Rollerball (Phase II)

(COL333 Assignment 5)

COL333 TAs

September 2023

1 Goal

The goal of this assignment is to create a bot to compete and win at three variants of the game of Rollerball

2 The Game of Rollerball

2.1 Starting Position

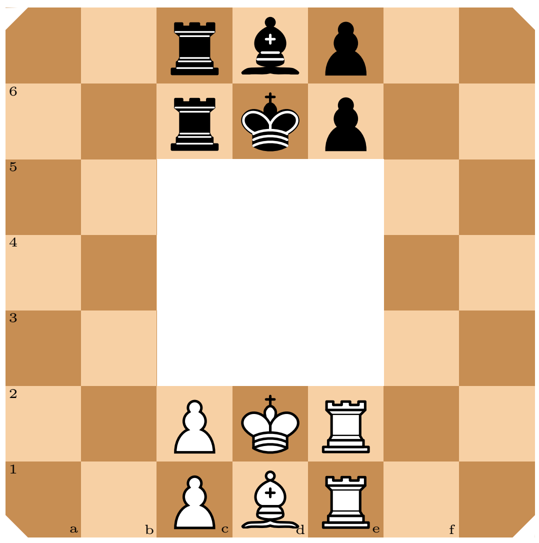


Figure 1: Starting position for Board 1: 7_3 rollerball

This assignment edition features three different board sizes with different starting configurations of the pieces, as described in Figures 1, 2, 3. We introduce a new piece, the knight in Board 3.

The game proceeds similar to chess, with the objective being to checkmate the opponent king. Further details are described below.

2.2 Movement Rules

Due to four-fold symmetry of all the boards, we shall only describe the rules over one of the four segments, for the knight. The movement of all other pieces is similar to that in the previous edition of the assignment.

2.2.1 Knight

The Knight moves like it does in traditional chess, given that the move is valid. It can move in both the clockwise and anti-clockwise directions.

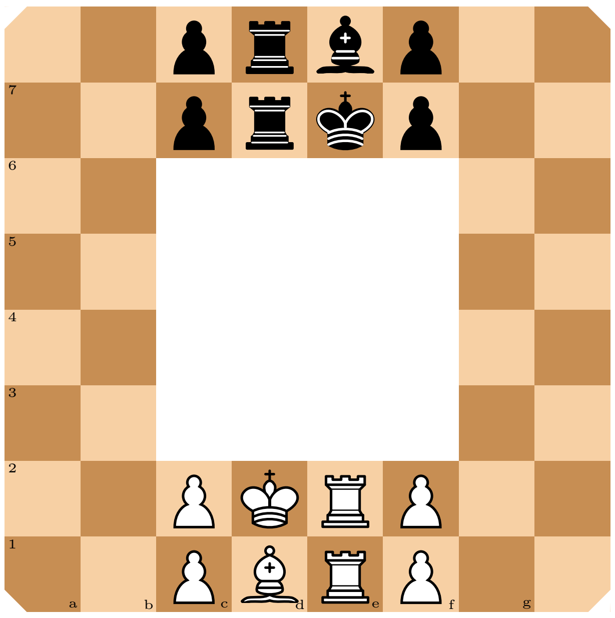


Figure 2: Starting position for Board 2: 8_4 rollerball

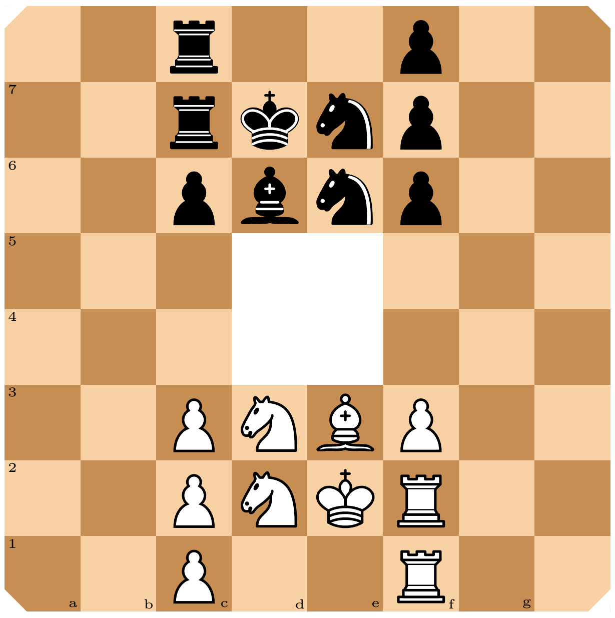


Figure 3: Starting position for Board 3: 8_2 rollerball

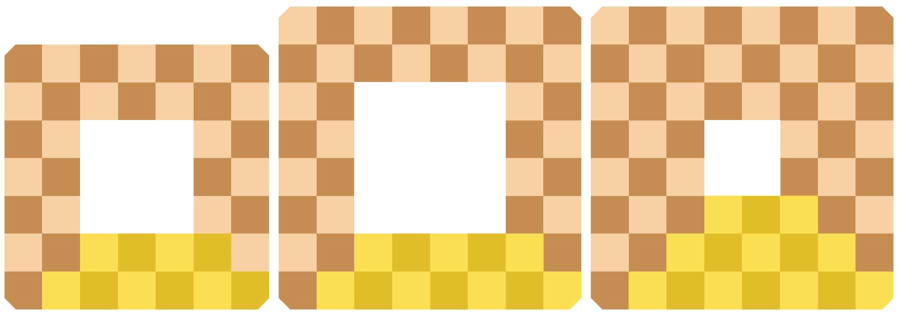


Figure 4: Four-fold symmetries of the boards, and squares over which moves are described

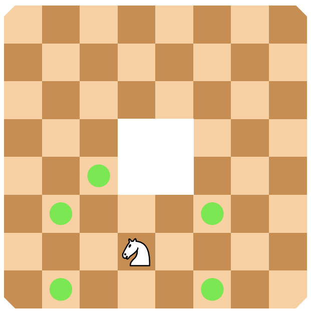
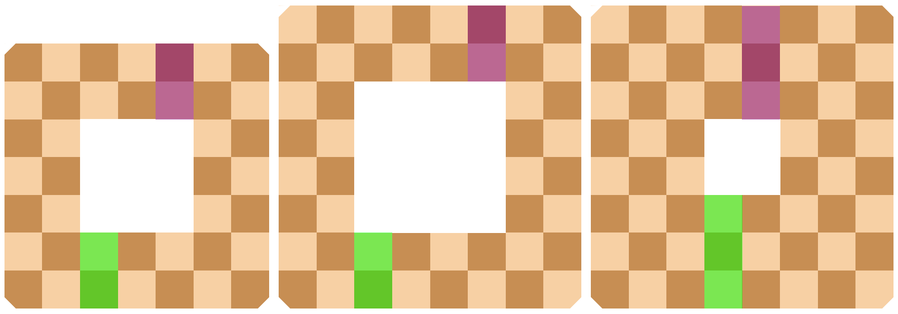


Figure 5: Knight move example

2.2.2 Pawn Promotion Rules

Pawns are promoted when they reach the following squares on the corresponding boards (Violet indicates squares for white pawn promotion, and green indicates squares for black pawn promotion). Pawns can only promote to Rooks or Bishops, even in the 8_2 board.



2.3 Objective

We will play three tournaments of Rollerball with each of the three board configurations. In each **battle** of the tournament, two bots will compete against each other in two **matches**, once as the white player and once as black. There will be two components to the scoring of each match. A **Discrete Score** and a **Continuous Score**. Details of scoring in the tournament, battles and matches are mentioned in the sections below.

3 Scoring

In this edition, each bot has a *total time pool* to allocate among the different moves as per its choice. The time duration per move is not fixed. Due to this design choice, you can store state information across different moves in this assignment and use history to make informed decisions about time allocation for different stages in game.

The game ends when at least one of the following conditions holds true for the current board state.

1. There is a checkmate. In this case, the King that gets the checkmate loses, and the other player wins (as in classic Chess).
2. Threefold repetition of moves from both the players. Please read this for details of the rule. This condition ends in a draw.
3. Stalemate. If neither of the player is able to play a move on the board, it results in a draw.
4. Time for one of the players runs out. This player loses and the other player wins.

3.1 Evaluation

Your final submission will be based on a tournament setting. Your developed bots will play against each other in a round-robin group stage followed by a knockout stage. Details for the tournament structure are mentioned later in the doc. Your final score will be based on your final standing in the tournament. Each match in a tournament will comprise two games: 1 time as black and 1 times as white. Deterministic code is strongly recommended, since we will only be playing one game on either side – you don't want a low probability sample to ruin your game. The score for your match will be a combination of Discrete Score and Continuous Score (think of it as NRR in cricket). The Discrete Score will primarily determine a winner, but the Continuous score will break ties in the match and group stage.

3.1.1 Discrete Score

$\mathbb{1}_{[X]}$ is the identity function for outcome X holding for a player.

Discrete Score $= \mathbb{1}_{Win} + 0.5 * \mathbb{1}_{Draw}$

3.1.2 Continuous Score

Continuous Score = Victory Score + Margin Score

3.1.3 Victory Score

The losing player gets a victory score of 100 - Victory score of the winning player. If the game draws, each player gets a victory score of 40 (this is to discourage drawing of games). Let the number of moves played by the winning player be n. Then, the victory score for either player is computed as follows. ($\mathbb{1}_{[X]}$ is the identity function for outcome X holding for a player.)

Victory Score $= \ \mathbb{1}_{[Win]}100 + (\mathbb{1}_{[Lose]} - \mathbb{1}_{[Win]})(5\lfloor \frac{n}{20} \rfloor + \min\{10, n\}) + \mathbb{1}_{[Draw]}40$

You can plot and see how the Victory score varies as a function of the number of moves it takes to win to better understand the scoring metric.

3.1.4 Margin Score

Each piece on the board (except the Kings) is allotted a score as in classic chess. The following table summarises the score value for each piece type. At the end of the game, the sum of the scores for the remaining pieces is calculated for each player. This is **Margin Score** for the player.

Note that even the losing player gets assigned a Margin score. So if one makes a better judgement of their position at any game stage, they can protect their pieces and let the other player win to attain a Margin score. On the other hand, the winning player can sabotage the losing player by taking as many pieces as they wish.

Piece Type	**Score**
Bishop	5
Rook	3
Knight	3
Pawn	1

4 Tournament Structure

We will have three tournaments, one for each board type. The marks will be equally distributed for the three tournaments. Each tournament comprises of the following two stages.

4.1 Group Stage

All teams will be distributed among 16 groups, $G_1...G_{16}$. Each group G_i consists of the teams ranked $i + 16k, k \in \mathbb{W}$ in A2. Each team plays a battle with every other team within a group. Top two teams in each group move on to the knockout stage.

Each battle comprises two matches, and the scoring of each match is as mentioned in the scoring section. Each player is allocated a total time of 2,3,4 minutes per match for tournaments over boards 7_3, 8_4, 8_2 respectively. Add the discrete and average continuous scores to arrive at the final tuple of scores per team within a group. We order the teams based on the discrete scores and use continuous scores for tiebreaks. Although unlikely, for further ties, we look at the performance in A2. For further ties, we toss a coin. The top two performing teams per group move to the knockout stage.

4.2 Knockouts

Let $T - j - i$ be the team that ranked j in the i^{th} group. The knockouts start with the following game tree, each winning team moving up the ladder.

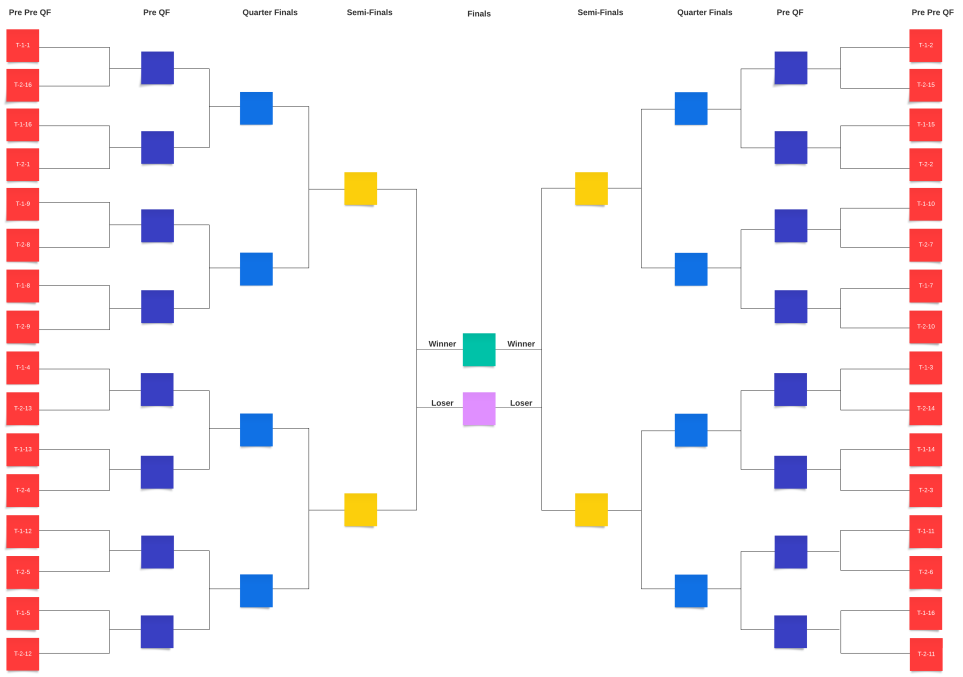


Figure 6: GameTree

Wins are decided in the same way as the group stage. However, in case of a tie in both the discrete and the continuous scores, we play another battle but with a time allocation of 1 minute for each player, and the decision is taken based on the discrete and continuous score of this battle. If this battle also ties in, we move on to a battle of 30 seconds for each player. For further ties, we look at the performance of group stage, and incase it is same we check scores of A2. For further ties, we toss a coin.

5 Tournament Evaluation

Each tournament is worth 4 points and is graded independently. For each tournament, we have the following score allocation.

- Winner = 12 points (won't play other tournaments)
- Runner up = 4 points
- Second Runner up = 3.6 points
- Third Runner up = 3.5 points
- Losing Quarter Finals = 3.2 points
- Losing Pre Quarter Finals = 2.8 points
- Losing Pre-Pre Quarter Finals = 2.4 points
- 3rd in the group = 2 points
- 4th in the group = 1.66 points
- 5th in the group = 1.33 points

- 6th or 7th in the group = 1 points

6 Starter Code Overview

We host the starter code on GitHub. The link to the repository can be found on the course webpage. Rollerball's GUI is implemented in JavaScript, and the core engine is written in C++. A similar interface is used as Assignment2. Note that *for this assignment, we are only accepting solutions in C++.*

6.1 Requirements

1. gcc $\geq$ 11
2. python $\geq$ 3.7

6.2 Quickstart

```
git clone https://github.com/Aniruddha-Deb/rollerball-v2 && cd rollerball-v2
make rollerball
```

If all goes well, you should have an executable called `rollerball` in `bin`. To run the GUI, launch a web server from the `web` directory.

```
cd web
python3 -m http.server 8080
```

You can then open localhost:8080 on your browser to view the GUI. Here you can select one of the three board types.

To launch the bots (assuming you're in the directory)

```
./bin/rollerball -p 8181
```

You can then connect the GUI to the bots. You would also need to start another bot for black on port 8182 to join and start the game.

6.3 Web UI Changes

For this iteration, we have provided the source code for the Web UI as well. Those interested in developing/modifying this may do so. The UI is written in Vue, and contains a small README in the `websrc` directory that will help you in getting started. Note that **The TAs are not responsible for any bugs you may encounter while changing the UI code.** Posts on Piazza regarding questions about any files or modifications in `websrc` will not be answered.

- The Web UI (and engine) now support **Starting new games without reloading the UI / restarting the engine(s)**.
- The UI also supports starting/stopping games and disconnecting/reconnecting from bots.
- The clock in the UI runs at a resolution of 10 ms, and may not be as accurate as the clock we use when evaluating your engines. Please use this clock as a guideline, and not as a benchmark.
- The three board sizes can be changed using the buttons below the board, when a game is not in progress.

6.4 C++ Binding changes

Board:

- **All the methods in the Board class have been made public** for students to use.
- **Methods that modify Board state are suffixed with an underscore** (_). This means that `do_move` is now `do_move_`. Other methods modifying state have been made public and follow the same convention. You may need to change your code as a result of this.

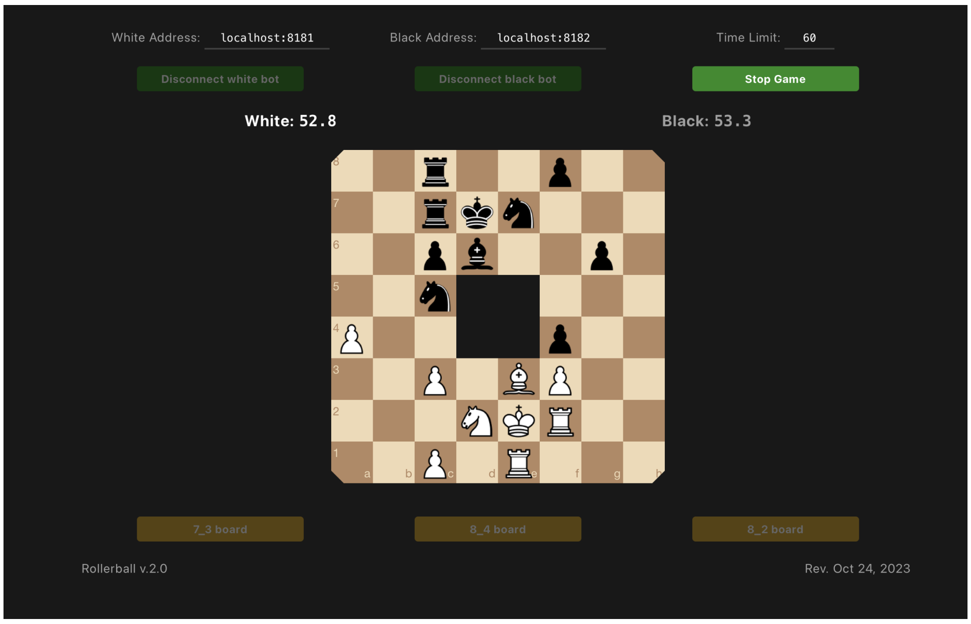


Figure 7: A screenshot of the UI in action (8_2 board)

- **Copy constructors have been implemented for Board and BoardData**, and the `copy` method has been removed.
- Move generation has been generalized across multiple boards. This may lead to the generation being slightly slower than it was previously. You are advised to make changes taking this into account.
- BoardData and board utility methods have been split off into their own header files to clean up the code

Engine:

- **engine.hpp can now be edited**, allowing students to store state across multiple invocations of their Engine. Note that this inherits from **AbstractEngine**, so you cannot remove or edit the signature of the `get_best_move` method.
- Due to managing time, the engine class now has a `time_left` field, which stores the time left (in ms). This is synced with the arbiter's clock whenever the arbiter asks the engine to find the best move. Note that a timeout on the arbiter's end implies that you lose, irrespective of the time on the clock you maintain.

BoardData:

- Additional fields have been added for extra pieces, such as for knights and extra pawns, and the suffixes (`ws,bs`) have been renamed to `1,2,3,4`.
- A field called `board_type` indicates the type of the board: the type may be one of `SEVEN_THREE`, `EIGHT_FOUR` or `EIGHT_TWO` corresponding to the boards described above.
- BoardData now has a corresponding board mask, indicating the squares on the board which are valid. These can be seen in `constants.hpp`, along with macros.
- BoardData also stores the pawn promotion squares in an array `pawn_promo_squares`. This is used for move generation.

Other Changes:

- The `DEAD` constant has been changed from `pos(7,7)` to `0xff`
- Printing/debugging functions have been moved to `butils.hpp`
- Move generation has been refactored completely, and should have fewer bugs now

- Documentation for methods can be found in the respective .hpp files

6.5 Implementing the Engine

You would need to implement the `get_best_move` method in `engine.cpp`. This method will be called on an engine object when the server decides to search for a move, and it should do the following:

- Search for the best move given the board type, and store this best move in `best_move`.
- Terminate before the time specified in `time_left` runs out.
- Not modify the board passed to it (Note that the board is declared `const`). You may make a copy of the board and modify that if needed.

The starter code's engine.cpp randomly picks a move from the moveset and sets the best move to it, as an example. This time, you are free to edit both 'engine.cpp' and 'engine.hpp'. The Engine class inherits from AbstractEngine class, and as long as you do not change the function signature of `get_best_move`, you are free to make edits in these two files.

6.6 Other details

- You are **not allowed** to use any libraries other than those that come with the language. This means that libraries such as boost are not allowed.
- You are only allowed to modify and submit `engine.cpp` and `engine.hpp`
- Your code must compile with the Makefile provided **without any modifications**.
- Your implemented algorithm should be **single-threaded**. You are not allowed to make use of multiple threads, asyncio or any other form of parallel execution.
- You are allowed to use any and all techniques subject to these constraints (including but not limited to value functions, tablebases, neural networks, MCTS, distillation, learning via Self-Play etc). Note that the assignment is competitive, and the TA bots are not necessarily the best benchmark to optimize against :)

7 Submission Details

- Submit a single zip containing 3 files: 1.) Readme.md and 2.) 'engine.cpp' 3.) 'engine.hpp'. In 'engine.cpp', you are allowed to write board-specific code conditional on the `board_type` flag.
- The zip file should be named 'entrynum1_entrynum2.zip' and running 'unzip file_name.zip' should extract the files in current directory.
- The 'Readme.md' must detail: a.) Name and entry number of teammates b) Names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None. c.) A small writeup on the methods used (optional).
- You can use verification server to make sure your submission confirm with the evaluation protocol. Check piazza for updated details.
- Submission Deadline is **11:59pm, 17th November** on Moodle. Any updates will be posted on Piazza.

8 General Guidelines

- You must use either the same partner as in Phase 1. Or you may work alone. In case your partner has withdrawn or has audited (and does not want to submit the assignment) you can find *another such* partner, and post it on Piazza in the dedicated Note for it. In either case, you must make sure that your code is not caught in plagiarism against other codes in this assignment.
- You are allowed to use only C++

- Your code must be your own. You are not allowed to use **ChatGPT** or any **coding assistant** in general for this assignment. Note: It is not very difficult to catch such assistants :)
- Note that the GUI takes a server address to connect to, so it is possible to set up a hotspot and connect to your friends' bots to play against them. While we encourage collaboration in this manner, we strongly condone cheating. Please see the guidelines between collaboration and cheating at `https://www.cse.iitd.ac.in/~mausam/courses/col333/autumn2023/`.
- You must not discuss this assignment with anyone outside the class. Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team. Please read academic integrity guidelines on the course home page and follow them carefully.
- You get a zero if your player does not follow the submission guidelines in this document.
- We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.