Technical Report: Online Tournament Ranking System

Kevin Chen

December 5, 2024

1 Project Scope and Objectives

The objective of this project is to design and develop an online ranking system to efficiently manage and rank players in a tournament setting. This system allows for real-time updates to player rankings based on new match results, ensuring that rankings are both accurate and up-to-date.

The primary objectives are:

- **Efficient Ranking Updates**: Implement sorting algorithms to rank players based on scores and win rates in real-time.
- **Dynamic Match Handling**: Handle and update player statistics after each match.
- **Accurate Data Representation**: Maintain accurate and accessible data structures to store player information.

2 Algorithm Design

The ranking system is built on sorting algorithms and dynamic data updates, ensuring efficient processing of player statistics. Below is an overview of the core algorithms and techniques used:

2.1 Sorting Algorithm: Comparator-Based Sorting

Instead of manually implementing a sorting algorithm like Quick Sort, the system uses the STL `std::sort` function with custom comparators. This approach leverages C++'s optimized sorting libraries for efficiency and flexibility. Sorting can be performed based on:

- **Win Rate**: Primary criterion for ranking.
- **Ranked Score**: Tracks cumulative performance in matches.
- **Average KDA (Kills/Deaths/Assists)**: Evaluates player skill in a match.

2.2 Dynamic Match Handling

A function dynamically updates player statistics after each match. Key features include:

- Incrementing `matchesPlayed`, `kills`, `deaths`, and `assists`.
- Updating the win rate and ranked score for both players.
- Automatically re-sorting players by the selected ranking criterion after updates.

3 System Architecture and Data Representation

The system architecture follows a modular approach, with separate components for data storage, sorting, and match handling.

3.1 Data Structures

- **Player Structure**: Stores each player's attributes, including name, ID, match statistics (`matchesPlayed`, `wins`, `kda`, etc.), and ranked scores.
- **Vector Container**: The `std::vector` container is used to manage a dynamic list of players, enabling efficient operations like insertion, deletion, and sorting.

3.2 Flow of Data

1. **Player Initialization**: Players are either preloaded or added dynamically by the user.
2. **Match Results Input**: New match results are input into the system.
3. **Statistical Updates**: Player statistics are updated based on match results, including win rate and KDA recalculation.
4. **Sorting**: The player list is re-sorted based on the selected ranking criterion.
5. **Rank Display**: Rankings are displayed in real-time.

4 Development Challenges and Solutions

4.1 Real-Time Updates and Sorting

One challenge was ensuring the system could handle real-time updates to rankings after each match. This was resolved using the highly efficient `std::sort` function with comparators to dynamically sort players based on the selected criterion (win rate, ranked score, or KDA).

4.2 Dynamic Data Validation

Ensuring data integrity (e.g., preventing duplicate player IDs) required implementing a search function to verify player existence before adding or modifying player data. This validation reduces runtime errors and improves usability.

4.3 Managing Complex Player Statistics

Updating player statistics like KDA, win rate, and ranked score after each match required careful implementation of formulas and edge case handling (e.g., zero deaths or zero matches). The formulas were optimized to handle these cases.

5 Testing and Validation

The system was rigorously tested to ensure functionality and reliability.

5.1 Sorting Validation

The sorting feature was tested with multiple datasets to verify:

- Players are ranked correctly by win rate, ranked score, or average KDA.
- The system handles ties and large datasets efficiently.

5.2 Match Handling Validation

Match handling was tested with consecutive matches to ensure accurate updates to:

- Match statistics (e.g., `matchesPlayed`, `kills`, `assists`).
- Player rankings after each update.
- Special cases like zero kills, zero deaths, or invalid inputs.

5.3 Data Persistence Validation

The save and load functionalities were tested to verify that:

- Player data is written to and retrieved from a file.
- The system maintains ranking consistency after reloading data.

6 Code Execution Example

Below is an example of the system's functionality:

- **Initial Player List:**

```
ID   Name       Matches   Wins    WinRate   Kills   Deaths   Assists   RankedScore
1    Alice      10        7       70.00     50      10       15        1000
2    Bob        15        10      66.67     60      5        10        900
3    Charlie    12        8       66.67     40      12       20        800
4    Dave       8         4       50.00     20      8        12        700
```

- **After Adding a New Match:**

```
Player 1 Wins:
ID   Name       Matches   Wins    WinRate   Kills   Deaths   Assists   RankedScore
1    Alice      11        8       72.73     55      12       20        1050
2    Bob        16        10      62.50     65      8        15        880
3    Charlie    12        8       66.67     40      12       20        800
4    Dave       8         4       50.00     20      8        12        700
```

7 Conclusion

The Online Tournament Ranking System provides an efficient and accurate solution for managing player rankings in real-time. By leveraging optimized sorting algorithms and modular design, the system is highly extensible and reliable for tournament management.

7.1 Future Improvements

- Adding a graphical user interface to improve usability.
- Enhancing file handling to support multiple tournament sessions.
- Exploring advanced ranking algorithms to incorporate additional performance metrics.