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

Michael Bowen

Summary

I started by running the program with VisualVM running and was able to pinpoint a few problem methods to target from the start. I ran it with a continuous run for a while, then tried the write function. The primary culprits at this point were Cell.toString() and MainPanel.convertToInt().

For Cell.toString(), I traced through the code to determine what the end result of the function was. I then wrote a few pinning tests to make sure the results were what was expected. Then, I removed the loop from the function and simplified how the function returned the value it should so that the function would simply return the correct value without doing extra work. Rerunning the pinning tests verified that the functionality had not changed.

I followed much the same process for MainPanel.convertToInt(). Tracing through the code quickly showed that the entire function was redundant – the value that needed to be converted to an int was already an int. Nonetheless, I wrote a few pinning tests to ensure the expected values were returned by the function, then changed the body of the function to simply return the value that was passed to it. Because the convertToInt() function was private, I wrote a debugging function that accepted an int, called convertToInt() and returned the result so that pinning tests could be written for the debugging function. Functionality was not changed and the above two methods were streamlined.

After dealing with those two functions, I reran the program with VisualVM and noticed two other functions seeming to cause problems – Cell’s constructor and MainPanel.runContinuous(). For Cell’s constructor, I found that MainPanel.backup() was creating entirely new objects every time it was called, and it was called every iteration. By tracing the code, I found that the only use for the backup file was to undo the last run, and the undo functionality only needed to know whether the Cell was alive or not. So I refactored the function, after writing a couple pinning tests, to no longer create new objects each time, but to simply update the Cell’s alive status to mirror that of the current run array.

The runContinuous function was another culprit. For this function, I could not quite figure out how to write pinning tests. So I ran a few manual tests where I checked functionality during continuous run and made sure the same things occurred after the refactor. For the refactor itself, I removed the extraneous loop. Afterwards, running VisualVM still shows runContinous as eating the most CPU time, but that’s to be expected as the nature of the method is to run continuously.

Code for the JUnit tests: github.com/mjb236/CS1632/tree/master/Deliverable%204

Manual Tests

Preconditions: There exists a vertical line of three alive cells on the screen.

Execution Steps: Press the Run button.

Postconditions: The top and bottom cells of the vertical should die and the cells immediately to the right and to the left of the middle cell will become alive.

Preconditions: There exists a vertical line of three alive cells on the screen.

Execution Steps: Press the Run Continuous button.

Postconditions: The vertical line of three alive cells should rapidly alternate between a horizontal line of alive cells, centered on the center of the original line, and back to the original line of cells.

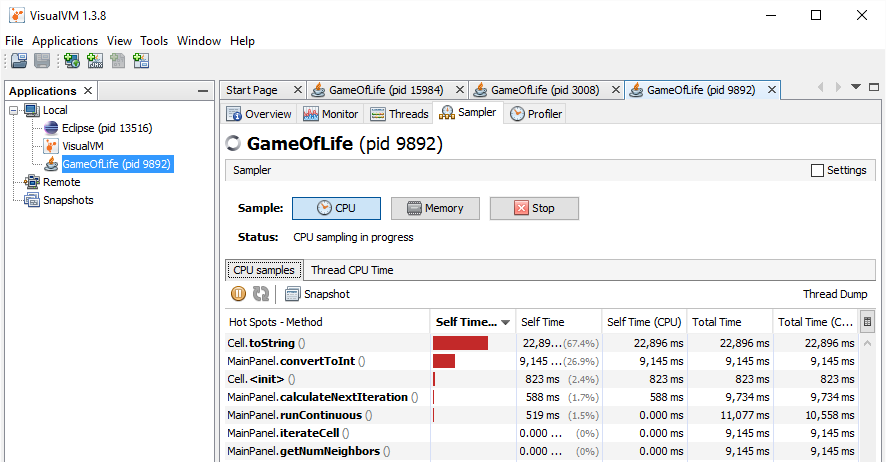
Preconditions: An arbitrary number of cells are alive on the screen.

Execution Steps: Press the Run Continuous button.

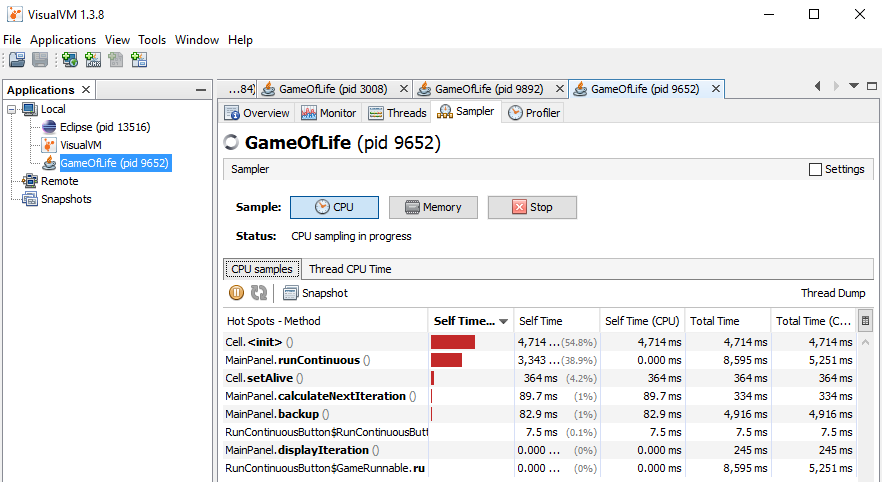
Postconditions: The messages “Calculating…” and “Displaying…” are repeated shown in the terminal, indicating that the program is running continuously.

Screenshots

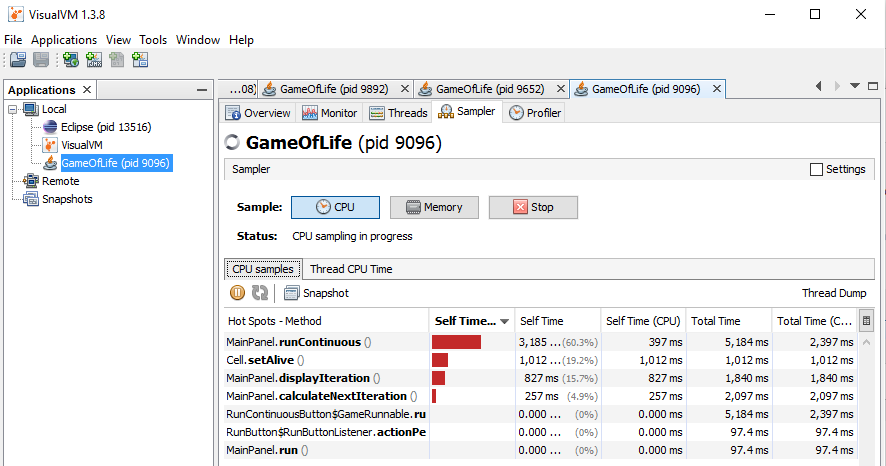
Before any refactor (includes running continuously and then a write):



After refactoring MainPanel.convertToInt() and Cell.toString()(includes running continuously and then a write):



After refactoring MainPanel() and backup() (includes running continuously and then a write):



After refactoring MainPanel.runContinuous()(includes running continuously and then a write):

