

CS 1632 - DELIVERABLE 5

Members: Christian Boni

Project: Performance Testing Conway's Game of Life

## Summary

The purpose of this deliverable is to performance test Conway's Game of Life. The source code for Conway's Game of Life was given to me (by my professor Bill Laboon) with several deliberate performance issues. First and foremost, I did some research to gain a better understand on the behavior of Conway's Game of Life and the theories behind it. Secondly, I read the documentation accompanied by the source code to learn how the program operates and interacts with the user. After doing all of this I could then proceed with some exploratory testing on the application. When I did not notice any major performance issues during the exploratory testing, I downloaded and installed the VisualVM profiler tool for some additional help. VisualVM is essentially a tool that allows you view detailed information about any Java Application while it is actively running on the Java Virtual Machine (JVM). Now with VisualVM running I was able to individually monitor each method's interaction with the Central Processing Unit (CPU). As I was profiling Conway's Game of Life on VisualVM I cycled through all of the different buttons and functionality on the actual application. When I finished, I quickly discovered 3 methods that seemed to be using a majority of the CPU's processing time. These methods included: `Cell.toString()`, `MainPanel.runContinuous()`, and `MainPanel.convertToInt()`.

The next step in the deliverable was to attempt to locate the code in each of these methods which was causing the performance problems. When going over the code for the `Cell.toString()` method I realized that there was a rather large for-loop which had no use at all. The purpose of the `toString()` method was to return either an "X" indicating a living cell or a "." indicating a dead cell. The issue in this method was the for-loop which iterated 10,000 times appending the text of the current cell to a string during each iteration. However, when the for-loop finished only the first letter of the string was used for the comparison anyway. Thus, the for-loop was completely irrelevant. An almost identical problem occurred in the `MainPanel.runContinuous()` method. Another for-loop which iterated 10,000 times performed a calculation on an unknown public global variable at each iteration. It turned out to be the case that immediately after the for-loop this calculation was thrown out and replaced with the value immediately before the for-loop. Again making this for-loop useless. The `runContinuous()` method is a void method which runs continuously until the user clicks the "Stop" button on the application. For that reason, it was rather hard to create automated pinning tests for it. As a result, I chose to create several manual pinning tests for it which immensely simplified the testing process whenever I finished refactoring. The last and final refactored method, `MainPanel.convertToInt()` also contained an unnecessary for-loop. This for-loop again iterated 10,000 times and during each iteration appended a "0" to a string known as the padding. This padding string was then concatenated with a string of the integer that was passed in as a parameter. After analyzing the `convertToInt()` method and removing the unneeded loop I found out that this method had absolutely no purpose. This method simply returns the same exact integer that was passed in as a parameter. However, to be sure of this I did infact keep the method as a basis to use for several of the automated pinning tests which tested its functionality.

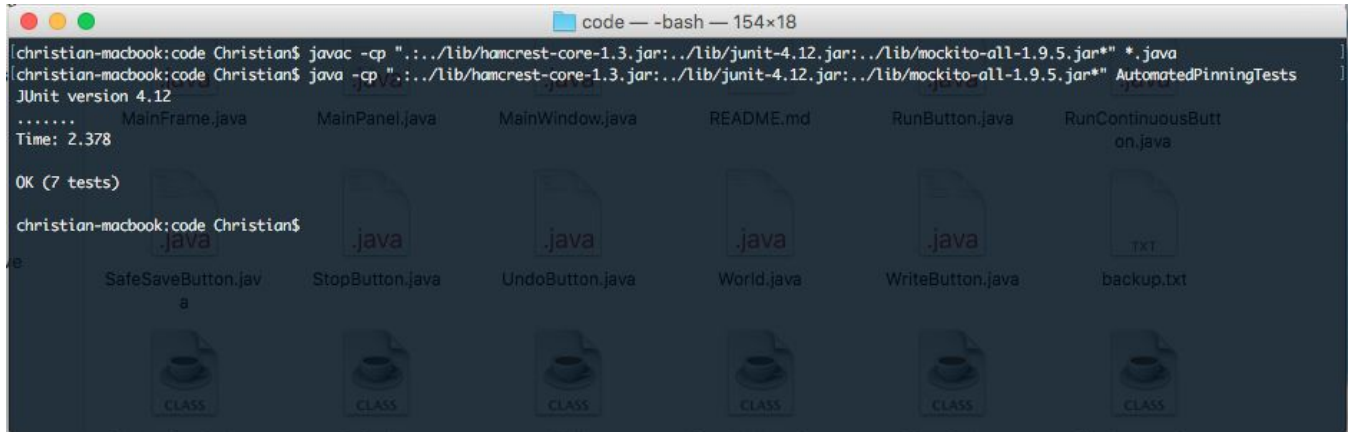
## Source Code

<https://github.com/thisbechristian/quality-assurance/tree/master/deliverables/d5>

## Unit Tests

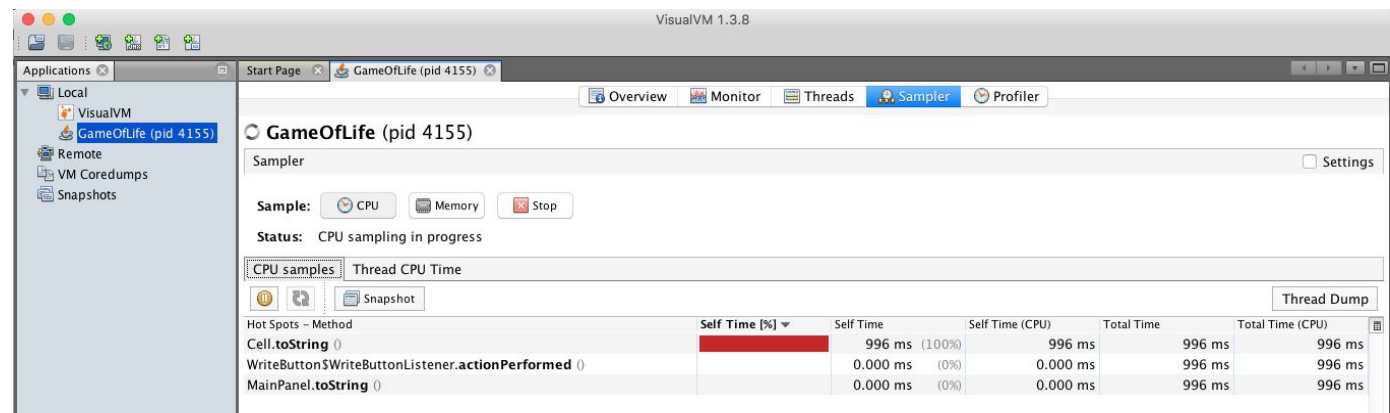
```
code — bash — 154x18
christian-macbook:code Christian$ javac -cp ".../lib/hamcrest-core-1.3.jar:.../lib/junit-4.12.jar:.../lib/mockito-all-1.9.5.jar*" *.java
christian-macbook:code Christian$ java -cp ".../lib/hamcrest-core-1.3.jar:.../lib/junit-4.12.jar:.../lib/mockito-all-1.9.5.jar*" AutomatedPinningTests
JUnit version 4.12
..... MainFrame.java MainPanel.java MainWindow.java README.md RunButton.java RunContinuousButton.java
Time: 2.378

OK (7 tests)
christian-macbook:code Christian$
```

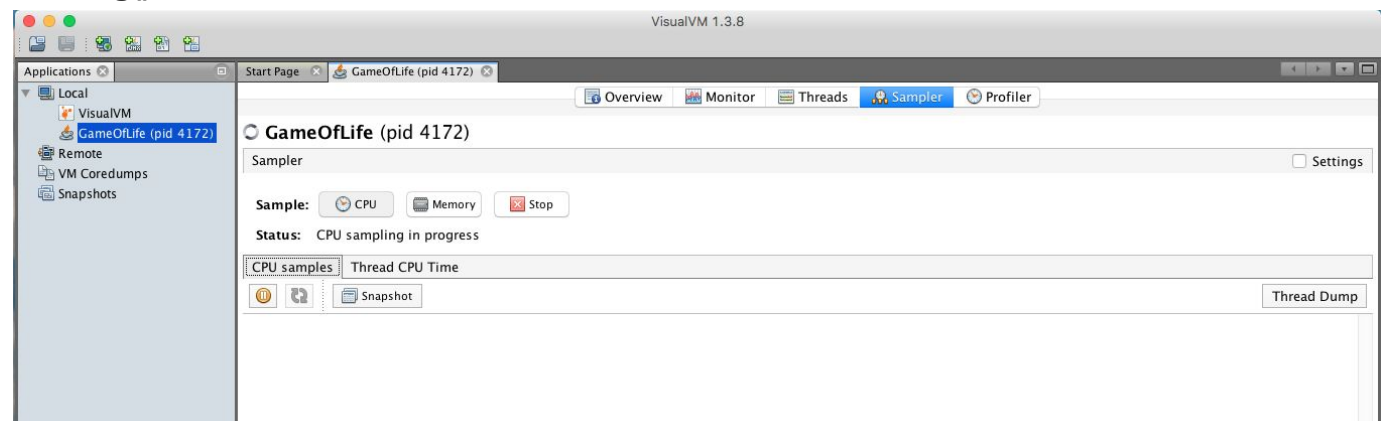


# Profiler Results

## toString() Before Refactor:

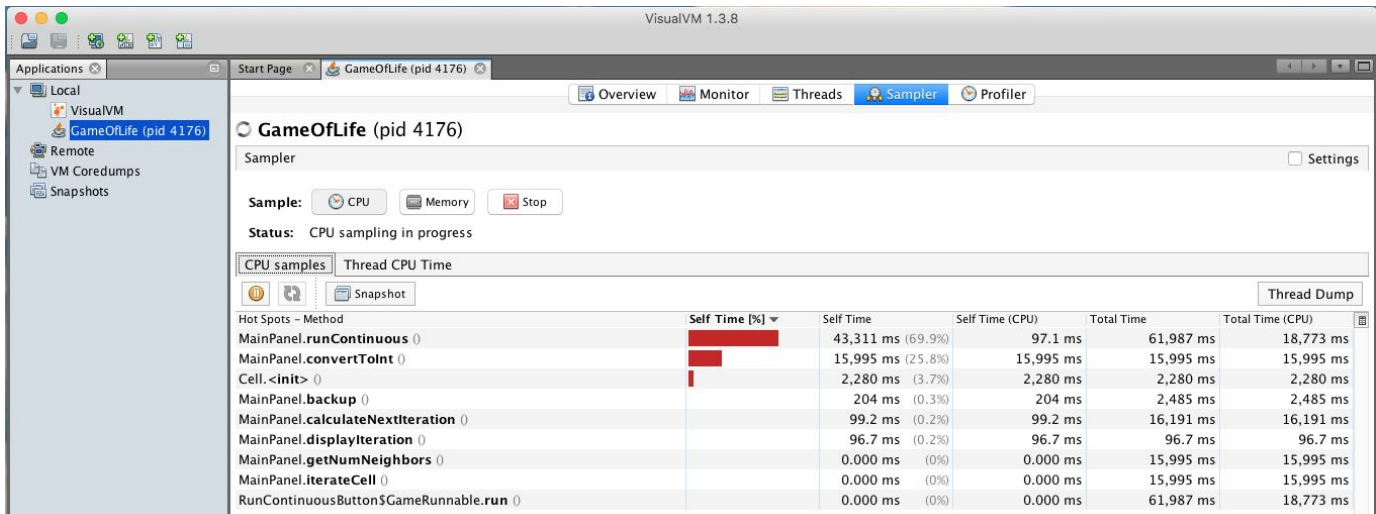


## toString() After Refactor:

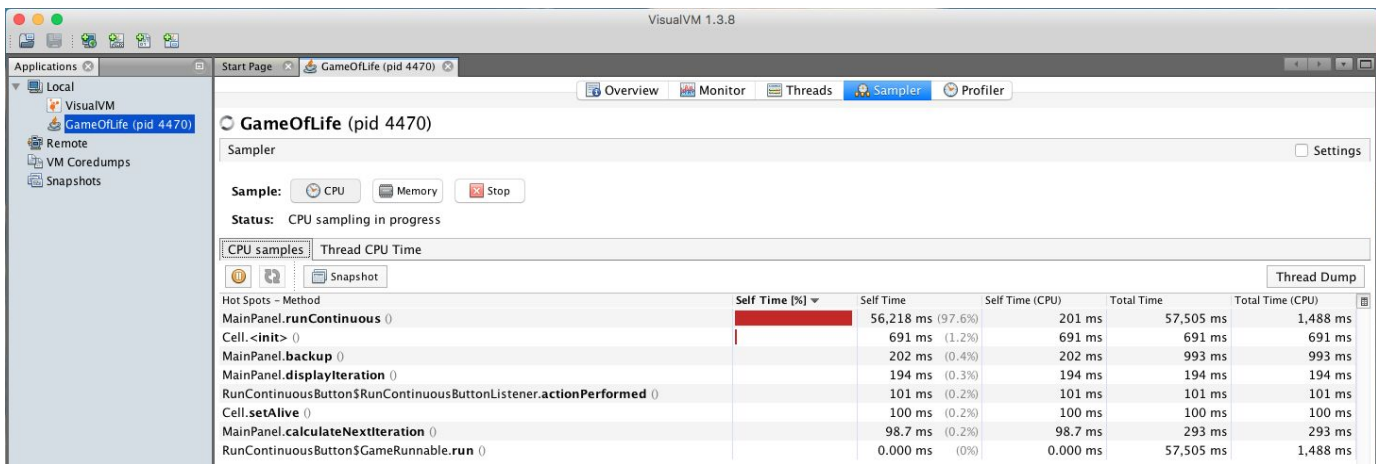


**\*While pressing the “Load” and “Write” buttons in the “Game of Life” window while VisualVM was sampling the application I could not get it to pick up any method hot spots. This shows me that the refactoring on the toString() method did indeed work.**

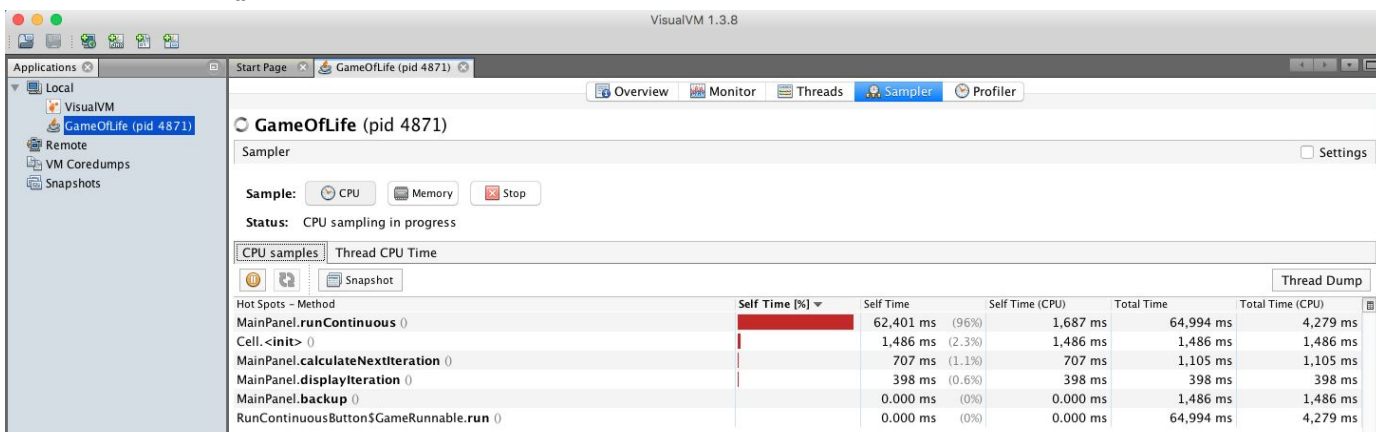
## convertToInt() & runContinuous() Before Refactor:



## convertToInt() After Refactor:



## runContinuous() After Refactor:



\*Not much of a difference after the refactor for the runContinuous() method. It contains thread.sleep(20) call which causes the thread to sleep for 20 milliseconds. This is done to slow down the graphical interface so the user can observe the changes in the Game of Life.