RockPaperScissors Concurrency Report

Danish Hussain - P14181072

April 2016

Supervisor 1: David Smallwood

Supervisor 2: Eseosa Oshodin

Contents

1 Introduction **3**

2 Choice Of Language **4**

3 Design And Implementation **5**

4 Experience Gained **7**

1 Introduction

This report is written in relation to the rock paper and scissors game I have developed as part of my assignment. The report will highlight my choice of language, as well as covering various aspects of the code for the design and implementation whilst providing justification for my chosen approaches and methods. To conclude the report, I will attempt to explain the experiences I have gained.

2 Choice Of Language

The language I chose to use for this assignment was Java, there are various reasons that influenced my decision, the main one being familiarity. Being someone who does not possess the strongest programming skills it was always going to be the factors of difficulty and understanding that would affect my decision and the only two languages I am familiar with is C and Java. Another reason as to why I chose Java was because I knew it would be easier to conduct research on concurrency and there will be numerous guidelines and tutorials which would aid me with my understanding. In terms of the language itself then it has various advantages over C such as the built-in libraries and the whole concept of object-orientated, the fact that Java is a language that was developed to be easy to learn made it my ideal choice. Another reason as to why I chose Java over other languages was because it is multi-threaded, the assignment involves working with threads and Java has multithreading integrated into it. Overall I knew the assignment would be difficult so it was best to stick with a language I was familiar with, attempting to do the assignment in a different language would have cost me a lot of time and possibly an incomplete solution.

3 Design And Implementation

One of the most difficult aspect of the programming was the referee class hands down, due to the fact that the referee plays a huge factor in the program it was vital to get the implementation spot on. For the choosing of two players I chose to use a LinkedBlockingQueue, from which I called the take method for each player to remove their messages from the queue and compare them. The main reasons for choosing this queue was that it is thread-safe and also it is optionally bounded which meant that I did not have to specify a fixed size, so it could grow as required. Another reason for the LinkedBlockingQueue was that generally it has higher throughput for it, however a disadvantage that I chose to ignore was that it could be less predictable.

The referee determines the outcome of each game based upon the method specified in my result class, in which I have set an integer value to each variable e.g. (win = 2, lose = 1 and draw = 0), which I have then also defined in the referee class so when the method is called the values of each class are compared and if they are equal then the results are passed respectively to each player. I chose to implement the results in this way because I thought it was the easiest way for me to determine the winner, and it would also be a simple concept for anyone assessing or working with my code to understand and alter.

When the player threads run method is called, a new message object is created which holds a player, unique id, shape and a link to an MVar which the referee can insert the return communication into. For the NTURNS, each player selects a random shape for their message which is done by calling a set method from the shape class and then setting the value of a random shape. Once the shape is selected the player thread calls the addToQueue method specified in the referee class which allows it to add the message to the queue which is maintained by the referee. After the message is added the player thread sleeps for the specified time and waits for a response according to the method stated in the MVar. After the player receives their response, it calls an inner class method called incrementResult () which compares the result received and then updates the internal state of each thread respectively.

The referee thread closes down gracefully when all of the player threads have stopped and the main program gives it the signal (i.e. call the close method), this part for me was probably the easiest to implement and I did not face any substantial difficulties with it, I expected it to be a lot harder and on par with the other difficulties of this assignment. In terms of implementation I used a volatile Boolean for both player and referee classes which while is true the run method keeps running, and when it finishes the inner class close method is called to terminate the thread and then prints out a message, i.e the player thread number and their game outcomes and the referee thread to say the thread has stopped.

In terms of design decisions, I decided that I wanted to attempt a simple ASCII display which I have never tried before. Although I feel like I could have possibly created something more creative given more time, I can proudly say that I am quite satisfied with the display I have created and believe the assessors will appreciate the effort as it is something different from the given example code. The implementation of the code was quite simple as it is just a combination of print messages, however getting the message to print perfectly on the console was something that was quite challenging.

4 Experience Gained

When we were first given the assignment I was unsure I would be able to produce the work I have, after numerous hours and dedication I can finally say that I did it and I am glad that we were given such a challenging assignment. I have definitely gained a lot of experience from undertaking this assignment, for example I have a greater understanding of how to use concurrent threads in Java and I understand most of the concepts whereas when I started I hardly knew anything. One specific example I can highlight would be the implementation of the blocking queue, firstly setting it up and then secondly allowing the player to add from his class which was a concept that took me a long time to figure out.

The aspect I am happiest with is the overall results, just to know that I have the correct threads printing and stopping at the right time with their correct internal updated states, the relief I got from finally fixing it after so much effort is unexplainable. The most difficult part in the assignment was getting the correct thread numbers to print their respected scores, which took me days to figure out but thankfully I got there in the end. The display took me a very long time to do in attempts to get the maximum marks for the assignment, hopefully I have done enough and my results are displayed clearly.

If I was given a similar problem again I believe I would be able to tackle the issue with more confidence and I believe in my ability to develop a solution a lot quicker than this one. I would approach the problem firstly by writing down what each thread needs to do and then deciding the relevant methods that will need to be implemented. I can now easily understand concepts such as thread-safe queues, MVar, producer-consumer relationships, so I think it would be quite interesting to have another shot at producing something similar. If I had more time on this assignment, then I feel like I would make a greater effort on the display to show full creativity and flair for the solution I have produced.