Fail-Safe Cloud Tournament Engine with Error Detection and Error Recovery

UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

Kyle Chapman (20703236)
Supervisor: Dr. Cornelia P. Inggs
Cosupervisor: Mr. Andrew J. Collett

9 November 2020

Contents

Chapter 1

Introduction

The Fail-Safe Cloud Tournament Engine (FSCTE) is an extension of the Cloud Tournament Engine (CTE) developed by Reece Murray in 2018, which is an extension of the Tournament Engine (TE) built on the Ingenious Framework[1] (IF). The goal for the CTE is to allow people to play a turn-based game, provided it is supported in the IF, against each other in a match, where many matches make up a tournament. Multiple tournaments can be run to determine how good each player is. Tournaments can separated into two categories: private and public.

Private tournaments are tournaments that only the administrators can add players to, and they are mainly used to test some players against other players. Public tournaments are tournaments that any user can add their own players to and administrators can also add players to. These tournaments are used to put players against each other to rank each player based on how many wins they get against other players.

It was found that there were some major limitations to the CTE, such as minimal error reporting, lack of error recovery and an unmaintainable code base to name a few. As a result, there were situations that arose in which a player failure resulted in a failure of the entire tournament. This prompted the need for a version that was fail-safe and included error reporting and error recovery, as far as possible. The FSCTE aims to address these limitations and improve the overall system stability by being fail-safe, which means that if any error were to occur, the response to the error would cause as little harm to the overall system stability as possible.

[1]`https://bitbucket.org/skroon/ingenious-framework/src/master/`

1.1 Scope

The system used to manage the FSCTE is made up of multiple Docker[2] containers, which run each section of the FSCTE in their own separate environment. If some error occurs in a section of the system, it will not affect the other sections, since they are in separate environments.

The FSCTE uses the following three main sections that are run in their own Docker containers:

- **Web User Interface**: Allows a user to interact directly with the FSCTE.
- **Database**: Stores all the information relevant to the FSCTE.
- **Database watcher (Golang)**: Watches the database for any changes, such as tournaments being started, and sends messages to the front-end accordingly.
- **REST API**: Facilitates file upload to, and download from, the web user interface, as well as CAS authentication for logging in with Stellenbosch University credentials.

The layout of the entire FSCTE system is further discussed in chapter 3, *design and implementation*.

1.2 Document Outline

This report will discuss the overview of the system in chapter 2, the implementation and design of the system in chapter 3, then onto the testing in chapter 4 and discussion of the future of the project in chapter 5. Finally, the conclusion will be presented in chapter 6.

[2]`https://docs.docker.com/engine/docker-overview/`

Chapter 2

Overview

This chapter will give an overview of the requirements for the FSCTE system and how the requirements have been met.

The CTE used in 3^{rd} year at Stellenbosch University is designed to manage many Othello [1] tournaments, involving many players, concurrently. A user is able to view public tournaments, players, referees, schedulers, etc. and view relevant statistics for ongoing and completed tournaments. The admin, usually the lecturer, is able to add tournaments, schedulers and rankers and the students are only allowed to upload their players and add them to public tournaments.

2.1 Separate enviroments

Each section of the FSCTE needs to be in their own separate environment, so that if some error occurs, it impacts only that specific section and not the entire FSCTE. Docker was chosen for this purpose, since it is relatively easy to spin up a section in its own environment and redeploy that section if some error occurs.

The docker container is created by writing a `Dockerfile` that specifies which base environment you want to use (e.g. `ubuntu`, `node`, `neo4j` etc.) and the command you want to run in the container. The container exits when the command finishes, so in order to keep the container running forever, infinite loops are useful. This means that the command will only finish when the user forcefully stops the container.

Docker-compose[1] is used to spin up all of the docker containers used by the FSCTE, using their respective `Dockerfile`s, and allows for more control of how

[1]`https://docs.docker.com/compose/`

the container should be set up. Local directories can be mounted into the docker containers so that any files that are generated inside the container can be accessed outside of the container, on the host machine. This is useful for accessing the log files after a match has completed, so that it can be seen if the match was a success or not.

2.2 Web User Interface

The previous web user interface implemented in the CTE was not very user-friendly, which resulted in a lot of confusion for the students who used the CTE.

Chapter 3

Design and Implementation

A scheduler is used to schedule matches between two players in a round-robin fashion, where every player plays against every other player. After every player has played against every other player, the overall win-loss ratio can be calculated that can be used to distinguish between good, average and bad players. The win-loss ratio and the Elo rating [2] can be used in conjunction.

A ranker is used to distinguish between the good and bad players by comparing the Elo of the two players, where every player starts on the same amount of Elo. For every win that a player has, they gain a certain amount of Elo and for every loss that a player has, they lost a certain amount of Elo. The amount gained or lost decreases for every match that the player plays, meaning that eventually the Elo rating will be a true reflection of the skill level of the player.

A referee is used to monitor the moves that the two players make in a match and check if any player makes an invalid move or times out. One player sends a move to the referee, which then checks if the move is valid and only sends the move to the other player if the move made is valid. If the move made is invalid, or the player times out, the match will be forfeit and the last player to send a valid move becomes the winner.

Chapter 4

Testing

Chapter 5

Future Work

This chapter covers the future work that could be implemented in the Fail-Safe Cloud Tournament Engine (FSCTE) to extend the current functionality. The conclusion will be provided in section 6.

5.1 Kubernetes

Kubernetes[1] can be used to manage the docker containers, so that if a container crashes, another instance of that container can be redeployed without hesitation. Due to time constraints, it was not feasible to add Kubernetes to the current FSCTE.

5.2 Other Game Types

In the current FSCTE implementation, only the `Othello` game is supported. Supporting other game types will allow the FSCTE to be more abstract and versatile. Due to time constraints, it was not feasible to attempt to support other types of games.

[1] `https://kubernetes.io/`

Chapter 6

Conclusion

The main requirements of the FSCTE that needed to be satisfied were:

- have effective error handling
- have effective debugging facilities
- scale with tournaments and players
- have a logical, clean and maintainable code base
- have a more user-friendly web user interface

All of the above requirements were satisfied. The error handling requirement was satisfied by making sure that at any point where an error could occur, it was decided if the error was serious or not. If the error was serious, then an error message would be displayed and the component that caused the error would be stopped. If the error was not serious, then an error message would be displayed, but nothing would happen to the component that caused the error.

The effective debugging facilities requirement was satisfied by logging all information that happens inside each docker container, so that if some error occurs, the user can look in the docker logs to see at which stage the error occurred. This allows the user to know exactly what was going on before the error occurred so that the user can pinpoint the issue and fix the problem.

The tournaments and player scaling requirement was satisfied by writing the code base in such a way that it performs very well for any number of tournaments and players. Golang was mainly used to allow scaling to be possible through the use of concurrent threads running different tasks at the same time.

The maintainable code base requirement was satisfied by providing extensive documentation throughout the code base, so that anyone who reads the code base can understand what is going on without any issue.

The user-friendly web user interface requirement was satisfied by ensuring that the user can navigate to any page or get any information they want in a maximum of two mouse clicks. A lot of time was spent making sure that the web user interface looked clean and visually appealing, as well as easy to use.

Working on the FSCTE for the past year has been a great challenge, but also a great reward. Initially, it was tough to understand what was going on in the CTE because of the lack of documentation and confusing way that some sections were implemented. After understanding the previous work, the toughest challenge was getting the players to play a match and communicate their results to the web user interface. However, it was a fun and informative experience.

Bibliography

[1] Masters Traditional Games. *Rules and Instructions for Reversi and Othello*. 2019. URL: `https://www.mastersofgames.com/rules/reversi-othello-rules.htm`.

[2] Adam Newell. *What is Elo? An explanation of competitive gaming's hidden rating system*. Jan. 2018. URL: `https://dotesports.com/general/news/elo-ratings-explained-20565`.