1 g4e6 g1e1 e6c8 e1c1 e5d4 c1d1 c5d3 e7f7 d4e3
d1a1 e3f4 f7e7 d3b4 a1c1 b4d5 e7f7 c8d7 c1f1 f4e5 f1a1 e4g5 f7g6 g5f3 g6g7 d7g4
g7g6 d5f4 g6g7 f3d4 a1e1 e5f5 e1c1 g4e2 c1e1 e2h5 e1a1 f4e6 g7h6 h5e8 a1a8 e8c6
a8a1 f5f6 h6h7 e6g5 h7h8 d4e6 a1a6 c6e8 a6a8 e8h5 a8a1 h5g6 a1f1 f6e7 f1a1 g5f7
h8g8 f7h6 g8h8 h6f5 a1a7 e7f6 a7a1 f5e3 a1e1 e3d5 e1g1 g6f5 g1f1 d5f4 f1a1 f4g6
h8g8 g6e7 g8h8 e6g5", "7k/4N3/5K2/5BN1/8/8/8/r7 b - - 100 113"]
```

For example, we feed this list of moves into our specific test function:

```
for test in tests:
    chess = Chess()

    fen = chess.test_input_moves(test[0].split(" "))
    r = fen == test[1]
    result = "OK" if r else "ERROR"
    assert r
    print("Test:", result)
```

The test plays the game and prints the result of the comparison between the expected board state result(FEN) with the FEN given by our function call. If our game returns the exact expected FEN, the test was successfull.

Similar tests were made with:

1) Insufficient material
    1.1) Opposite colors bishops draw
    2.2) Knight and King vs King
    3.3) Bishop and King vs King
    4.4) King vs King
2) Stalemate
3) Three Fold Repetition
4) 50 move without pawn move or capture

4 User Docs

There are three ways to interact with this program, by directly calling their functions in the interpreter or in a script, by calling the *play_cli*() or the *play_gui*() functions.

4.1 play_cli() and play_gui()

You can enter directly the CLI interface by running: $ *python*3 *main.py* − *cli*. Here the program enters in a loop and continously asks moves until it reaches a endgame condition, such as checkmate or draw.

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

White's turn to move!

Legal moves: a2a4 a2a3 b2b4 b2b3 b1c3 b1a3 c2c4 c2c3 d2d4 d2d3 e2e4 e2e3 f2f4 f2f3 g2g4 g2g3

Move: e2e4
********************************
8| r | n | b | q | k | b | n | r |
7| p | p | p | p | p | p | p | p |
6|   |   |   |   |   |   |   |   |
5|   |   |   |   |   |   |   |   |
4|   |   |   |   | P |   |   |   |
3|   |   |   |   |   |   |   |   |
2| P | P | P | P |   | P | P | P |
1| R | N | B | Q | K | B | N | R |
   a   b   c   d   e   f   g   h
********************************

Black's turn to move!

Legal moves: a7a5 a7a6 b8c6 b8a6 b7b5 b7b6 c7c5 c7c6 d7d5 d7d6 e7e5 e7e6 f7f5 f7f6 g8h6 g8f6

Move:
```

You can also enter directly the GUI interface by running: $ *python*3 *main.py* − *gui*. There is no time control, just dragging and dropping pieces, and promoting pawns. The program takes care of prohibiting illegal moves and moving enemy pieces while not your turn. Just as the play_cli(), the game goes on as long as it doesn't reach an endgame condition.

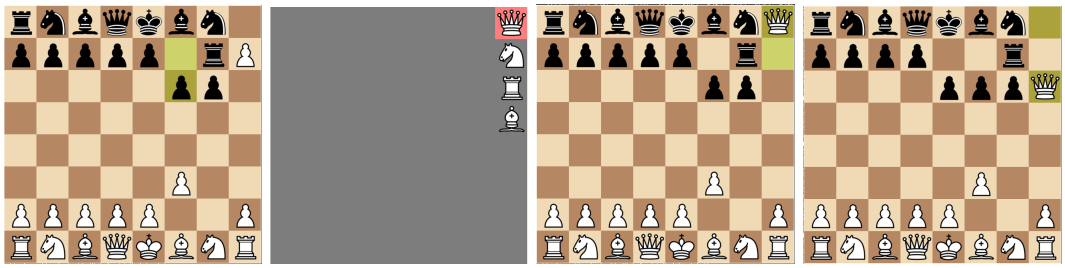


Figura 1: Overview of different states of the game while in GUI. With a Pawn promoting to a Queen.

4.2 Other Uses

By importing the mychess module, we open other possibilities, like loading a position and playing it from there.