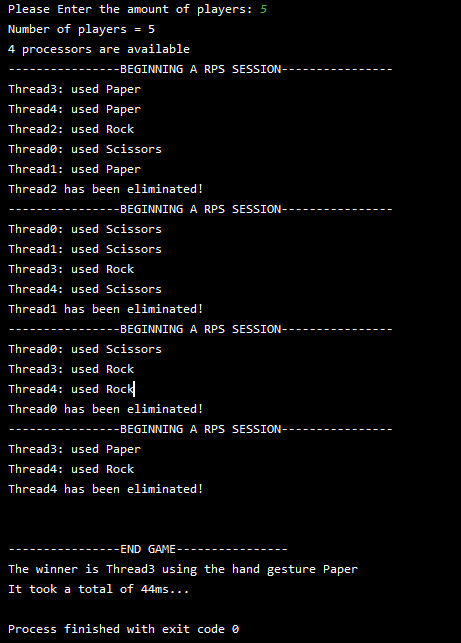
Bryce Callender

<https://github.com/BryceCallender/CS-3700/tree/master/RockPaperScissorsThreaded>

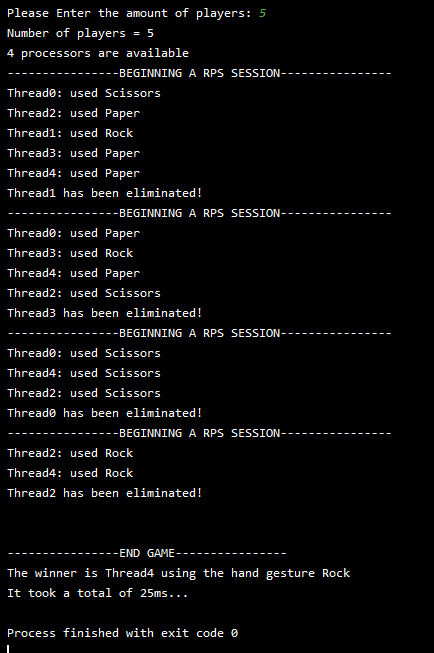
GAME 1 IMPLEMENTATION 1 OUTPUT

Explanation: This code uses a threadpool and then executes the threads on a threadpool service. The calculation for the winner will not happen till all threads increment the atomic integer then the code can precede.



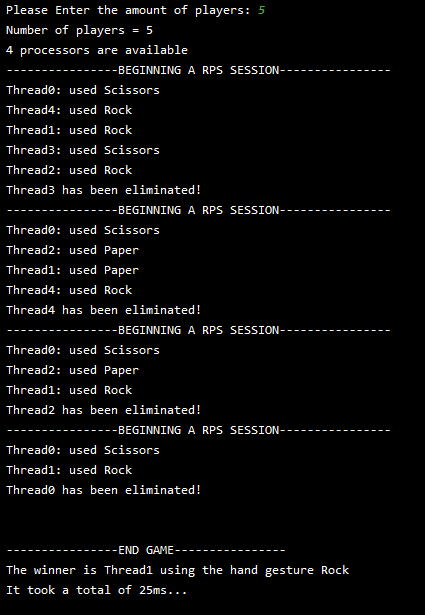
GAME 1 IMPLEMENTATION 2 OUTPUT

Explanation: This uses the threadpool and instead of executing commands I submit them waiting for the futures to be done. Once futures are done the results are calculated.



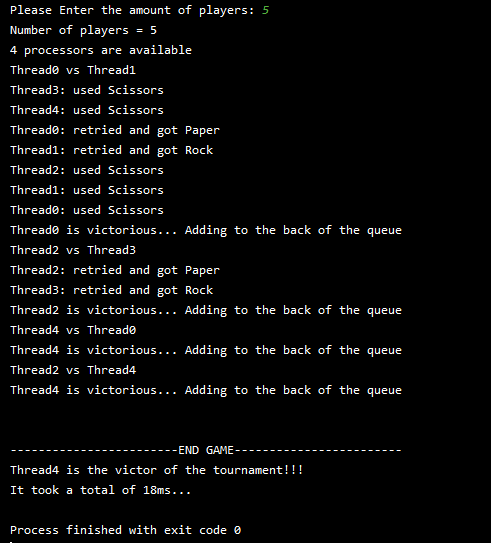
GAME 1 IMPLEMENTATION 3 OUTPUT MEMOIZED

Explanation: Has the same time as above as making a map and adding to it with only 5 people is just a bit less efficient so it comes out to the same time, but for larger output it is better. Uses a threadpool with futures and a memorized table for quick access.



GAME 2 IMPLEMENTATION 1 OUTPUT

Explanation: Uses a threadpool with executing the runnables. The threads are responsible for adding themselves to a blockingqueue. The winner thread can then take 2 players out and battle them. If the queue is empty the thread will block.



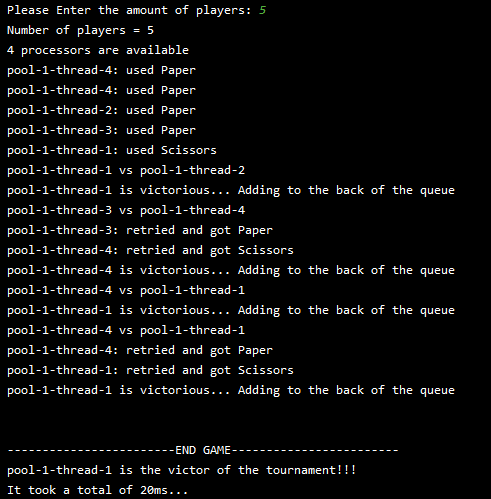
GAME 2 IMPLEMENTATION 2 OUTPUT

Explanation: This one is similar in structure to the above except it was an experiment with countdownlatch instead of using an atomic integer. Main thread will wait till that countdownlatch hits 0!



GAME 2 IMPLMENTATION 3 OUTPUT

Explanation: This uses a threadpool with future submissions. Once the futures are done then they can take stuff out of the arraylist and then read who won.



DECLARATION OF INDEPENDENCE

Single Thread



Multi Core output

