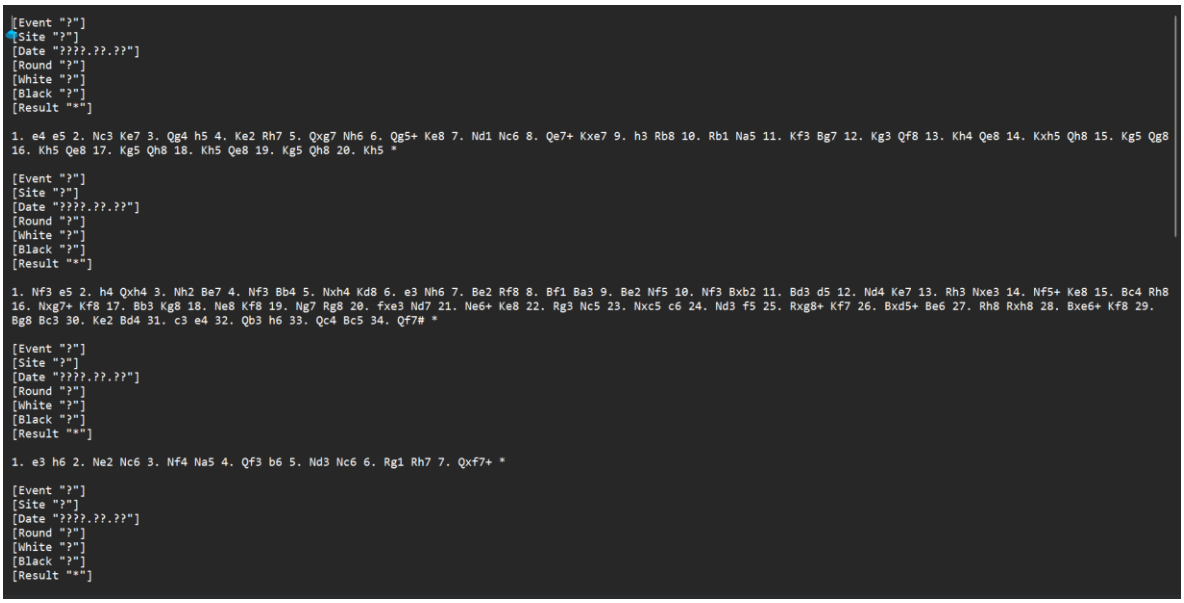
The Wizard's Game

Solution:

In challenge '*Wizard's Logic.txt*' and *'output.png'* is given where txt file contains logic that how output.pgn file is encoded where pgn file represent portable gaming notion

Approach to solve this challenge

1. Output.pgn



```
[Event "?"]
[Site "?"]
[Date "????.??.??"]
[Round "?"]
[White "?"]
[Black "?"]
[Result "*"]

1. e4 e5 2. Nc3 Ke7 3. Qg4 h5 4. Ke2 Rh7 5. Qxg7 Nh6 6. Qg5+ Ke8 7. Nd1 Nc6 8. Qe7+ Kxe7 9. h3 Rb8 10. Rb1 Na5 11. Kf3 Bg7 12. Kg3 Qf8 13. Kh4 Qe8 14. Kxh5 Qh8 15. Kg5 Qg8
16. Kh5 Qe8 17. Kg5 Qh8 18. Kh5 Qe8 19. Kg5 Qh8 20. Kh5 *

[Event "?"]
[Site "?"]
[Date "????.??.??"]
[Round "?"]
[White "?"]
[Black "?"]
[Result "*"]

1. Nf3 e5 2. h4 Qxh4 3. Nh2 Be7 4. Nf3 Bb4 5. Nxh4 Kd8 6. e3 Nh6 7. Be2 Rf8 8. Bf1 Ba3 9. Be2 Nf5 10. Nf3 Bxb2 11. Bd3 d5 12. Nd4 Ke7 13. Rh3 Nxe3 14. Nf5+ Ke8 15. Bc4 Rh8
16. Nxg7+ Kf8 17. Bb3 Kg8 18. Ne8 Kf8 19. Ng7 Rg8 20. fxe3 Nd7 21. Ne6+ Ke8 22. Rg3 Nc5 23. Nxc5 c6 24. Nd3 f5 25. Rxg8+ Kf7 26. Bxd5+ Be6 27. Rh8 Rxh8 28. Bxe6+ Kf8 29.
Bg8 Bc3 30. Ke2 Bd4 31. c3 e4 32. Qb3 h6 33. Qc4 Bc5 34. Qf7# *

[Event "?"]
[Site "?"]
[Date "????.??.??"]
[Round "?"]
[White "?"]
[Black "?"]
[Result "*"]

1. e3 h6 2. Ne2 Nc6 3. Nf4 Na5 4. Qf3 b6 5. Nd3 Nc6 6. Rg1 Rh7 7. Qxf7+ *

[Event "?"]
[Site "?"]
[Date "????.??.??"]
[Round "?"]
[White "?"]
[Black "?"]
[Result "*"]
```

- By seeing this pgn file a chess player can determine this easily this represents chess board moves.
- This pgn file represent a move of chess and in that chess move flag is hidden

- Now look into encoding logic which is given into challenge

```
def encode():
    file_bits_count = len(file_bytes) * 8
    file_bit_index = 0
    chess_board = Board()
    output_pgns = []

    while file_bit_index < file_bits_count:
        legal_moves = list(chess_board.generate_legal_moves())
        max_binary_length = min(int(log2(len(legal_moves))), file_bits_count - file_bit_index)

        move_bits = {move.uci(): to_binary_string(i, max_binary_length)
                     for i, move in enumerate(legal_moves)}

        byte_index = file_bit_index // 8
        file_chunk = "".join(to_binary_string(byte, 8)
                             for byte in file_bytes[byte_index: byte_index + 2])
        next_chunk = file_chunk[file_bit_index % 8: file_bit_index % 8 + max_binary_length]

        for move_uci, move_binary in move_bits.items():
            if move_binary == next_chunk:
                chess_board.push_uci(move_uci)
                break

        file_bit_index += max_binary_length

        if chess_board.legal_moves.count() <= 1 or chess_board.is_insufficient_material() \
           or chess_board.can_claim_draw():
            pgn_board = pgn.Game()
            pgn_board.add_line(chess_board.move_stack)
            output_pgns.append(str(pgn_board))
            chess_board.reset()
```

Now to decode this pgn file we have to create a code using which we can decode this chess moves

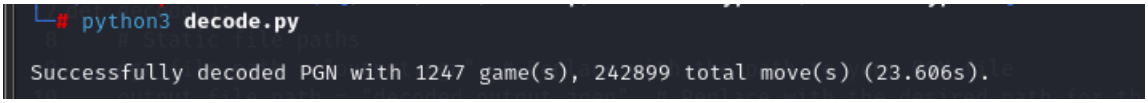
Decode.py

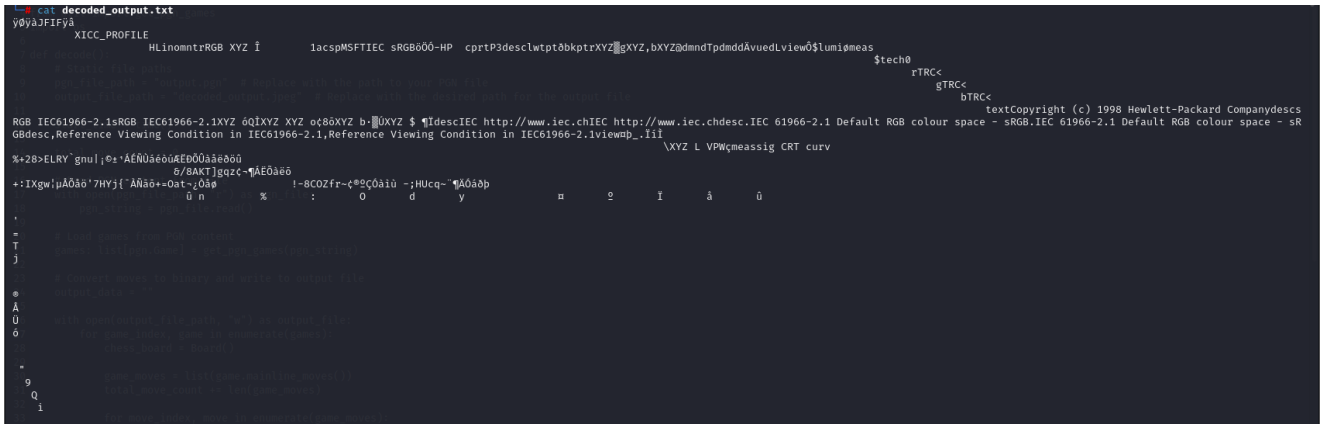
```
from time import time
from math import log2
from chess import pgn, Board
from util import get_pgn_games

# Path to the PGN file
pgn_file_path = "output.pgn"
# Path to save the decoded output
output_file_path = "decoded_output.txt"

def decode(pgn_file_path: str, output_file_path: str):
    start_time = time()

    total_move_count = 0

    # Read PGN string from file
    with open(pgn_file_path, "r") as file:
        pgn_string = file.read()

    # Load games from PGN file
    games: list[pgn.Game] = get_pgn_games(pgn_string)

    # Convert moves to binary and write to output file
    with open(output_file_path, "wb") as output_file:
```

```
    output_data = ""

    for game_index, game in enumerate(games):
        chess_board = Board()

        game_moves = list(game.mainline_moves())
        total_move_count += len(game_moves)

        for move_index, move in enumerate(game_moves):
            # Get UCIs of legal moves in the current position
            legal_move_ucis = [
                legal_move.uci()
                for legal_move in list(chess_board.generate_legal_moves())
            ]

            # Get binary of the move played using its index in the legal moves
            move_binary = bin(legal_move_ucis.index(move.uci()))[2:]

            # If this is the last move of the last game,
            # binary cannot go over a total length multiple of 8
            if game_index == len(games) - 1 and move_index == len(game_moves) - 1:
                max_binary_length = min(
                    int(log2(len(legal_move_ucis))),
                    8 - (len(output_data) % 8)
                )
            else:
                max_binary_length = int(log2(len(legal_move_ucis)))

            # Pad move binary to meet max binary length
            required_padding = max(0, max_binary_length - len(move_binary))
            move_binary = ("0" * required_padding) + move_binary

            # Play move on the board
            chess_board.push_uci(move.uci())

            # Add move binary to output data string
            output_data += move_binary

            # If output binary pool is a multiple of 8, flush it to the file
            if len(output_data) % 8 == 0:
                output_file.write(
                    bytes([int(output_data[i * 8: i * 8 + 8], 2)
                           for i in range(len(output_data) // 8)])
                )
                output_data = ""

print(
```

```
        f"\nSuccessfully decoded PGN with {len(games)} game(s), "
        f"{total_move_count} total move(s) ({round(time() - start_time, 3)}s)."
    )

# Call the decode function
decode(pgn_file_path, output_file_path)
```

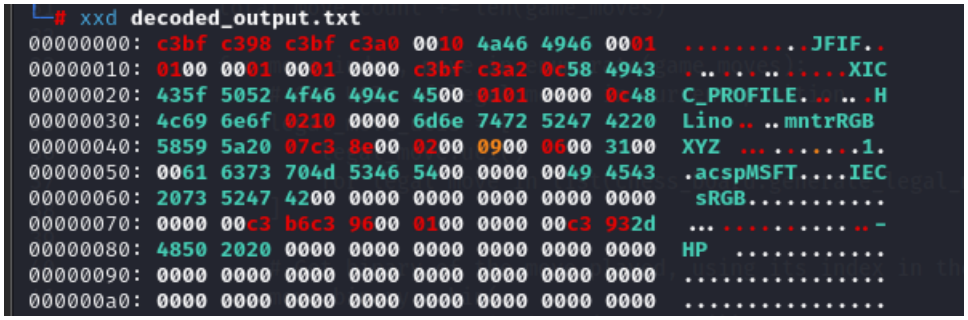
After running this code you will get decoded_output.txt file





As txt file contains some raw data so we can determine this as this is not a txt format.

By checking hex of this you can determine that actually this file is image



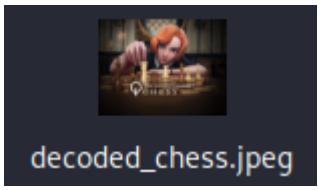
Modify code to decode pgn to jpeg

```
from time import time
from math import log2
from chess import pgn, Board
from util import get_pgn_games

# Path to the PGN file
pgn_file_path = "output.pgn"
```

```
# Path to save the decoded output
output_file_path = "decoded_chess.jpeg"

def decode(pgn_file_path: str, output_file_path: str):
    start_time = time()

    total_move_count = 0

    # Read PGN string from file
    with open(pgn_file_path, "r") as file:
        pgn_string = file.read()

    # Load games from PGN file
    games: list[pgn.Game] = get_pgn_games(pgn_string)

    # Convert moves to binary and write to output file
    with open(output_file_path, "wb") as output_file:
        output_data = ""

        for game_index, game in enumerate(games):
            chess_board = Board()

            game_moves = list(game.mainline_moves())
            total_move_count += len(game_moves)

            for move_index, move in enumerate(game_moves):
                # Get UCIs of legal moves in the current position
                legal_move_ucis = [
                    legal_move.uci()
                    for legal_move in list(chess_board.generate_legal_moves())
                ]

                # Get binary of the move played using its index in the legal moves
                move_binary = bin(legal_move_ucis.index(move.uci()))[2:]

                # If this is the last move of the last game,
                # binary cannot go over a total length multiple of 8
                if game_index == len(games) - 1 and move_index == len(game_moves) - 1:
                    max_binary_length = min(
                        int(log2(len(legal_move_ucis))),
                        8 - (len(output_data) % 8)
                    )
                else:
                    max_binary_length = int(log2(len(legal_move_ucis)))

                # Pad move binary to meet max binary length
                required_padding = max(0, max_binary_length - len(move_binary))
```

```
            move_binary = ("0" * required_padding) + move_binary

            # Play move on the board
            chess_board.push_uci(move.uci())

            # Add move binary to output data string
            output_data += move_binary

            # If output binary pool is a multiple of 8, flush it to the file
            if len(output_data) % 8 == 0:
                output_file.write(
                    bytes([int(output_data[i * 8: i * 8 + 8], 2)
                           for i in range(len(output_data) // 8)])
                )
                output_data = ""

    print(
        f"\nSuccessfully decoded PGN with {len(games)} game(s), "
        f"{total_move_count} total move(s) ({round(time() - start_time, 3)}s)."
    )

# Call the decode function
decode(pgn_file_path, output_file_path)
```

Output:



Check exifmeta data of this jpeg file

In comment section of data, you can see something is stored in decoded manner

- Encoded data: “++++++++++[>+>+++>+++++++>++++++++++<<<<-
]>>>+++.+++++.++++++.----------.--.++++++++++.---------.++++++++++.---
--------.>++++++++++++++++++++++++.<--.----------
.++++++++++++++++++++++++++.----------.----------
.++++++++++++++++++++++++++++++++++++.++++++++++.<++++++++
++++++++++.++++++++++++++++++++++++.>.<----------.>.----------
--.<---.>+++.-------
.++++++++++++++++++++++++.<++++++++++++++++++.+++++.----------
--------.++++++++++++++++.++++++++.++.----------.>----------.------
---------.++++++++++++++++++++++++++++.-------.<----.>------
.++++++++++++.----------.<..>-------.++++++++++++++++++++.----------
-.<++++.>+++++++.---------.++++++++++++++++++.----------
.++.+++++++.<++++++++++++++++++.>>++.”

This is encoded using BrainFuck algorithm.