JavaLife

Clinton Wadley

CS 1632 - DELIVERABLE 5:

Performance Testing Conway's Game of Life

https://github.com/cwadley/SlowLifeGUI

**Summary**

For this deliverable, we were given a Java application that simulates Conway’s Game of Life, but does so inefficiently. Our task was to use performance testing to identify what methods were causing the inefficiencies, and to then refactor them.

I first looked at the code and determined how the classes of the program worked. I focused on the line of execution running through the classes as different functions were triggered, identifying what objects were instantiated and what methods were called. This reading turned up several methods that I suspected were inefficient.

I then used VirtualVM to profile the running application in order to confirm my suspicions about those methods, and identify any others that I had missed. The methods I chose to refactor are listed below:

**MainPanel.convertToInt(int x)**

This method was both unnecessary, since its purpose is to convert an int to an int, and also terribly inefficient, since it created a string of all zeros through 1000 iterations, appended the string value of the input x onto the end, and then sent the string through the integer parser. One peculiar behavior of the method, however, is that it throws a NumberFormatException for negative numbers, because of how it builds the string. Because of this, and also to make testing easier, I did not eliminate the method, but made it public and refactored to simply return the same integer in the case of a positive number, or throw a NumberFormatException in the case of a negative number.

**Cell.toString()**

This method is used to output the current state of the cell. The previous implementation was horribly inefficient because it used 10000 iterations to build a string consisting of the text of the button repeated 10000 times, only to operate on the first character of that string. I refactored this method to eliminate the iterations entirely, and just output the text of the button directly.

**MainPanel.runContinuous()**

This method is called when the “Run Continuous” button is pressed. It has a loop that executes continuous iterations with Thread sleeps between each one. Just before the method iterates the cells, it had a useless loop that iterated 10000 times to modify the variable \_r, only to reassign \_r to its original value at the end. I removed this loop entirely, and the interrupted exception now does not trigger anything. This method was difficult to unit test because of the threads involved, so I have written several manual test cases in the next section as pinning tests.

In addition to these modified methods, I went through and reformatted the code, fixing improper indentation.

**Manual Test Cases**

**IDENTIFIER:** 1

**TEST CASE:** Ensure that a blinker behaves properly under run continuous

**PRECONDITIONS:** A spinner is input on the button panel

**INPUT VALUES:** The user clicks the “Run Continuous” button

**EXECUTION STEPS:**

1. Run “java GameOfLife 15” at the command prompt
2. Click the buttons at coordinates (5, 5), (5, 6), (5, 7)
3. Click the “Run Continuous” button

**OUTPUT VALUES:** The user will see an oscillating cross pattern centered at (5, 6)

**POSTCONDITIONS:** The program will be still in the running condition

**IDENTIFIER:** 2

**TEST CASE:** Ensure that a toad behaves properly under run continuous

**PRECONDITIONS:** A toad is input on the button panel

**INPUT VALUES:** The user clicks the “Run Continuous” button

**EXECUTION STEPS:**

1. Run “java GameOfLife 15” at the command prompt
2. Click the buttons at coordinates (5, 5), (5, 6), (5, 7), (6, 4), (6, 5), (6, 6)
3. Click the “Run Continuous” button

**OUTPUT VALUES:** The user will see patterns alternating between the input shape and a bracket shape

**POSTCONDITIONS:** The program will be still in the running condition

**IDENTIFIER:** 3

**TEST CASE:** Ensure that a block behaves properly under run continuous

**PRECONDITIONS:** A block is input on the button panel

**INPUT VALUES:** The user clicks the “Run Continuous” button

**EXECUTION STEPS:**

1. Run “java GameOfLife 15” at the command prompt
2. Click the buttons at coordinates (5, 5), (5, 6), (6, 5), (6, 6)
3. Click the “Run Continuous” button

**OUTPUT VALUES:** The user will see the block persist with no changes.

**POSTCONDITIONS:** The program will be still in the running condition

**IDENTIFIER:** 4

**TEST CASE:** Ensure that a glider behaves properly under test continuous

**PRECONDITIONS:** A glider is input on the button panel

**INPUT VALUES:** The user clicks the “Run Continuous” button

**EXECUTION STEPS:**

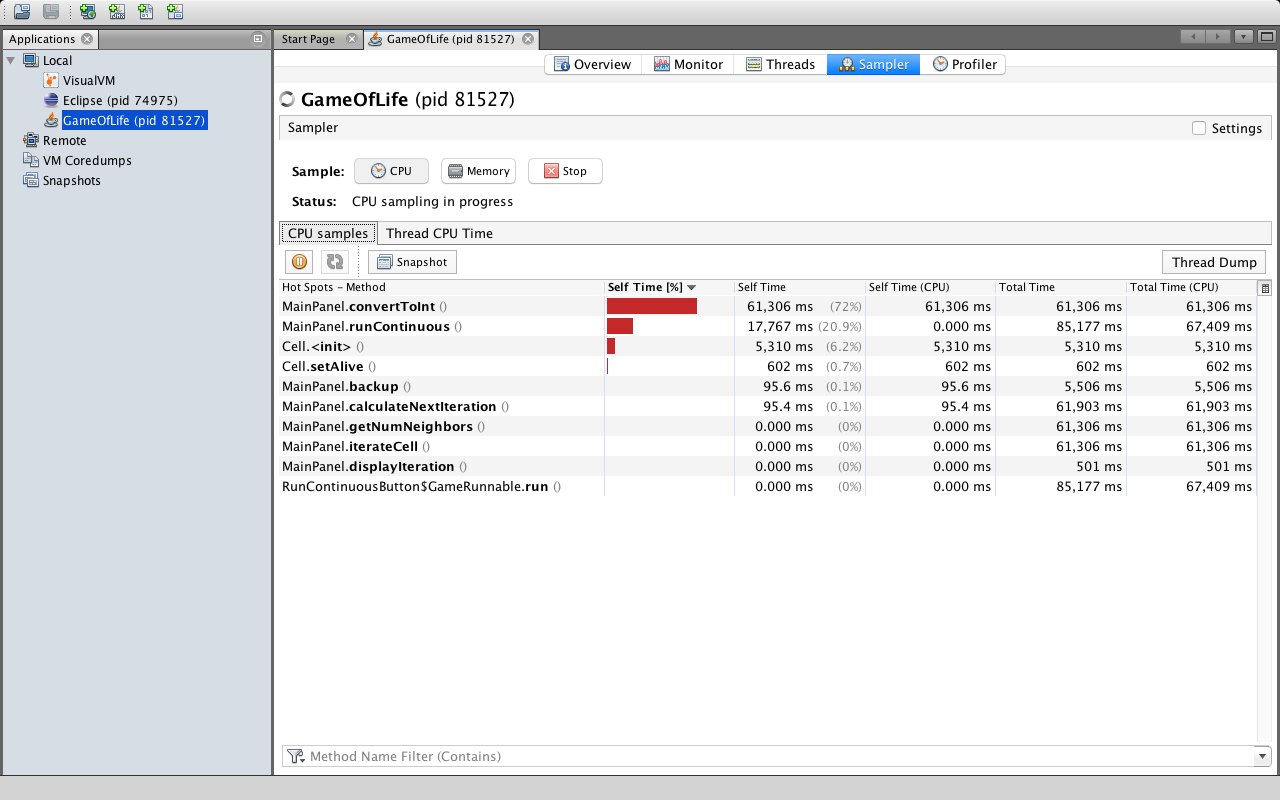
1. Run “java GameOfLife 15” at the command prompt
2. Click the buttons at coordinates (3, 6), (4, 7), (5, 5), (5, 6), (5, 7)
3. Click the “Run Continuous” button

**OUTPUT VALUES:** The user will see the patter oscillating, gliding diagonally across the screen.

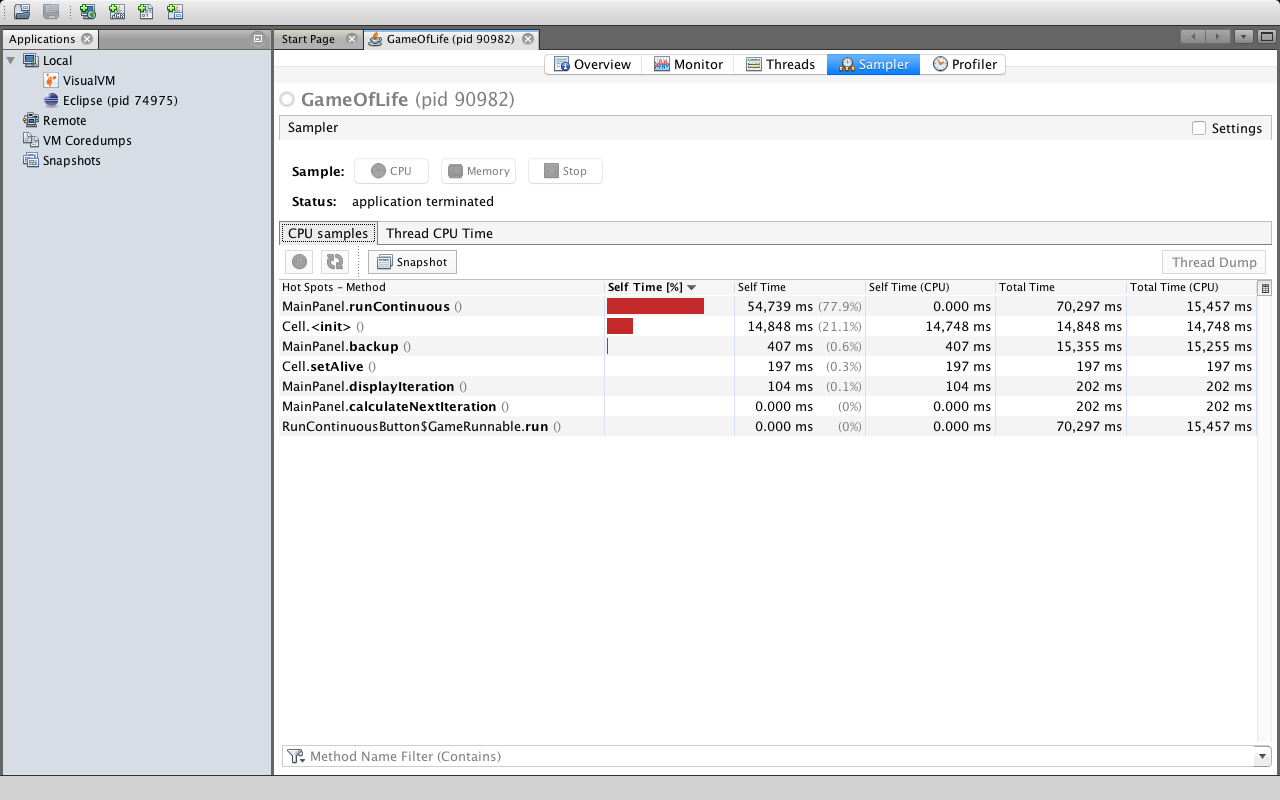
**POSTCONDITIONS:** The program will be still in the running condition

**Screenshots**

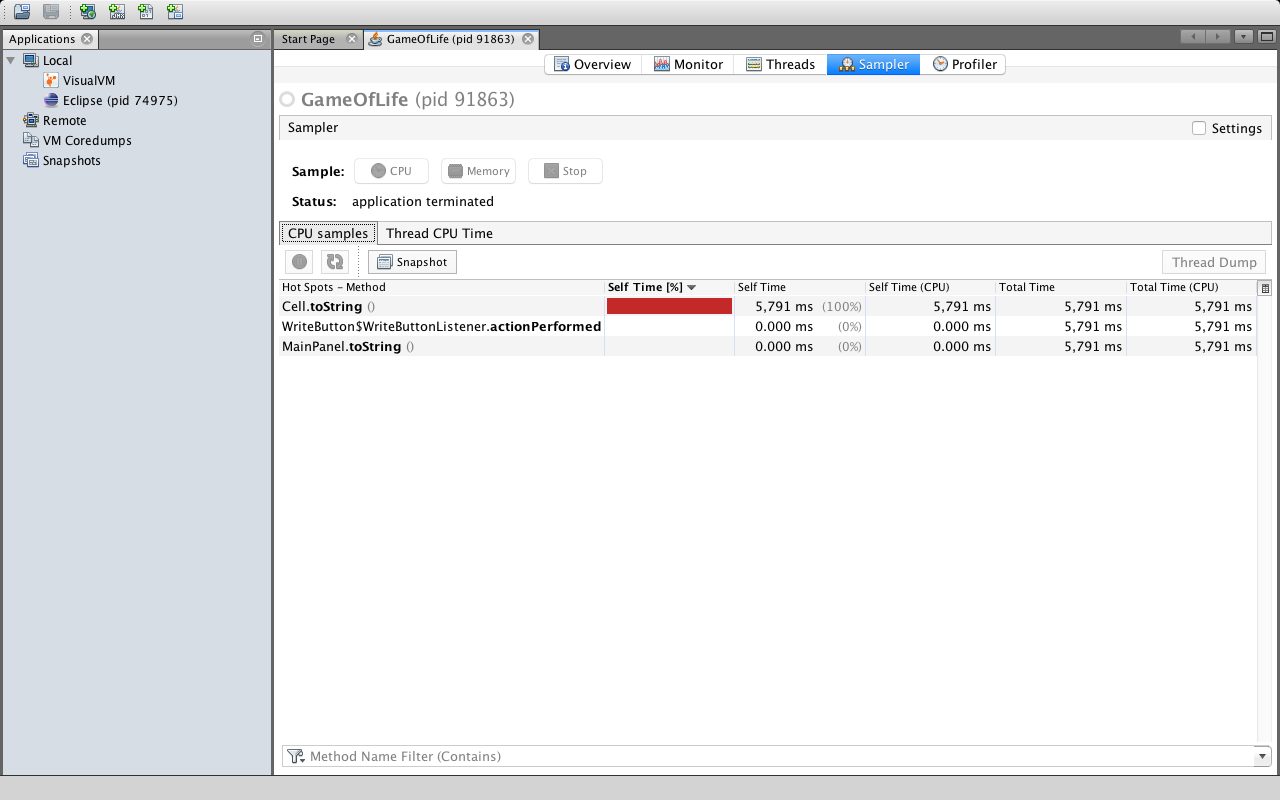
**Before convertToInt() refactor**

****

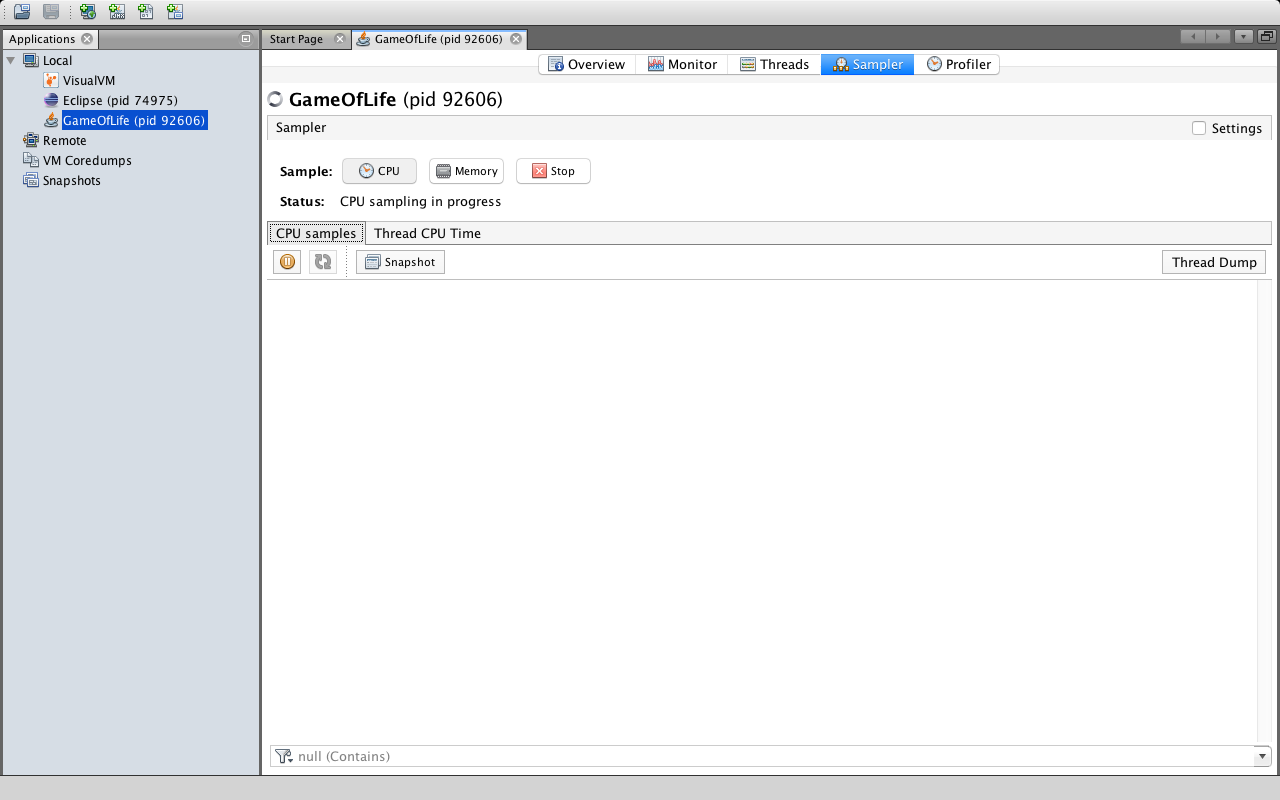
**After convertToInt() refactor**

****

**Before toString() refactor**

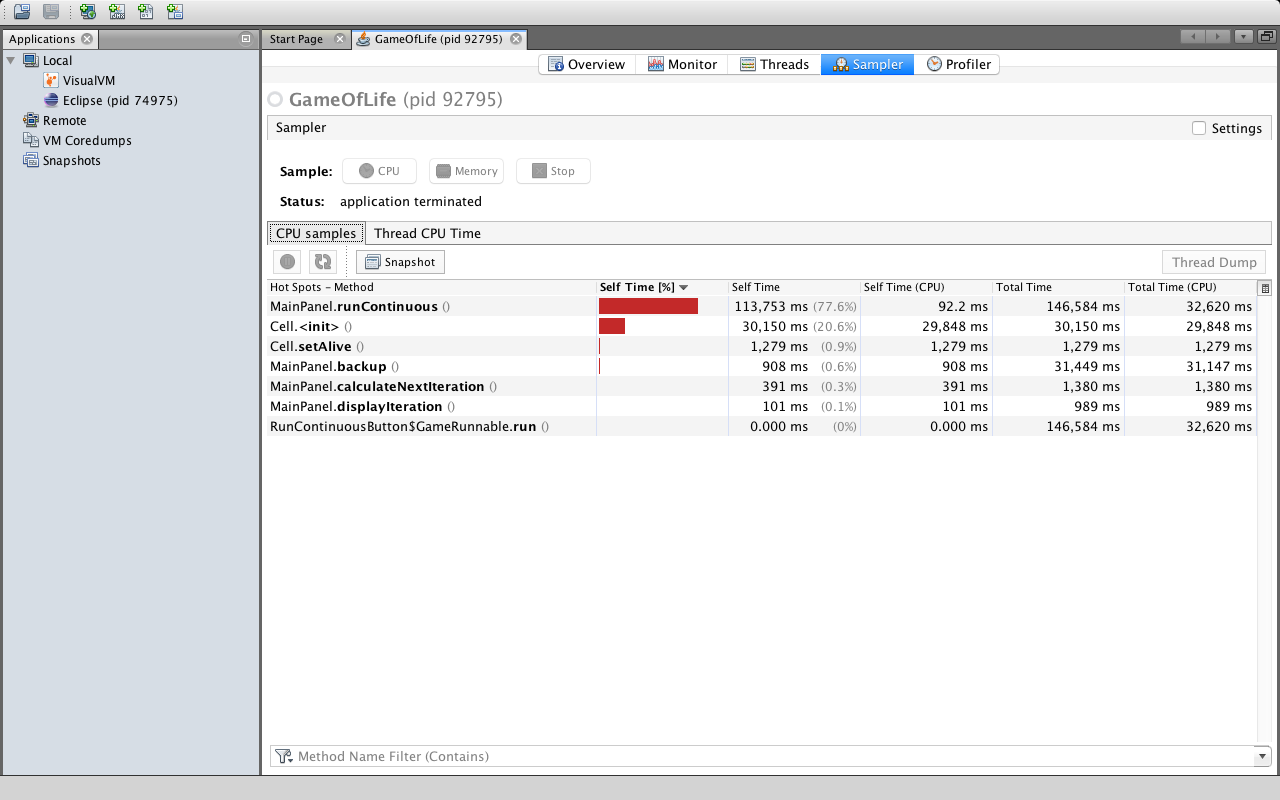
****

**After toString() refactor**

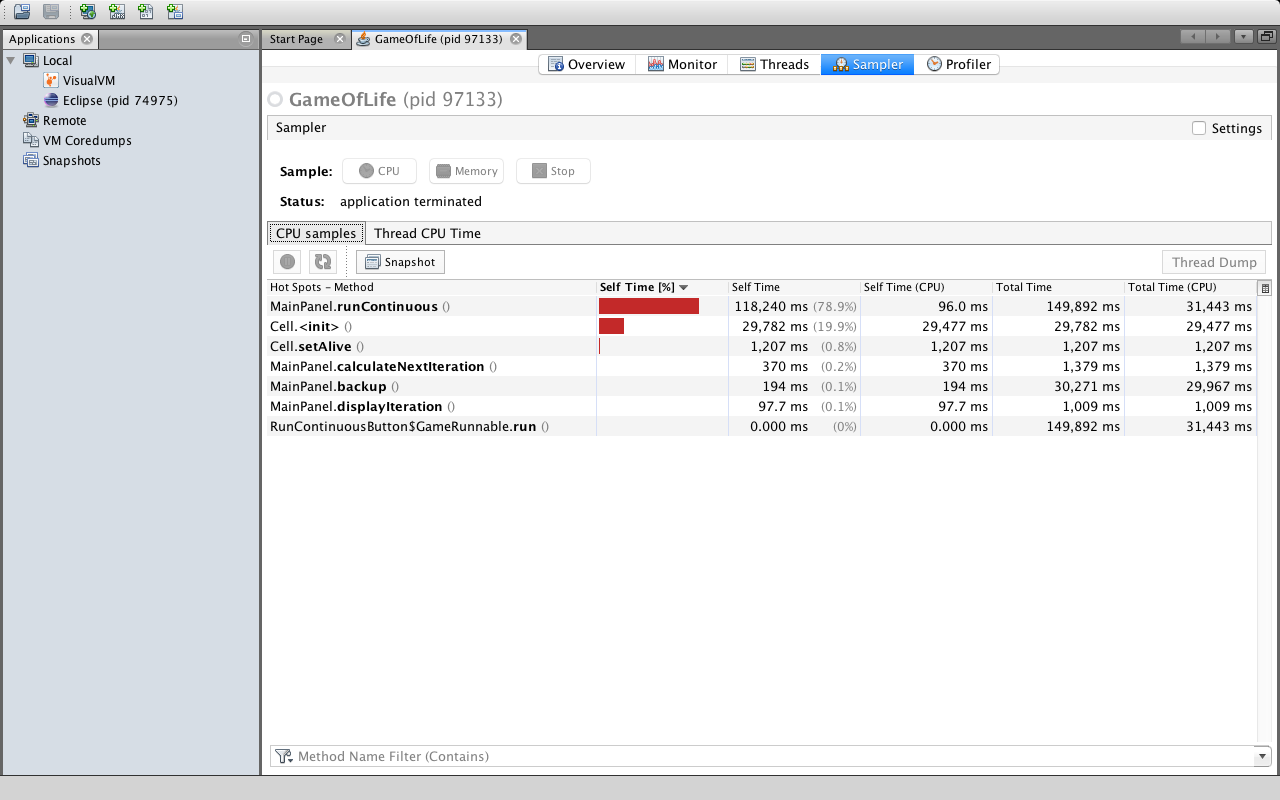
****

The speedup was so significant that the method no longer registered on the sampling screen.

**Before runContinuous() refactor**

****

**After runContinuous() refactor**

****

The overall performance is nearly identical to before the refactor, because the loop execution was insignificant compared to the thread sleep, but the CPU time has decreased.