CS 1632 – DELIVERABLE 5: PERFORMANCE TESTING CONWAY’S GAME OF LIFE

PROJECT UNDER TEST: JAVALIFE

ALEC FOX

TUTH 4PM

Code found at: https://github.com/adf37/CS1632\_Projects/

jUnit test file: TestProfiler.java under SlowLifeGUI\_modified

SUMMARY:

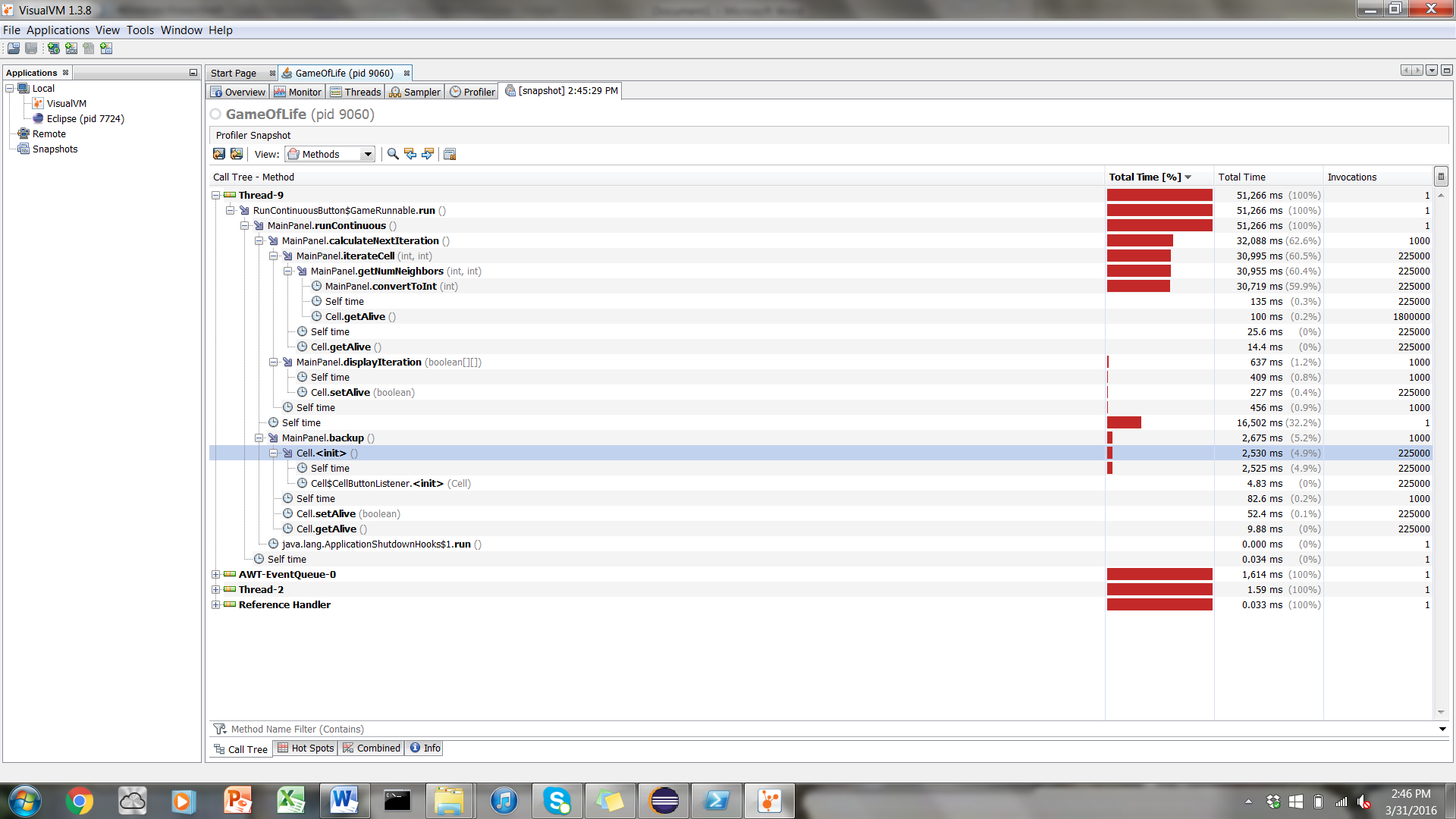
To profile the program “Conway’s Game of Life” I used the virtualVM profiler that I downloaded online. This tool allows you to run an instance of the game and profile the CPU usage of the various methods that are called during its execution. To properly profile the program and to see how my modifications and refactoring of the code impacted overall performance I set an iteration maximum of one thousand and used the run continuous button to execute these iterations. On the first run of the game with the original code the program took about fifty-one seconds to run one thousand iterations. From this point I was able to examine how CPU intensive each method was to start my refactoring process.

I could see from the first snapshot that the calculateNextIteration method was the most CPU intensive by far accounting for a little over 60% of the CPU time. This is when I made the first modification in the getNumNeighbors method to get rid of the call to convertToInt which simply returned, after a lengthy delay the original value that was being passed in. By skipping this method and simply returning the number of neighbors instead CPU time was cut down to about twenty one seconds for one thousand iterations of the game.

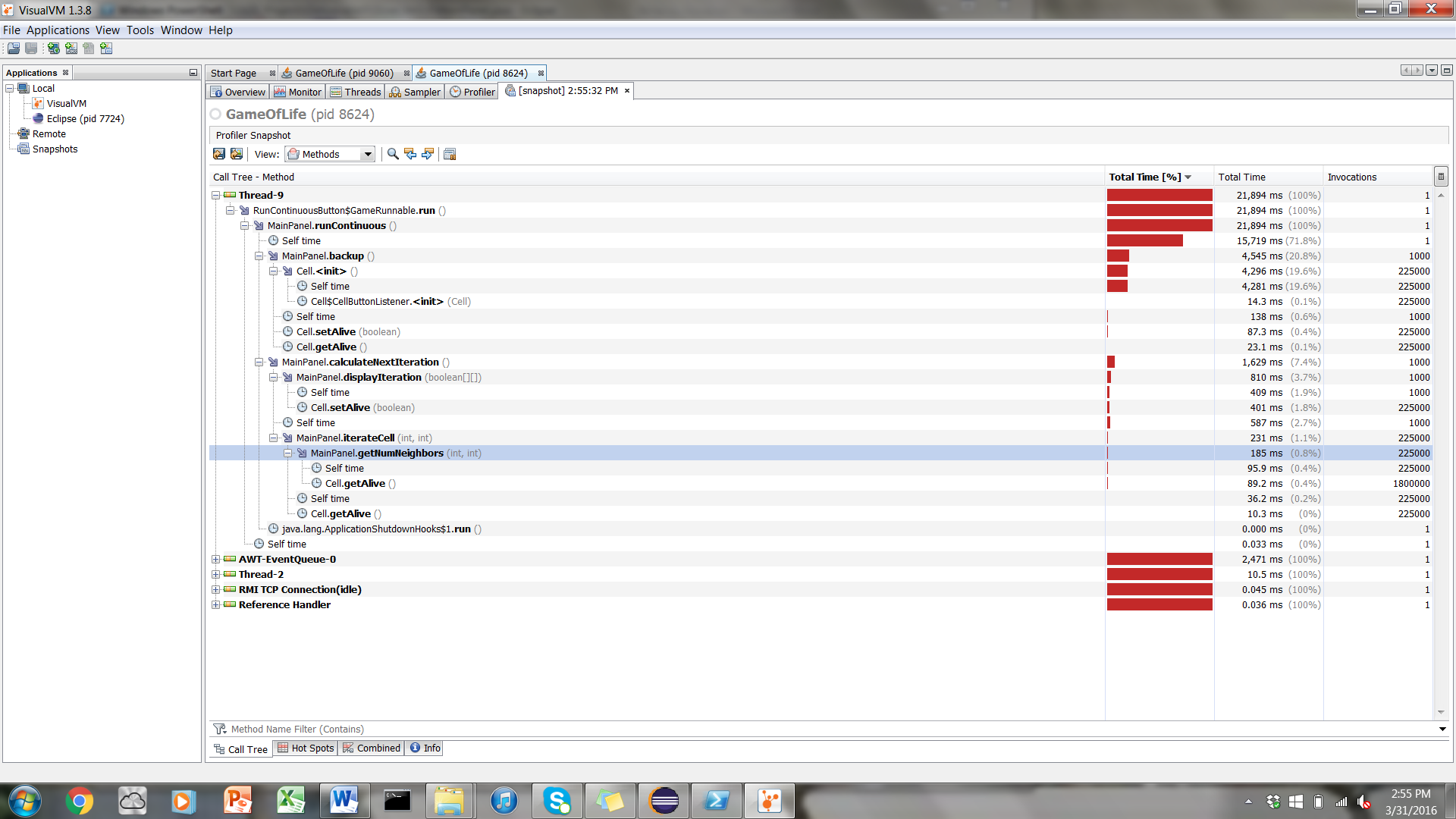
Secondly, I could see that the run continuous method had an unnecessary delay within its own method call. The code section was doing a number of for loop iterations to change a variable, \_r, that was reset after the completion of this section. By commenting out this section the program was able to run even faster at about eight seconds for one thousand iterations.

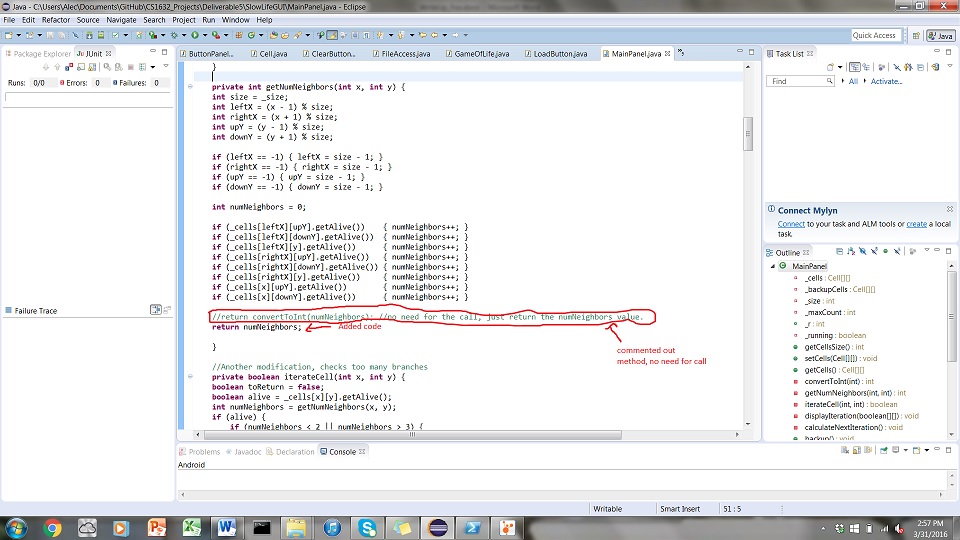
Finally, I could see from the profiler that the backup method was taking up about half of the remaining CPU usage for each run of my tests. In the backup method I noticed that a nested for loop was being used to copy each individual cell of the current cell array to the backup cell array. This process could be sped up by eliminating this nested for loop and using the java util’s built in Arrays.copyOf method to copy the contents of the current cell array to that of the backup array much quicker. With this refractor I was able to cut down total CPU time to about five seconds for one thousand iterations of Conway’s Game of Life.

SCREENSHOTS:

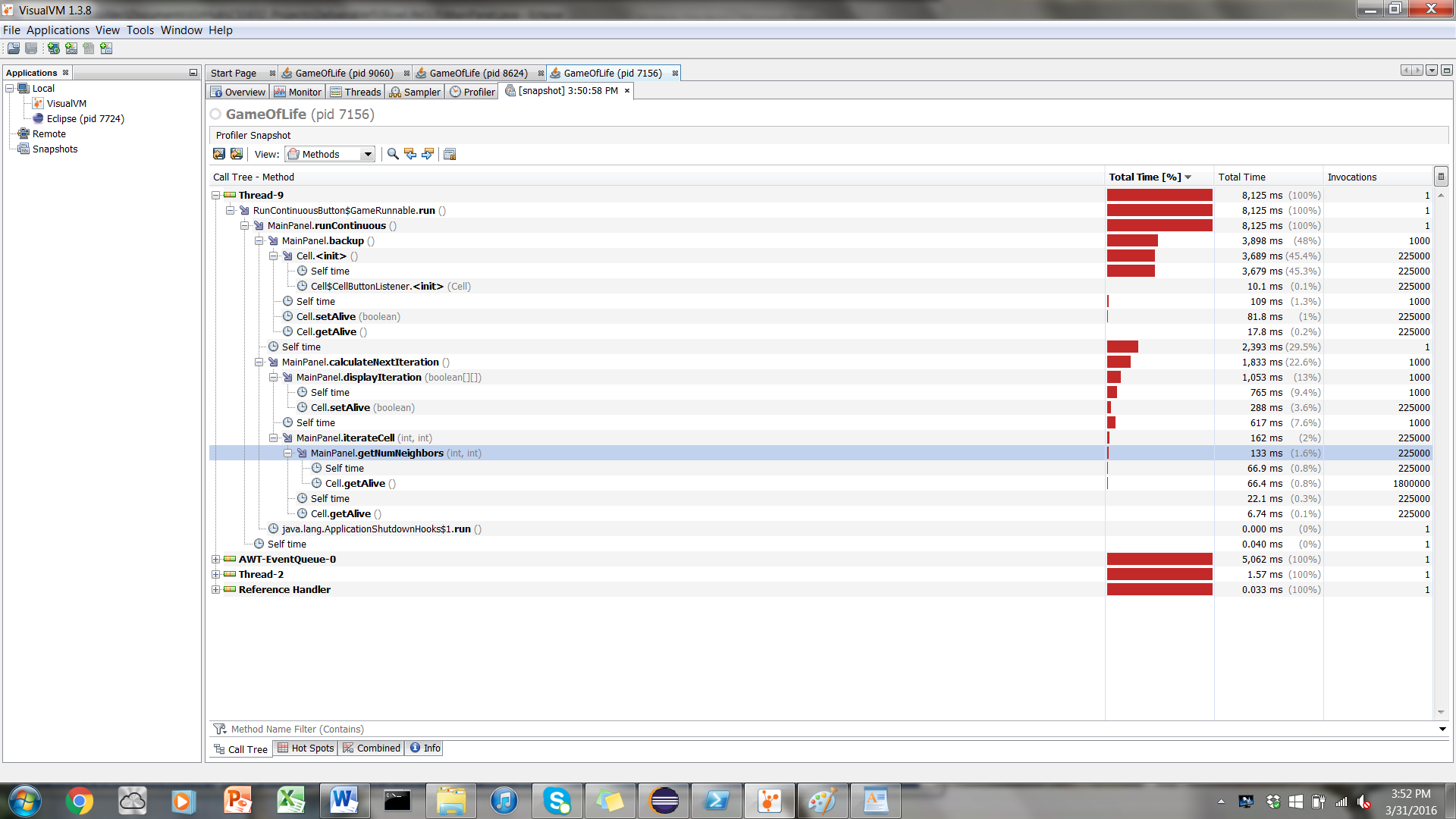


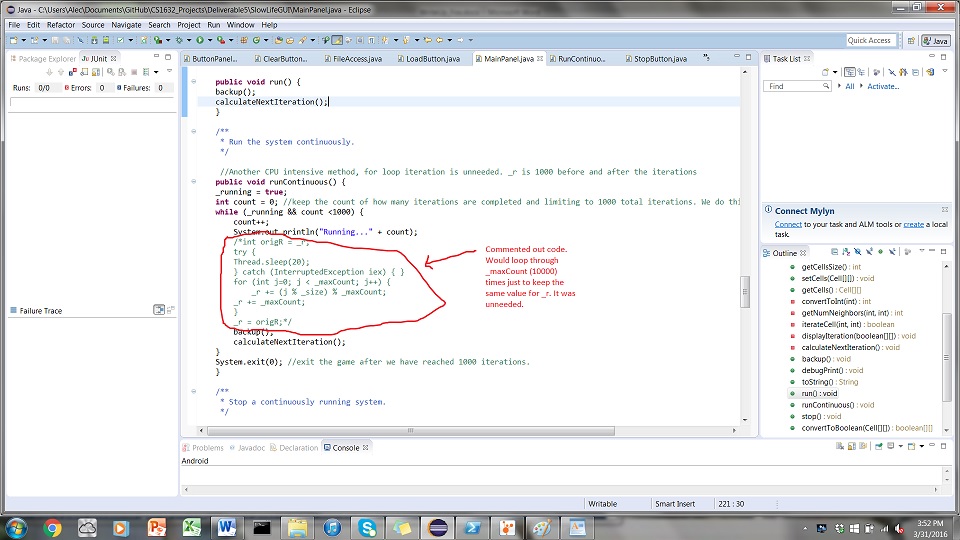
The above is the snapshot of 1000 iterations of the Game of Life using the run continuous button and zero code modifications.



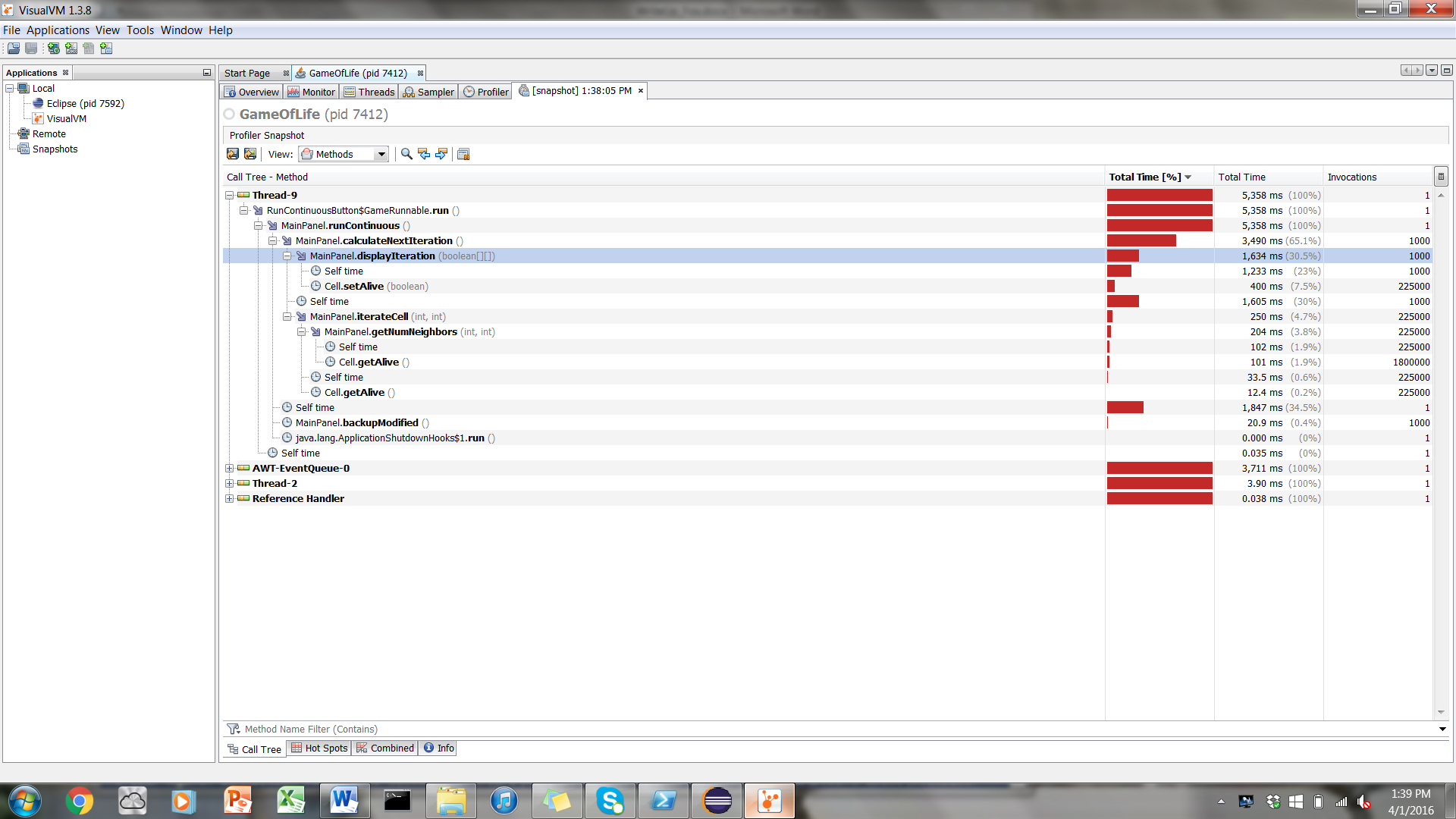
The above snapshot is the result of the following change: 

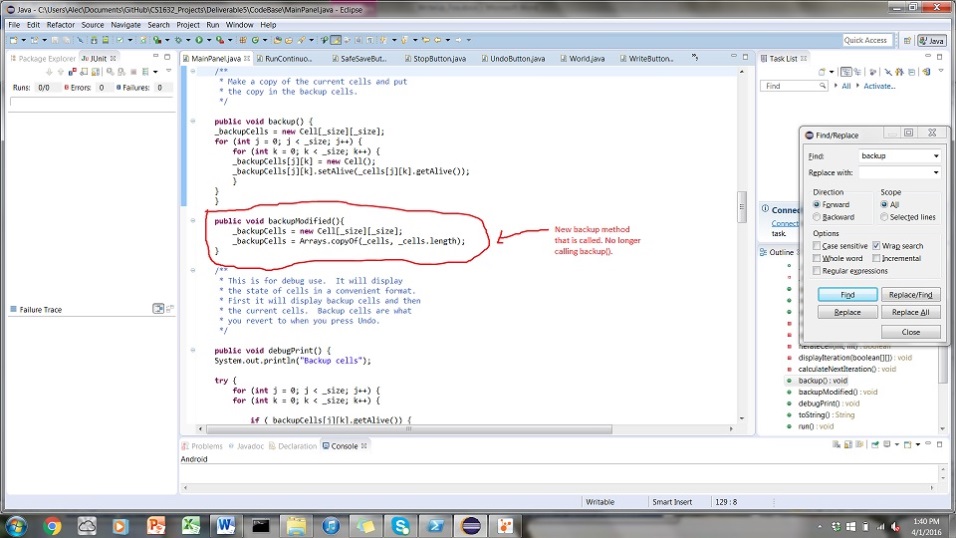
Change number 2:



With the following change:

Change 3:





Junit Test Execution:

