Norges Teknisk-Naturvitenskapelige Universitet

TPK4186 - Advanced Tools for Performance Engineering
Spring 2023

Assignment 2: Chess Games

Prepared by Antoine Rauzy

Contents

1 Introduction

1.1 Presentation of the Problem

Data bases of chess games are widely available on internet. For example the website http://www.computerchess.org.uk/ccrl/4040/games.html provides thousands of games played by chess engines. The joined archive `Stockfish_15_64-bit.commented.[2600].pgn.gz` contains 2600 games played by Stockfish 15, one of the best chess AI available.

The objective of this assignment is to design a Python script that loads these games, perform various statistics on them, a display the results at a suitable format.

In archives, games are stored at the pgn (portable game notation) format. Figure 1 shows the encoding of the first game of the archive.

The encoding of a game consists of two parts.

The first part, from line 1 to line 12 contains meta-data on the game. The second part, from line 14 till the end, encodes the game itself.

The game consists of a series of moves and terminates with the result of the game, `1-0` in this case, which means that white pieces have won. The encoding of a move consists of:

– The number of the move followed by a dot.

– The move of white pieces possibly followed by a comment within braces.

– The move of black pieces possibly followed by a comment within braces.

```
[Event "CCRL 40/15"]
[Site "CCRL"]
[Date "2022.04.20"]
[Round "820.1.525"]
[White "Stockfish 15 64-bit"]
[Black "Arasan 23.3 64-bit"]
[Result "1-0"]
[ECO "A03"]
[Opening "Bird's opening"]
[PlyCount "99"]
[WhiteElo "3511"]
[BlackElo "3333"]

1. f4 {+0.00/1 0s} d5 {+0.00/1 0s} 2. b3 {+0.00/1 0s} Bg4 {+0.00/1 0s} 3. Bb2
{+0.00/1 0s} Nc6 {+0.00/1 0s} 4. h3 {+0.00/1 0s} Bh5 {+0.00/1 0s} 5. g4
{+0.00/1 0s} Bg6 {+0.00/1 0s} 6. Nf3 {+0.00/1 0s} h5 {+0.00/1 0s} 7. g5
{+0.00/1 0s} e6 {+0.00/1 0s} 8. e3 {+0.00/1 0s} Nge7 {+0.00/1 0s} 9. Bb5
{-0.36/32 45s} a6 {(h4) +0.59/28 20s} 10. Bxc6+ {(Bxc6) -0.24/30 10s} Nxc6
{(Nxc6) +0.62/28 18s} 11. d3 {(d3) -0.28/29 10s} Qe7 {(h4) +0.74/28 18s} 12.
Qe2 {(Qe2) -0.12/30 17s} O-O-O {(O-O-O) +0.92/27 18s} 13. Qf2 {(Qf2) -0.12/32
29s} f6 {(f6) +0.50/26 20s} 14. Nh4 {(Nc3) -0.07/31 9s} Bf7 {(Bf7) +0.64/28
21s} 15. gxf6 {(gxf6) -0.04/32 14s} gxf6 {(gxf6) +0.59/30 21s} 16. Nc3 {(Nc3)
-0.12/32 13s} Qe8 {(Qd7) +0.43/27 21s} 17. O-O-O {(O-O-O) +0.00/29 19s} e5
{(Rg8) +0.43/25 21s} 18. Rdf1 {(Rdf1) +0.00/33 11s} Be6 {(Be6) +0.23/27 19s}
19. f5 {(Kb1) -0.06/35 65s} Bf7 {(Bf7) +0.14/33 21s} 20. Kb1 {(Ng6) +0.00/34
30s} Rg8 {(Bc5) +0.39/27 42s} 21. Rhg1 {(Rhg1) -0.15/32 8s} Rxg1 {(Bc5)
+0.41/27 22s} 22. Rxg1 {(Rxg1) +0.00/31 12s} Bc5 {(Bc5) +0.34/27 39s} 23. Ng6
{(Bc1) +0.00/36 29s} b6 {(Qd7) +0.07/29 38s} 24. Na4 {(Ne2) +0.06/35 19s} Bd6
{(Bd6) +0.00/29 17s} 25. Nc3 {(Nc3) +0.00/35 10s} Bc5 {(Bc5) +0.00/32 17s} 26.
Bc1 {(Na4) +0.00/40 56s} Qd7 {(Ne7) +0.04/26 17s} 27. Qf3 {(Qf3) +0.00/31 9s}
Kb8 {(Kb8) +0.07/28 17s} 28. Qxh5 {(Qxh5) +0.31/32 16s} Nb4 {(Nb4) +0.17/30
17s} 29. Na4 {(Na4) +0.01/36 26s} Bd6 {(Bd6) +0.16/30 15s} 30. Qg4 {(Bd2)
+0.00/36 33s} Qc6 {(d4) +0.00/25 21s} 31. Qg2 {(Qg2) +0.70/27 9s} b5 {(b5)
+0.37/29 17s} 32. Nb2 {(Nb2) +0.55/30 18s} Qb6 {(Qb6) +0.00/32 17s} 33. Bd2
{(a3) +0.56/31 14s} Bc5 {(a5) -0.46/29 31s} 34. c3 {(Rf1) +1.68/27 13s} Nc6
{(Nc6) -1.05/30 18s} 35. d4 {(d4) +1.52/28 19s} Bd6 {(Bd6) -1.24/24 14s} 36. h4
{(h4) +1.84/28 14s} b4 {(exd4) -1.90/26 13s} 37. dxe5 {(dxe5) +1.77/30 15s}
Nxe5 {(Bxe5) -2.07/26 13s} 38. cxb4 {(cxb4) +1.91/28 15s} d4 {(Nc6) -2.27/28
12s} 39. e4 {(e4) +2.62/28 18s} d3 {(d3) -2.82/28 12s} 40. Rd1 {(h5) +3.13/26
10s} Nc6 {(Nc6) -3.31/25 14s} 41. h5 {(Nxd3) +3.39/25 18s} Bxb4 {(Bxb4)
-3.94/27 24s} 42. Bc1 {(Bc1) +3.50/27 18s} d2 {(d2) -4.34/29 19s} 43. Bxd2
{(Bxd2) +4.00/28 16s} Ba3 {(Bxd2) -4.15/30 16s} 44. Qe2 {(Qf3) +4.17/24 14s}
Kb7 {(Kb7) -5.16/24 17s} 45. Bc1 {(Be3) +4.66/26 20s} Bxb2 {(Bxb2) -5.46/23
16s} 46. Kxb2 {(Kxb2) +4.68/26 25s} Rxd1 {(Bc4) -7.71/27 21s} 47. Qxd1 {(Qxd1)
+5.98/27 18s} Qf2+ {(Qf2) -7.85/32 32s} 48. Qd2 {(Qd2) +6.79/28 26s} Qg1 {(Qc5)
-8.42/35 15s} 49. h6 {(h6) +7.51/26 18s} Bxg6 {(Bg8) -8.21/38 15s} 50. fxg6
{(fxg6) +8.93/27 30s} 1-0
```

Figure 1: Encoding of a chess game at pgn format

1.2 Requirements

The objective of this assignment is to show your Pythonic skills.

Here follows a number of requirements.

1. You must provide your program together with a small document explaining how it is organized, what it is doing (which functionalities are implemented) and reporting experiments you have performed with it.

2. The assignment can be made either individually or in group (maximum 3 persons, preferably 2).

3. Assuming you named your group $Python's fan$, all the files you deliver must be included in a zip archive named:

   `TPK4186 - 2023 - Assignment 2 - Python's fan.zip`

   The deliverable of the assignment is this zip archive. Please recall your names in addition to the group name both in the header of the program and in the accompanying document.

4. The quality of a program is judged along three criteria: its completeness, its correctness and its maintainability:

   – A program is complete if it provides all functionalities demanded by the client. Some functionalities are however more important than other. You must first concentrate on the main functionalities, then develop the "nice-to-have" ones.

   – A program is correct if it is bug free. To ensure that your program is correct, you must test it extensively. Design tests before writing the first line of code. There is no such a thing than a program or a functionality that works "most of the time". Either it works, or not. If you are not able to make a functionality work, do not deliver it.

   – A program is maintainable if it is well presented, if the identifiers are significant, and so on. But before all, a program is maintainable if it well organized and as modular as possible. Separate the concerns.

2 Tasks

For this assignment you are asked to design several data structures and to implement quite a few functions. Each of these functions must be thoroughly tested.

The assignment consists in the following tasks.

2.1 Games

We shall first implement data structures and functions to manage games.

Task 1. Design a data structure to encode games and implement associated the set and get functions. Note this implies managing moves.

Task 2. Design functions to import a game from a text file.

Task 3. Design functions to export a game from a text file.

Task 4. Design a data structure to manage a data base (a list) of games. Extend your reader and your printer so to manage data bases.

Task 5. Design functions to export and import a game from an Excel spreadsheed.

2.2 Statistics

The results of the following tasks must be exported in a Word (or equivalent) document. HTML pages or LaTeX documents are accepted as well.

Task 6. Design functions to create a Word (HTML, LaTex) documents. This document must have a title, some sections, subsections, paragraphs... It will include tables and figures (plots of distributions).

Task 7. Design functions to extract the number of games won, drawn and lost by Stockfish in total, then with white pieces, then with black pieces. Design functions to insert a table with these results in your document.

Task 8. Design functions to plot the proportion of games still on-ongoing after 1, 2, 3... moves. Design functions to insert a figure plotting this distribution in your document.

Calculate also the mean and the standard deviation of the number of moves of a game. Design functions to insert a table with these results in your document.

Same questions for games played by Stockfish with white pieces and with black pieces. Note that it may be of interest to plot the three curves on the same figure.

Same questions for the games won by Stockfish and those Stockfish lost.

2.3 Openings

We want now to make statistics on openings. To do so, we need to build a dedicated data structure, in form a of tree, as illustrated in Figure 2. Each internal nodes of the tree contain:

– The player who has the turn.

– The list of moves that have been played in this position. Each of these moves corresponds to a branch of the tree.

– Meta-data such as the number of games won by white pieces, by black pieces and the number of games that ended up in a draw.

Leaves of the tree encode the meta-data associated with games.

Task 9. Design a data structure (and all its management functions) to encode a tree such as the one pictured in 2.

Task 10. Design functions to parse a data base file such as `Stockfish_15_64-bit.commented.[2600].pgn.gz` and load it into an opening tree.

Task 11. Design functions to print an opening tree at a given depth, i.e. number of 1/2 moves. The idea here is to get the number of games won by white pieces, by black pieces and the number of games that ended up in a draw in each opening. Design functions to insert your drawing in the report.

Task 12. The drawing of the previous question is only moderately interesting from a chess point of view: some openings are much more played than others. E.g. the Italian, Spanish and Sicilian defenses on `1. e4`. For these openings, we want to go deeper, while on rare openings such as the Bird (`1. f4`), one or two 1/2 moves suffice.

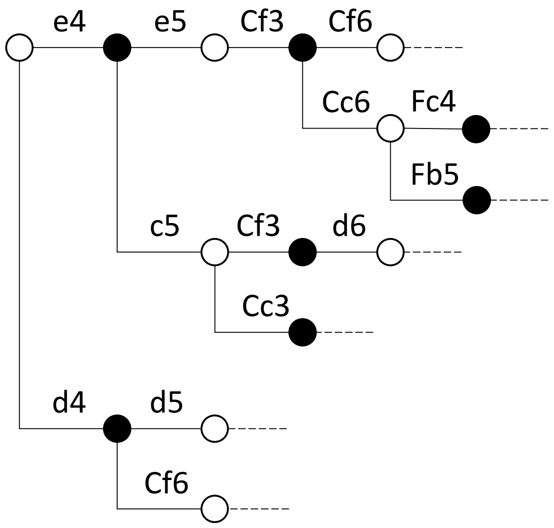

Figure 2: Tree of chess openings

Design functions to extract openings that have been played more than n times, where n is given by the analyst. Design functions to store these openings in a table, with, for each of them, the number of games won by white pieces, by black pieces and the number of games that ended up in a draw. Design functions to insert this table in your report.