**Matthew Damiani**

**CS 1632-DELIVERABLE 5: Performance Testing Conway’s Game of Life**

Performance Testing Summary

For my initial profiling of the application, I used the VisualVM tool to monitor CPU usage of methods within the application. More specifically, I was looking for methods that were consuming a lot of the CPU’s time. To do this, I open up the VisualVM tool and went to the “Sampler” tab of the “GameOfLife” application within Eclipse. From there, I selected the CPU option for the sample so that I could view CPU usage. I proceeded to press the “Run Continuous” button and let it run for 30 seconds before pressing “Stop”. I then pressed on the rectangles on each of the 4 corners of the grid, followed by the pressing of the ”Write” button a couple of times and then the “Clear” and “Load” buttons. I did all of this to try to have a majority of the methods withing the application run so I can see which are using the most CPU. After all of that, I looked at VisualVM to see the CPU usages of the individual methods. The top three CPU consuming methods were convertToInt(), toString(), and runContinuous, so I decided to analyze the actual code of those three methods to see what caused them to take up so much time. I would then refactor them to improve the performace of the application.

Within the convertToInt() method, I noticed it just took in an int and essentially returned that same int value except in between in did a lot of work only to come back to the same int value. Since this method simply returned the same int that was passed in, I got rid of everything in the method and wrote a single line of code which was to return the int that was passed in. This would eliminate all of the time wasted on manipulating the int, only to arrive at the initial value passed in.

Within the toString() method, I noticed that there was a for loop that would continuously append the same string . This was not necessary since you only needed one of the getText() string, you did not need to continuously keep appending it to itself. Therefore, I got rid of the for loop that continuously appended the same string to itself and that fixed the problem where toString() was taking up too much CPU time.

Within the continuousRun() method, I noticed there was a block of code that manipulated the \_r value through various loops and calculations but it never used that modified \_r value, it just set \_r to the original value before all of the work was done. This was a block of code that ultimately did not change the value of \_r and therefore was not needed since it just reset \_r to the original value after all of th work was done. As a result, I removed this chunk of code which was predominantly a for loop, and the large CPU usage issue for continuousRun() was resolved.

Additionally, some of my pinning tests were in the form of manual tests instead of Junit tests. As required by the assignment description, I have included those manual test cases below, in the format that we learnd from deliverable 1.

My code can be found at https://github.com/mdamiani610/SlowLifeGUI

Manual Test Cases

Identifier: testRunContinuous1()

Test Case: The game will run iterations without error when the “Run Continuous” button is pressed, and will continue to do so until the “Stop” button is pressed

Pre-condition: User has Execute GameOfLife.java and waited until the main page of the GUI appears

Execution Steps:

1. Press the “Run Continuous” button
2. Wait 30 seconds
3. Press the “Stop” button
4. Observe the terminal output and look for any errors that might have occurred during program execution.

Post-condition: The user should not see any error messages in the terminal output from the continuous running of GameOfLife for 30 seconds.

Identifier: testRunContinuous2()

Test Case: The game will run iterations without error when the “Run Continuous” button is pressed, and will continue to do so until the “Stop” button is pressed. This will be after manually running 3 iterations of the game by pressing the “Run” button 3 times.

Pre-condition: User has Execute GameOfLife.java and waited until the main page of the GUI appears

Execution Steps:

1. Execute GameOfLife.java and wait until the main page of the GUI appears
2. Press the “Run” button
3. Press the “Run” button
4. Press the “Run” button
5. Press the “Run Continuous” button
6. Wait 30 seconds
7. Press the “Stop” button
8. Check the terminal output to ensure that no errors occurred during the continuous run
9. There should be no error messages in the terminal output

Post-condition: The user should not see any error messages in the terminal output from the continuous running of GameOfLife for 30 seconds after three iterations of the game were run manually.

Identifier: testRunContinuous3()

Test Case: The game will run iterations without error when the “Run Continuous” button is pressed, and will continue to do so until the “Stop” button is pressed. This will be after pressing the “Write” and “Load” buttons to save and restore the state of the system.

Pre-condition: User has Execute GameOfLife.java and waited until the main page of the GUI appears

Execution Steps:

1. Execute GameOfLife.java and wait until the main page of the GUI appears
2. Press the “Write” button
3. Press the “Load” button
4. Press the “Run Continuous” button
5. Wait 30 seconds
6. Press the “Stop” button
7. Check the terminal output to ensure that no errors occurred during the continuous run
8. There should be no error messages in the terminal output

Post-condition: The user should not see any error messages in the terminal output from the continuous running of GameOfLife for 30 seconds after the state of the system was written and loaded

Identifier: testToString3()

Test Case: The game will successfully write the state of the system to a backup file when the “Write” button is pressed.

Pre-condition: User has Execute GameOfLife.java and waited until the main page of the GUI appears

Execution Steps:

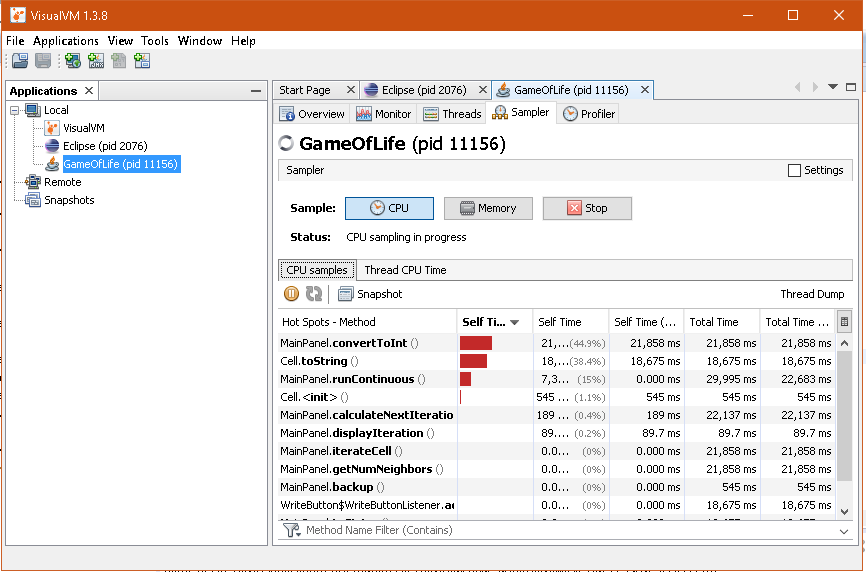
1. Click on the top left rectangle in the grid and observe it turn into a red rectangle with an “X” inside
2. Click on the top right rectangle in the grid and observe it turn into a red rectangle with an “X” inside
3. Click on the bottom left rectangle in the grid and observe it turn into a red rectangle with an “X” inside
4. Click on the bottom right rectangle in the grid and observe it turn into a red rectangle with an “X” inside
5. Press the “Write” button
6. Press the “Clear” button
7. Press the “Load” button
8. Observe the grid and note which rectangles are red with an “X” inside of them

Post-condition: The user should see a red rectangle with an “X” inside for the rectangles in the grid at the top left, top right, bottom left, and bottom right.

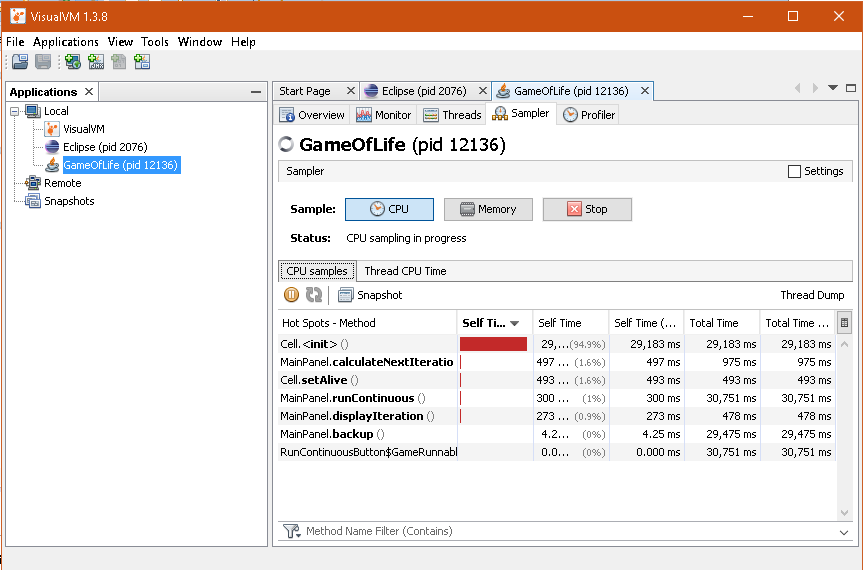
VisualVM Screenshots

(Running continuously for 30 seconds and then writing 3 times)

Before Refactoring:



After Refactoring:



Screenshot of executed Junit tests

