Technical Report: Final Project
EECE 2560: Fundamentals of Engineering Algorithms

Jaden Mack and Troy Hutchinson
Department of Electrical and Computer Engineering
Northeastern University
mack.ja@northeastern.edu hutchinson.t@northeastern.edu

December 5, 2024

Contents

1 Project Scope

The aim of this project is to develop a system that tracks the skill level and records of players within a tournament for a generic game, similar to the Glicko score system for fighting games or the Chess ELO system.

The project's main objectives are:

- To allow for players to be entered into a system that will track their rating between tournaments
- To allow for tournaments to be recorded via matches between players, with independently tracked records
- To give live updates to records in a scoreboard both for individual tournaments and via the MMR score

The expected outcomes include a functioning ranking-based system, proper documentation, and a final project report.

2 Project Plan

2.1 Original Timeline

The overall timeline for the project, optimistically, will attempt to meet that of the sample report, seen below. We plan to have more issues with the front-end than anything else so we will be allotting more time to that, and earlier:

- **Week 1 (October 7 - October 13)**: Define project scope, establish team roles, and outline skills/tools.
- **Week 2 (October 14 - October 20)**: Begin development, set up the project repository, and start coding basic system functionalities.
- **Week 3 (October 21 - October 27)**: Continue coding, work on backend integration and start the frontend design, and start writing technical documentation.
- **Week 4 (October 28 - November 3)**: Complete the backend and integrate frontend elements. Begin PowerPoint presentation.
- **Week 5 (November 4 - November 10)**: Finalize the system, conduct testing, and continue with the report. Try to test for ease of use with a friend to ensure universality.
- **Week 6 (November 11 - November 17)**: Revise and finalize the technical report and the PowerPoint presentation.
- **Week 7 (November 18 - November 28)**: Finalize Algorithm, UI

2.2 Original Milestones

Key milestones include:

- Project Scope and Plan (October 10).
- GitHub Repository Setup and Initial Development (October 10).
- Backend Completion (October 28).
- Frontend Completion (November 5th).
- Final System Testing and Report Draft (November 10).
- Final Presentation and Report Submission (November 28).

2.3 Team Roles

- **Jaden Mack:** Responsible for front-end and QT design and back-end framework
- **Troy Hutchinson:** Responsible for back-end C++ design and testing

3 Methodology

3.1 Pseudocode and Complexity Analysis

The main reporting function, reportMatch(), has a time complexity of O(N) This is due to it being required to possibly parse through the full list of players twice in order to find both to verify their existence, and then to begin the recalibration step. This is done through the findPlayer() command, which is contained within the function. It then calls calibrateScore(), but this has a complexity of O(1) so it does not add any additional complexity. The function performs as follows:

reportMatch:

```
If either player does not exist, end function
If Player1 won, give him a win and the opponent a loss, then call calibrateScore
If Player2 won, give him a win and the opponent a loss, then call calibrateScore
If neither player won, then give them each a tie
After this, resort the players
```

calibrateScore:

```
Calculate the odds of winning via the Bradley-Waite Model
Find the difference in score between the players
Determine the shift of score each player should individually experience
Return a pair with the new values of each player
```

The sorting algorithm for the players, done via quicksort through the std:sort function, is O(N * log(N)). As this was not an original function, pseudocode is omitted.

The file based importing and exporting is each O(N) complexity, as for each it merely parses through the file and imports each individually or goes through the list and exports each individually.

importPlayers:

```
Open file, if it can't be opened, send error message
For each player, import them to the tournament. Then, output to the console their stats
```

savePlayers:

```
Open file, if it can't be opened, send error message
For each player, convert them to a string and send the string to a new line in the file
```

3.2 Data Collection and Preprocessing

Data is collected using an external text file. The file can be opened to directly edit the players between uses should adjustments need to be made. As such, there is little need for external data unless there is pre-existing players within a region or at a tournament, and in many of these cases the tournament organizer has the ability to rank the players by perceived skill initially

4 Results

The program worked fully as intended in the C++, as can be said minus some self-imposed limitations in QT.

4.1 Score Calibration

The scores calibrated seem to be accurate to other models, namely that of puddle.farm (fighting game specific Bradley-Waite based model) and chess.com (Elo system). The skills shift as expected save for outliers with freshly made players with large skill gaps who play many games, though this is almost an intended effect. It is discussed more later.

4.2 Sorting Method, File Input and Output

All of these functions are only available in the C++, but they function fully as intended, and do so quickly. There are errors thrown if the file being pulled from does not follow the proper syntax, but that is to be expected. The export also has the issue of not clearing the file prior to adding them, so you need to export to a freshly made .txt file

5 Discussion

The scoring algorithm is successful at granting a decently accurate adjustment of skill given two players with marked skill, though it has a few downfalls after testing.

- The program struggles to calibrate large score swings in early players. Should a new player be very good, and the user not input an accurate score, the resulting swing will be large. This should be mitigated by the older strong player, who has more games, experiencing the declined return, but the lower skill user may gain a much larger than intended amount. Testing can likely reduce this by capping maximum skill swing
- The program is limited only to games that have a single scoring indicator. This is one dimensional and could lead to errors in games with many metrics
- In games with longer sets and games with more variance, the multiplier on winning (currently x1 per win over the loser) should be reduced, possibly with a player-editable variable
- Ties may want to calibrate the score to be more even but this could feel unfair to the higher score player, possibly a one sided score calibration may want to be used. Bradley-Waite says to take points from the winner in that case, but this part was omitted for now

6 Conclusion

The program as a whole has been a success. As a long-time proponent against the Glicko system due to disagreements with how it is calibrated, it seems to be a suitable replacement for small-scale use. The ability to make it sync up online to create a wider database is intriguing, but not totally necessary in my eyes to function. It was nice to work with a UI for the first time, and it helped to give the program life and make it easier to troubleshoot. The algorithm, especially in QT, is however limited by its own functionality. A lot more could be added such as the ability to display brackets and a number of other data points that could be held better in a CSV database than a text file, but time constraints and a lack of knowledge in certain coding fields hampers that.

7 References

- Godden, Kurt. ”Elo to Glicko: Your Rating Explained.” Chess.com, 2009, https://www.chess.com/blog/kurtgodden to-glicko-your-rating-explained.
- ”C++ Standard Library.” cppreference.com, 2024, https://en.cppreference.com/w/cpp/standardlibrary.
- ”Error Handling During File Operations in C/C++.” GeeksforGeeks, 2023, https://www.geeksforgeeks.org/error-handling-during-file-operations-in-c-c/.
- ”C++ Standard Library Reference.” Microsoft Learn, 2022, https://learn.microsoft.com/en-us/cpp/standard-library/cpp-standard-library-reference.
- Huff, Connor. ”MTH 406 Final Presentation: Bradley-Terry Model.” YouTube, uploaded by Connor Huff, 4.6 years ago, https://www.youtube.com/watch?v=FJkwQIR2Vw0.

A Appendix A: Code

The QT code can be found at
https://github.com/MiggityWack/2560FinalProject/tree/main/QT%20Code%20Final

and the C++ code at
https://github.com/MiggityWack/2560FinalProject/blob/main/Troy%20updated%20code