Ben Kurzyna

https://github.com/bjk71/QADeliverable4

CS 1632 - DELIVERABLE 4: Performance Testing Conway's Game of Life

Summary:

I approached this deliverable following the guidance of the project description, opening up the application with a size of 15 and beginning exploratory testing, looking for anything that seemed to slow the application down.

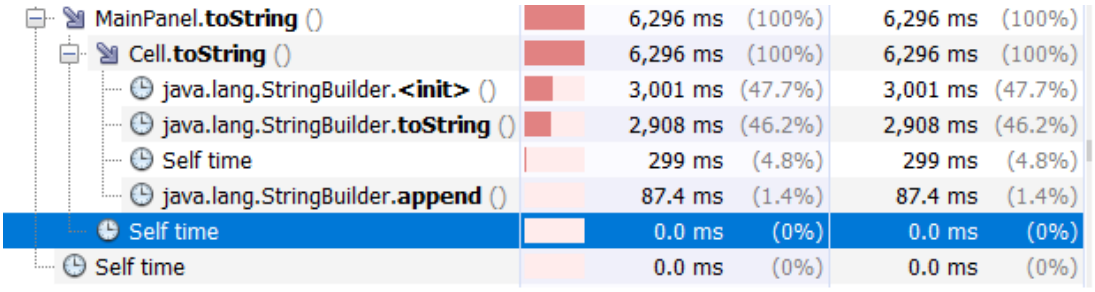
The first thing I noticed in my exploratory testing was that the “Write” button had a brief delay and did not seem to operate completely as expected. I used the profiler to check it out and I noticed that the Cell.toString() method was using up basically all of the cpu time. With this information I went to the Cell file and looked at the toString() method and sure enough, a useless for loop and some unnecessary comparisons were there. The change was simple enough, I just removed the For loop that was doing nothing, and I reconfigured the comparisons that produced return values to make the faster.

Then began exploratory testing once again, quickly noticing that the “Run Continuous” button also seemed to make the program rather slow. I used the profiler to check it out and saw that the ConvertToInt() method in the MainPanel function was responsible for 100% of the cpu time. I checked out the code and noticed that the ConvertToInt() method was doing a lot of work. This took me some time to understand as I was not completely familiar with its application, but I came to the conclusion that essentially everything that was done within the method was useless. It seemed that ConvertToInt took in an integer parameter and simply returned that same integer parameter and served no real point. I removed everything from the method and replaced it with a return statement that simply returned the parameter. Another alternative would have been to eliminate the calls to the function altogether as it did very little for the program, however I felt that my solution was sufficient given the circumstance.

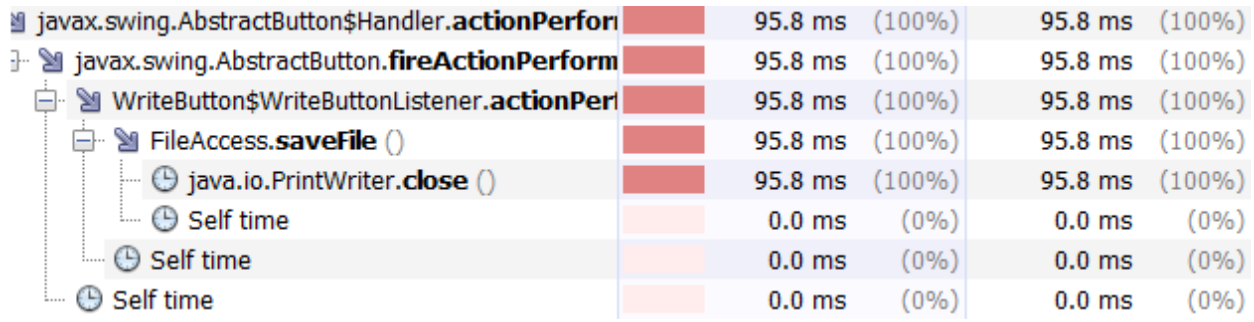
Though I had already removed the ConvertToInt() slowdown previously, I noticed from the profiler that the program was still being bogged down by the RunContinuous() method from MainPanel. At first I thought it was normal since that was what was being executed it made sense that it is what was taking up all the time. However, I checked it out just in case and sure enough the RunContinous() method had a for loop that looped `10000000 times and accomplished absolutely nothing. I simply removed this for loop and the time of the profiler showed the RunContinuous() time was taking significantly less cpu time.`

VisualVM Screenshots:

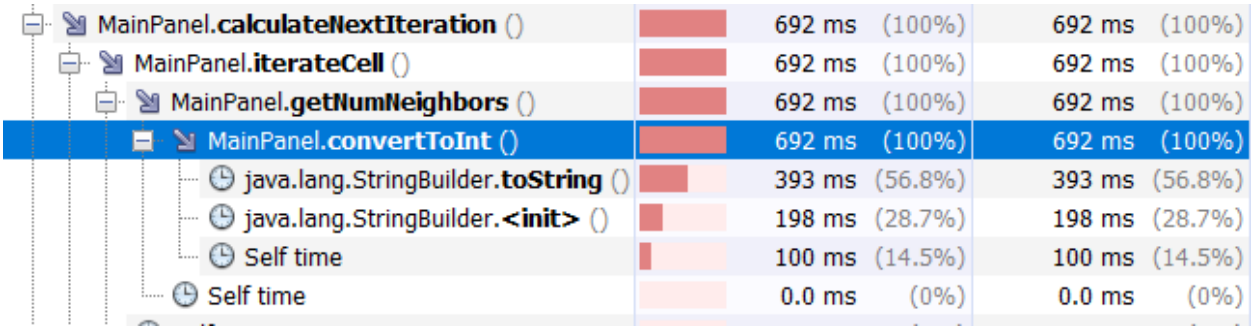
Cell.toString() before:



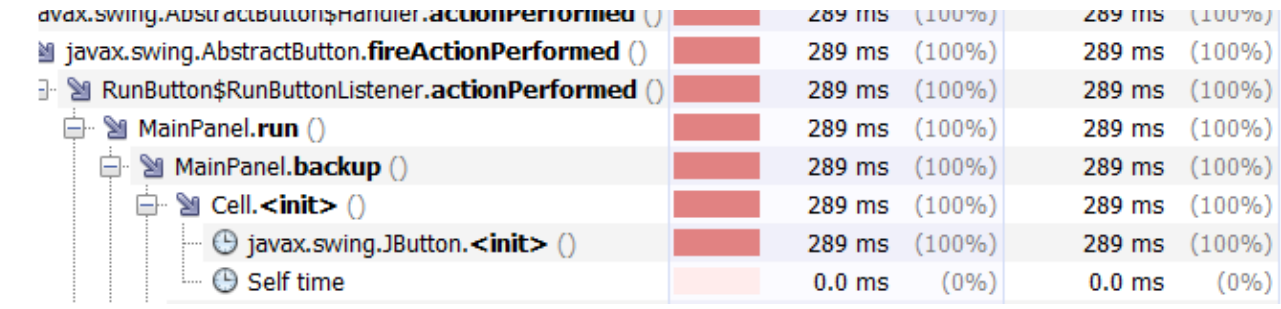
Cell.toString() after:



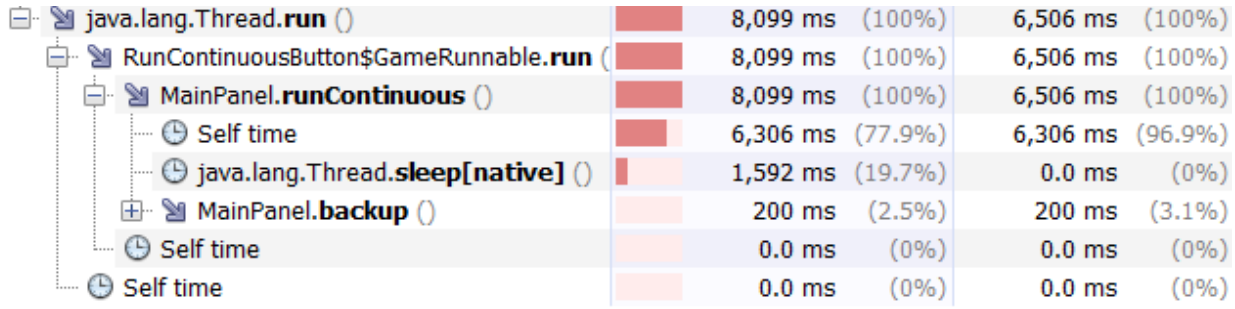
MainPanel.convertToInt() before:



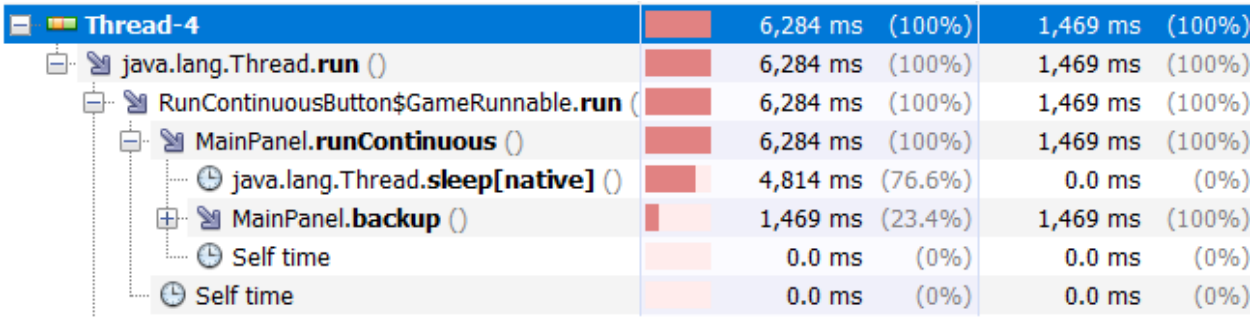
MainPanel.convertToInt() after:



MainPanel.RunContinuous() before:



MainPanel.RunContinuous() after (see self time):



Pinning Test:

For pinning tests I began with Junit tests (in the CellTest.java file) for the toString() method I had fixed. Since the only thing the method was responsible for after the fix was to return the status of the cell. And since there were only three possible statuses for each individual cell (alive, dead, or was alive but is now dead) I simply wrote three Junit tests for each of those possibilities that ensured the output was correct and matched the cell's status.

For the convertToInt() method I ran into a lot of trouble writing the Junit tests. I included one of the versions (in the MainPanelTest.java file) that I had come to put it does not function as expected and does not work. My issues with this were most likely that I had forgotten the necessary syntax for the Junit tests and was not able to find a proper solution using them. Instead, I provided manual test cases for this.

IDENTIFIER: Test-Normal-Int

TEST CASE: Tests that the integer 20 is input and the integer 20 is output.

PRECONDITIONS: The convertToInt() method is available.

INPUT VALUES: The integer 20.

EXECUTION STEPS: convertToInt(20) is called where 20 is an integer.

OUTPUT VALUES: The integer 20.

POSTCONDITIONS: The method convertToInt() should return the integer 20 when the same integer is passed into the method.

IDENTIFIER: Test-High-Int

TEST CASE: Tests that the integer 1000000 is input and that the integer 1000000 is output.

PRECONDITIONS: The convertToInt() method is available.

INPUT VALUES: The integer 1000000.

EXECUTION STEPS: convertToInt(1000000) is called where 1000000 is an integer.

OUTPUT VALUES: The integer 1000000.

POSTCONDITIONS: The method convertToInt() should return the integer 1000000 when the same integer is passed into the method.

IDENTIFIER: Test-Negative-Int

TEST CASE: Tests that the integer -50 is input and that the integer -50 is output.

PRECONDITIONS: The convertToInt() method is available.

INPUT VALUES: The integer -50.

EXECUTION STEPS: convertToInt(-50) is called where -50 is an integer.

OUTPUT VALUES: The integer -50.

POSTCONDITIONS: The method convertToInt() should return the integer -50 when the same integer is passed into the method.

For the runContinuous method writing Junit tests was out of the question. Since the function did not have a basic input or output, I believe the solution using Junit tests would have been far too difficult and impractical, so I provided manual test cases for this as well.

IDENTIFIER: Test-Full

TEST CASE: Tests how the runContinuous() method handles a full screen.

PRECONDITIONS: The program is running in a 5x5 grid.

INPUT VALUES: None.

EXECUTION STEPS: Select every cell in the 5x5 grid and select the “Run Continuous” button.

OUTPUT VALUES: None.

POSTCONDITIONS: Every cell should flash green and then remain that way as the program continues to run and display “Running”.

IDENTIFIER: Test-Empty

TEST CASE: Tests how the runContinuous() method handles an empty screen.

PRECONDITIONS: The program is running in a 15x15 grid.

INPUT VALUES: None.

EXECUTION STEPS: Select no cells and select the “Run Continuous” button.

OUTPUT VALUES: None.

POSTCONDITIONS: Every cell should remain empty and then remain that was as the program continues to run and display “Running”.

IDENTIFIER: Test-Cross

TEST CASE: Tests how the runContinuous() method handles a constantly changing screen.

PRECONDITIONS: The program is running in a 15x15 grid.

INPUT VALUES: None.

EXECUTION STEPS: Select a 3 cell wide and 3 cell high cross in the middle of the board then select “Run Continuous”.

OUTPUT VALUES: None.

POSTCONDITIONS: The cells should expand quickly and then the outer cells should pulse as either “alive” or “dead but was once alive” while the program continues to run and display “Running”.