Planning Document

Project Scotch

James Parkington

There has never been a more exciting time to be a chess enthusiast. In recent years, more and more prominent chess thinkers, grandmasters, and commentators have taken to platforms like YouTube and Twitch to stream their live games, talk about interesting positions, and even create experiments with the many bots being developed and released to bend our notions of how the game works. Streams where prominent players like Hikaru Nakamura, Daniel Naroditzky, and even top-ranked player Magnus Carlsen provide detailed descriptions of their techniques attract similar viewership to some of the top gamers in the world.

That huge increase in readily-available chess content presents individuals from all skill levels with the opportunity to improve their play, but it can be challenging to find commentary on a particular position without knowing exactly where to look.

My vision is to meet that challenge with a free online chess tool that would provide users with immediate access to top-level commentary on any position you enter into the tool. After supplying a board orientation, the tool would then crawl its database of YouTube videos, Twitch VODs, and any other free source of video-based chess material to return a queue of relevant embedded videos. This way, users would see how a professional would continue from their position and learn why.

However, rather than supplying content on just a specific position, it would be great if the tool could proactively try to understand how a player arrived at that position, so that the analysis can be even more contextual.

Problem Description

To build such a tool, it is necessary to have a performant way to not only store thousands of permutations of chess positions, but also to render them quickly on a chess board.

Fortunately, all of today's major chess-playing platforms offer something called a PGN game export for every public game played on their system.

In short, with this project I'll be attempting to build a Python program that:

1. Parses PGN files and breaks them down into each "move" in a game from start to completion

2. Converts those moves into a performant data type that contains the state of the 64-square board (*my first thought is a 64-bit string, but there will be some discovery for this step)*
3. Store each bit string in a sequential way for easy retrieval
4. Render each bit string on a chess board that can be navigated left and right, ideally be graphical interface or user input

Depending upon how possible these first four steps are, I'll then devise of a way to loop through patterns of moves from prominent games to see if any of those states match any states the user supplied in their initial PGN file.

Additional Background Information

PGN stands for Portable Game Notation and is a standard format for representing chess games. A PGN file contains a series of moves and other game data, such as the names of the players, the date of the game, and the result of the game. PGN files can be used to replay and analyze chess games.

```
[Event "Tilburg Interpolis"]
[Site "Tilburg NED"]
[Date "1985.10.13"]
[Round "1"]
[White "Kasparov, Garry"]
[Black "Sosonko, Gennadi"]
[Result "1-0"]

1.e4 e6 2.d4 d5 3.Nc3 Bb4 4.e5 c5 5.a3 Bxc3+ 6.bxc3 Qc7 7.Qg4 f5 8.Qg3 Ne7
9.Ne2 O-O 10.h4 cxd4 11.cxd4 Qxc2 12.h5 Bd7 13.h6 g6 14.Bg5 Nbc6 15.Rc1 Qa4 16.Qh4 Rae8
17.Rh3 Rf7 18.Rhc3 Nc8 19.Bf6 Nb6 20.Nf4 Rxf6 21.Qxf6 Re7 22.Nxg6 hxg6 23.Rg3 Be8 24.h7+
1-0
```

This is an example of the standard PGN format used to record chess games. Each file contains the following metadata at the start:

- The name of the event
- the date it was played
- the players' names
- result of the game

The moves of the game are then recorded in sequential notation, where each move consists of the piece moved and the square it is moved to. In this format, moves are numbered and any capture is signified with an **x** marker. Handling this type of notation in a way that returns an accurate board state will be the brunt of the project. As you can see, each move consists of a turn for both black and white, which specifies which piece arrived at the square. You do not

receive any context about the rest of the board. Therefore accurately presenting the board state after each move requires some knowledge of how each piece can legally take another.

Challenges

Short-Term

- Retrieving PGN games from an open-source database via API
- Building a free retrievable database within the confines of Jupyter Notebook that stores the results of many of these conversions
- Deciding on a notation to use for each of the 64 squares from move to move
- Designing a means of referencing a non-Python file as the input for the program

Long-Term

- Build out a methodology for tracing a position backwards to guess what the most likely opening(s) could've been for any position
- Explore OCR possibilities for video content
    - Ideally there's a tool that can recognize the shapes of a board and its pieces on-screen, which I can then render in the same notation for the embedding idea
- Test different sorting methods to try to find matches as quickly as possible from a given database of board states

Smaller Problems to Decompose

There are 2 different major sub-functions needed to achieve the short-term goals of this project, and those 2 sub-functions should operate in isolation of one another to meet different needs.

`convert_pgn(file)`: One of the sub-functions should be the actual conversion of PGN notation into the preferred bit-string design. Not only will this be crucial for the interactive part of the program, but it'll also be necessary at the start for building the database of games with their board states from move to move.

I'd love for the user to have the option to either supply a file or paste a snippet in. In either case, I'll need to design some guardrails for user input to prevent users from supply positions that can't exist on a chess board.

`match_existing_games(game)`: After deciding on a notation and a storage solution, this function would theoretically loop through every move of the user's game and try to match it to every move in the database. The challenge will be minimizing the amount of resources consumed by the matching algorithm to find plausible continuations.

From there, I'll likely write some functions specifically designed to return to the user a "best match", with information about that best match that the game occurred in, and any additional information I can readily glean about the position. For example, what opening did

the winner play? How many moves did the professional game have? Was there a continuation you could've tried if perhaps you resigned earlier than the professional would have?

Fortunately, there are plenty of open-source databases available for retrieving PGN games, in order to build out this database. In fact, instead of building my own, if the conversion algorithm is strong enough, I might be able to do the conversion in-place by querying one of these sources in an intelligent way repeatedly.

1. **The Chess Games Database** - This is a free online database that provides PGN files for over 500,000 games, including games from some of the top tournaments and players.
2. **The Million Base** - The Million Base is a collection of over 2 million PGN games, which can be downloaded for free from several sources online.
3. **Lichess** - Lichess is a free online chess platform that maintains a database of millions of PGN games, which can be accessed and downloaded by their users.

Quick Mock-up

This code is not operational, but it contains some of the broad ideas I'd like to achieve with this project.

```
pgn = open("game.pgn").read() # Load a PGN game from a file

board = [[None] * 8 for _ in range(8)] # Initialize a new chess board
pieces = {'P': '♙', 'N': '♘', 'B': '♗', 'R': '♖', 'Q': '♕', 'K': '♔',
          'p': '♟', 'n': '♞', 'b': '♝', 'r': '♜', 'q': '♛', 'k': '♚'}

# Convert a piece symbol to a 4-bit binary string representation
def binary_pieces(piece):
    unicode_pieces_as_strings = list(pieces).index(unicode_version)
    return unicode_pieces_as_strings

# Loop over each move in the game and convert the end state of the board to a 64-bit string
for move in pgn.split()[1:]:

    # Convert the board to a 64-bit string
    bit_string = ""
    for rank in range(8):
        for file in range(8):
            piece = board[rank][file]

            if piece is None:
                bit_string += "0000"
            else:
                bit_string += binary_pieces(piece)

    # Print the bit string for the current position
    print(bit_string)
```