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

David Neiman

Project Under Test: JavaLife

https://github.com/dsn9/SlowLifeGUI

**Summary**

I began my work on this assignment by conducting exploratory testing, running through many different game setups. After that, I used VisualVM to profile the application.

The first thing I noticed was that the method convertInt() in MainPanel took up 38.9% of the time. This made sense when I looked at the code and noticed that there was a loop that ran 1000 times during each call of the method. These loops appended a "0" character to a string each time, and then added the value passed into the method into the first place. After parsing this string to an integer, the method returned the same value that was passed into the method. After taking out the unnecessary loops and string operations, the convertInt() method performed very quickly.

I then looked at the method running for the most time, runContinuous(), but this was a little more involved since it made calls to several other methods in the code, and used other classes like Cell. I noticed in the Cell class that the toString() method involved loops, appending characters onto the string. However, since only the first character of the string was needed for this method, I was able to remove the loop and refactor this into a simple method that checked the first character and then returned a value. I also removed an unnecessary loop that was directly in the runContinuous() method.

I wrote several pinning tests to ensure that the refactored methods had the same observed behavior as the original methods. I wrote unit tests for the convertInt() and toString() methods. I set the text for the getText() method used in the toString() method. For both of these tests, I also made sure to test that they would fail given certain inputs; I tested that the original and refactored convertInt() methods both threw a NumberFormatException when given a negative integer input, and that the original and refactored toString() methods both threw a StringIndexOutOfBoundsException when given an empty string input. The runContinuous() method was much harder to test, since it was a void method that also involved other methods, so I wrote manual tests to ensure that the game still operated according the rules of Conway's Game of Life.

After refactoring these methods and profiling the application, it was clear that the application performed differently, not spending any significant time on methods such as convertInt(). My unit tests all passed, indicating that these refactored methods had the same observed behavior as the original methods, and the manual tests show that the application runs continuously in the same manner as it did prior to refactoring.

**Manual Tests (for running continuously)**

**Requirements:**

LIVE-CELL-SAME: Any live cell with two or three neighbors will remain a live cell.

LIVE-CELL-DEATH: Any live cell with fewer than two neighbors or more than three neighbors will become a dead cell.

REVIVE-DEAD-CELL: Any dead cell with exactly three live neighbors will become a live cell.

**Test Cases:**

IDENTIFIER: (1) LIVE-CELL-SAME-1   
TEST CASE: Run continuously on empty grid.  
PRECONDITIONS: Fresh start of program and 4 entered as an argument when running the GameOfLife program.  
INPUT VALUES: 4  
EXECUTION STEPS: Press the 'Run Continuous' button on the GUI.   
OUTPUT VALUES: N/A  
POSTCONDITIONS: The cells all remain empty and grey.

IDENTIFIER: (2) LIVE-CELL-SAME-2  
TEST CASE: Run continuously on grid with live cells that only have two or three neighbors.  
PRECONDITIONS: Fresh start of program and 4 entered as an argument when running the GameOfLife program.  
INPUT VALUES: 4  
EXECUTION STEPS: Click on the first and fourth cells of the first and second rows once each. Click on Press the 'Run Continuous' button on the GUI.   
OUTPUT VALUES: N/A  
POSTCONDITIONS: The cells all remain red and alive.

IDENTIFIER: (3) LIVE-CELL-DEATH-1  
TEST CASE: Run continuously on full grid with every cell alive and having more than three live neighbors.   
PRECONDITIONS: Fresh start of program and 4 entered as an argument when running the GameOfLife program.  
INPUT VALUES: 4  
EXECUTION STEPS: Click on every cell once. Press the 'Run Continuous' button on the GUI.   
OUTPUT VALUES: N/A  
POSTCONDITIONS: The cells all die and become green.

IDENTIFIER: (4) LIVE-CELL-DEATH-2  
TEST CASE: Run continuously on grid with alive cells that only have fewer than 2 live neighbors.   
PRECONDITIONS: Fresh start of program and 5 entered as an argument when running the GameOfLife program.  
INPUT VALUES: 5  
EXECUTION STEPS: Click the first cell in the first row, the middle cell in the third row, and the last cell in the fifth row once. Press the 'Run Continuous' button on the GUI.   
OUTPUT VALUES: N/A  
POSTCONDITIONS: The cells all die and become green.

IDENTIFIER: (5) REVIVE-DEAD-CELL-1  
TEST CASE: Run continuously on partially full grid, with many cells having three alive neighbors.   
PRECONDITIONS: Fresh start of program and 3 entered as an argument when running the GameOfLife program.  
INPUT VALUES: 3  
EXECUTION STEPS: Click on every cell in the first row once. Press the 'Run Continuous' button on the GUI.   
OUTPUT VALUES: N/A  
POSTCONDITIONS: All cells outside of the first row first become alive and red and then all cells in the grid die and become green.

IDENTIFIER: (6) REVIVE-DEAD-CELL-2  
TEST CASE: Run continuously on grid with no dead cells having three alive neighbors.   
PRECONDITIONS: Fresh start of program and 3 entered as an argument when running the GameOfLife program.  
INPUT VALUES: 3  
EXECUTION STEPS: Click on the first cell in the first row and the third cell in the third row. Press the 'Run Continuous' button on the GUI.   
OUTPUT VALUES: N/A  
POSTCONDITIONS: The two red cells die and become green and all others remain grey and empty.

**Traceability Matrix:**

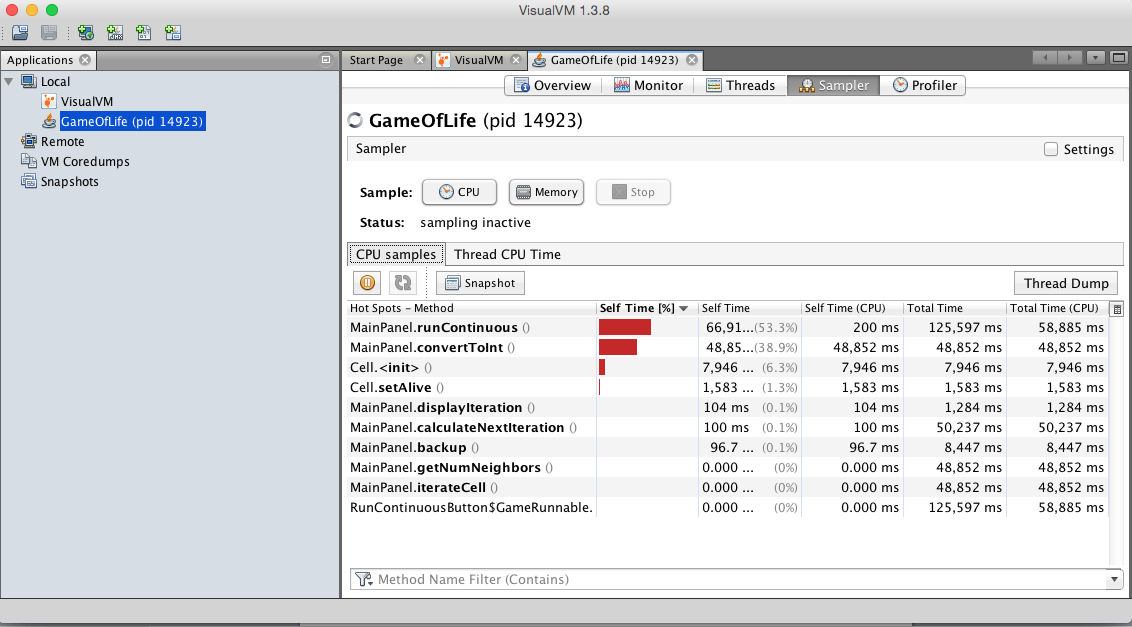
LIVE-CELL-SAME: LIVE-CELL-SAME-1, LIVE-CELL-SAME-2

LIVE-CELL-DIE: LIVE-CELL-DIE-1, LIVE-CELL-DIE-2

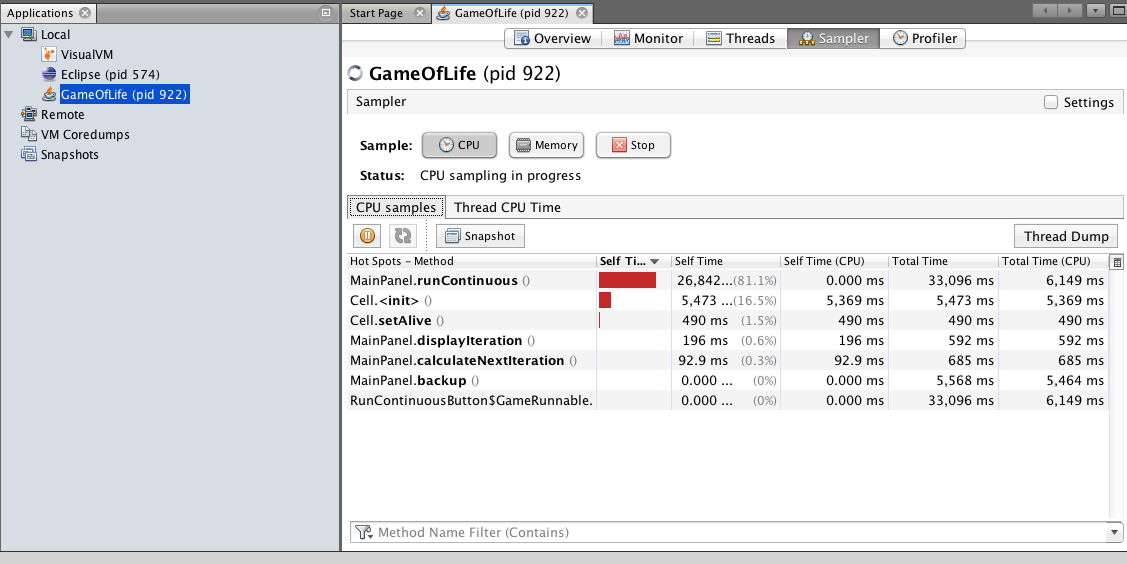
REVIVE-DEAD-CELL: REVIVE-DEAD-CELL-1, REVIVE-DEAD-CELL-2

**VisualVM Profiling Screenshots**

**Before Refactoring:**



**After Refactoring:**



**Screenshot of Automated Pinning Tests**

