CS 1632 – DELIVERABLE 5:

Performance Testing Conway's Game of Life

JavaLife

Brandon S. Hang

https://github.com/brandonhang/SlowLifeGUI

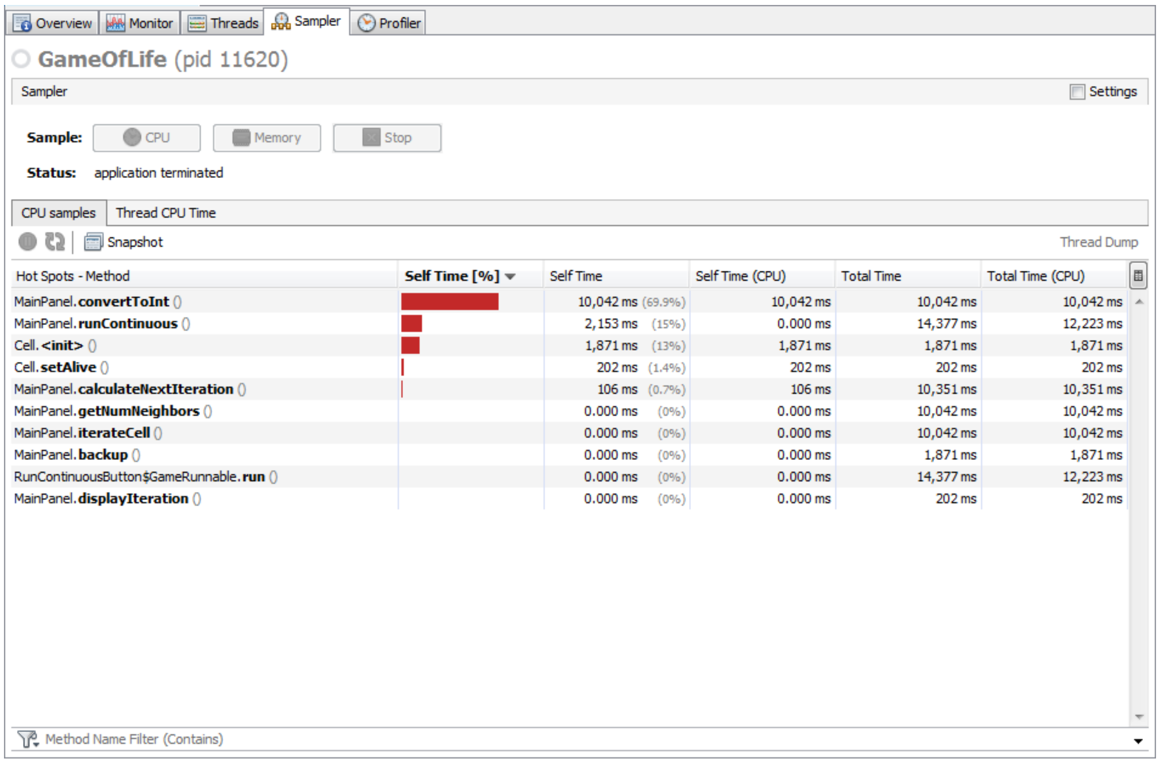
In order to profile this version of Conway's *Game of Life*, I utilized VisualVM and some exploratory testing. While exploratory testing, I noticed that the visuals were quite choppy when the game was run continuously. I also noticed that the "Write" function was slow and took quite some time to complete. I then used VisualVM to find which methods were causing the game to have such poor performance. For the "Run Continuous" mode, VisualVM showed that the MainPanel.convertToInt() method was taking the most time (Figure 1) while the Cell.toString() method was taking the most time when the "Write" function was used (Figure 2).

When refactoring the MainPanel.convertToInt() method, I noticed that it was unnecessarily creating a massive string of zeros (via a loop), appending it with the integer parameter, and then parsing it as an integer. I refactored it to simply return the integer that it was passed unless the parameter was negative. In order to retain the method's original function, I added some code that throws a NumberFormatException if the parameter is negative. The resulting performance increase can be seen in Figure 3. Pinning tests for this method required using reflection as the method was private.

After refactoring the MainPanel.convertToInt() method, I noticed that the MainPanel.runContinuous() method had some unnecessary mathematical calculations that were never used. I refactored this method by removing the calculations and variables involved. The resulting performance increase was, as expected, marginal (a few tenths of a second) since mathematical calculations are generally quite fast to begin with (Figure 4). Writing pinning tests for this test was rather difficult as the method itself runs in an infinite loop and requires external input in order to terminate. I tackled this challenge by using Thread and Runnable objects that enabled me to unit test the method while it was running.

When refactoring the Cell.toString() method, I saw that it grabbed the text of the JButton and built an enormous string via a loop. It then only checked the first character of that string in order to return either "X" or ".". Thus, I refactored this method by removing the string creation code and only checking the first character of the string returned by JButton.getText(). The resulting performance increase was so great that the method could not be detected by VisualVM.

Repo: https://github.com/brandonhang/SlowLifeGUI



GameOfLife (pid 11620)

Sampler · Settings

Sample: CPU · Memory · Stop

Status: application terminated

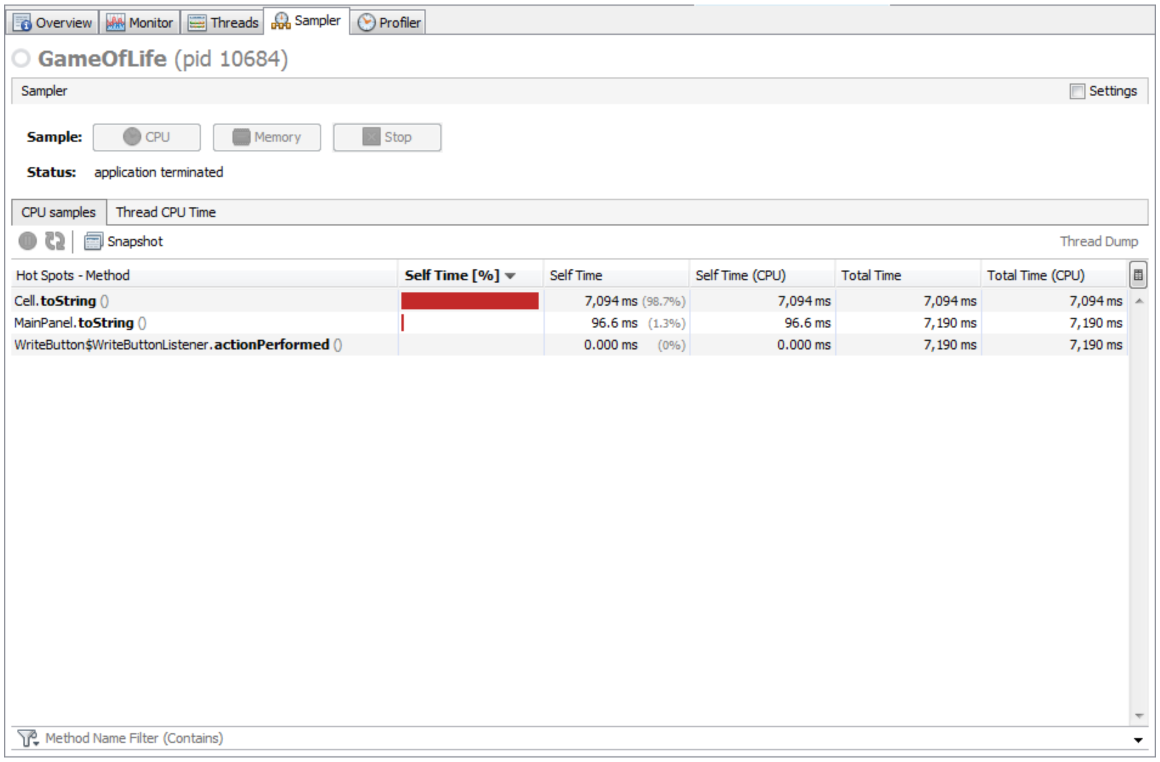
CPU samples · Thread CPU Time

Snapshot · Thread Dump

Hot Spots - Method	Self Time [%]	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)
MainPanel.**convertToInt** ()		10,042 ms (69.9%)	10,042 ms	10,042 ms	10,042 ms
MainPanel.**runContinuous** ()		2,153 ms (15%)	0.000 ms	14,377 ms	12,223 ms
Cell.**<init>** ()		1,871 ms (13%)	1,871 ms	1,871 ms	1,871 ms
Cell.**setAlive** ()		202 ms (1.4%)	202 ms	202 ms	202 ms
MainPanel.**calculateNextIteration** ()		106 ms (0.7%)	106 ms	10,351 ms	10,351 ms
MainPanel.**getNumNeighbors** ()		0.000 ms (0%)	0.000 ms	10,042 ms	10,042 ms
MainPanel.**iterateCell** ()		0.000 ms (0%)	0.000 ms	10,042 ms	10,042 ms
MainPanel.**backup** ()		0.000 ms (0%)	0.000 ms	1,871 ms	1,871 ms
RunContinuousButton$GameRunnable.**run** ()		0.000 ms (0%)	0.000 ms	14,377 ms	12,223 ms
MainPanel.**displayIteration** ()		0.000 ms (0%)	0.000 ms	202 ms	202 ms

Method Name Filter (Contains)

Figure 1. Baseline CPU performance of the "Run Continuous" mode before refactoring.



GameOfLife (pid 10684)

Sampler · Settings

Sample: CPU · Memory · Stop

Status: application terminated

CPU samples · Thread CPU Time

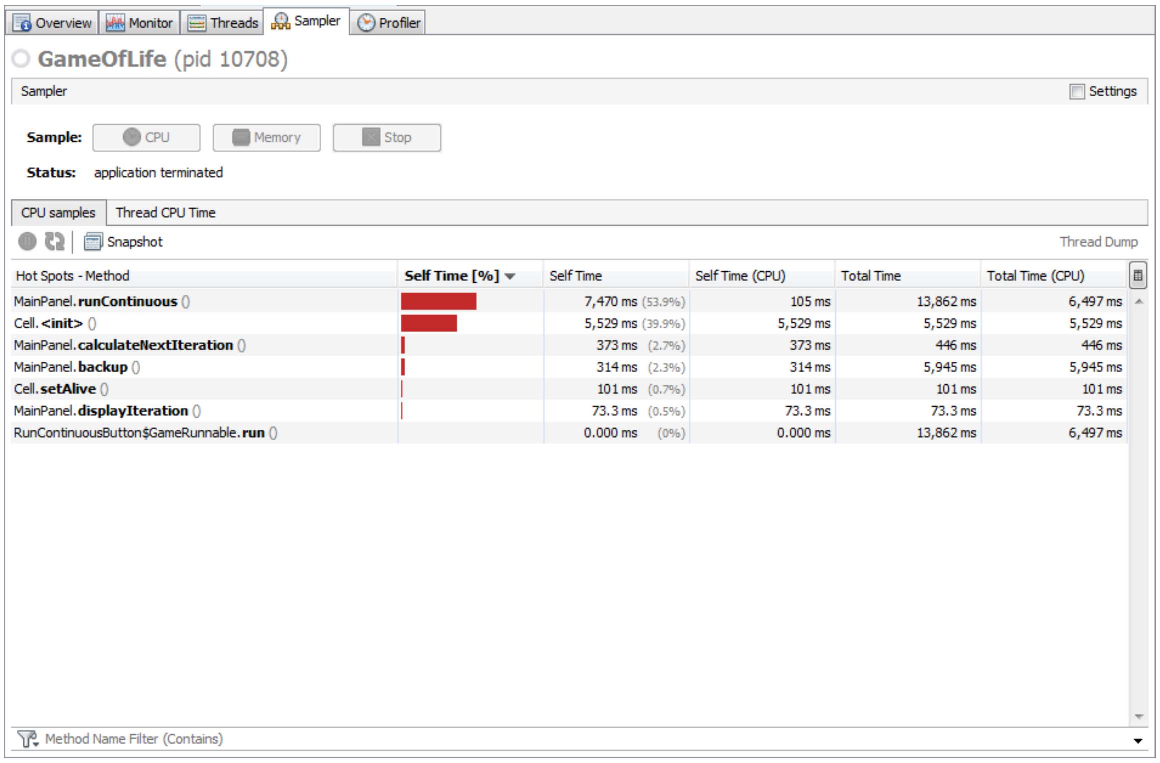
Snapshot · Thread Dump

Hot Spots - Method	Self Time [%]	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)
Cell.**toString** ()		7,094 ms (98.7%)	7,094 ms	7,094 ms	7,094 ms
MainPanel.**toString** ()		96.6 ms (1.3%)	96.6 ms	7,190 ms	7,190 ms
WriteButton$WriteButtonListener.**actionPerformed** ()		0.000 ms (0%)	0.000 ms	7,190 ms	7,190 ms

Method Name Filter (Contains)

Figure 2. Baseline CPU performance of the "Write" function before refactoring.

Repo: https://github.com/brandonhang/SlowLifeGUI



Overview | Monitor | Threads | Sampler | Profiler

GameOfLife (pid 10708)

Sampler

Sample: CPU Memory Stop

Status: application terminated

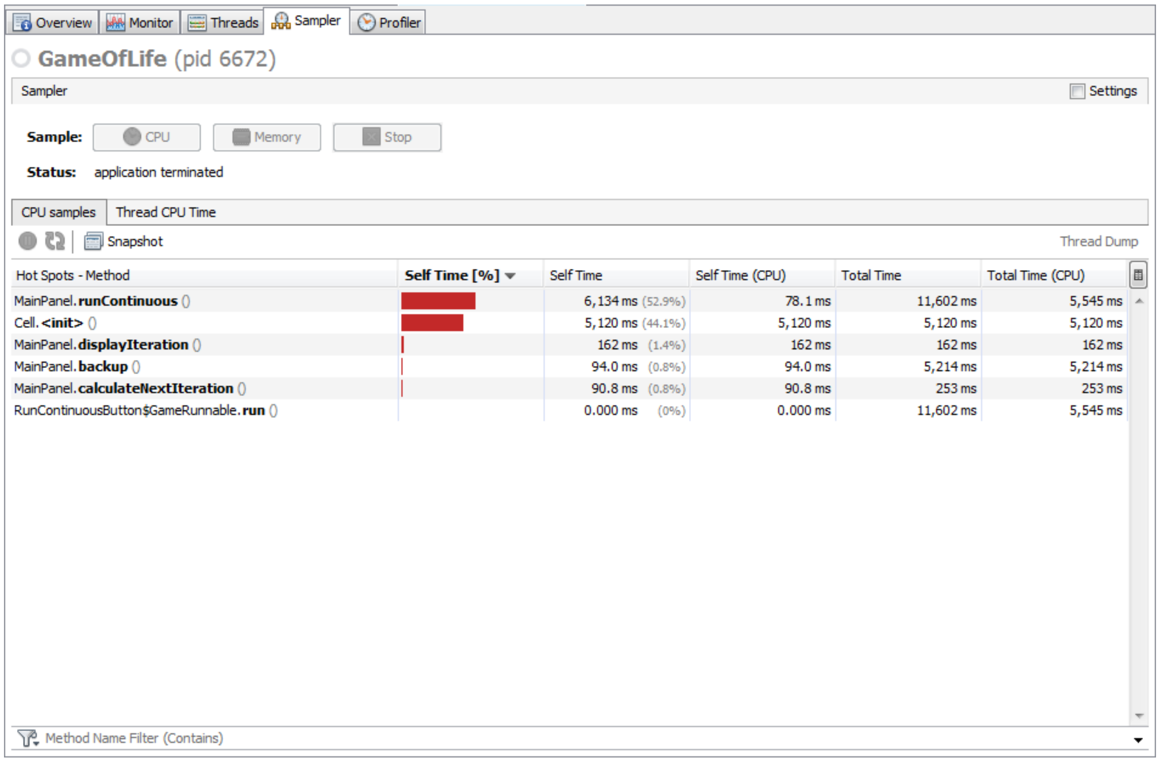
CPU samples | Thread CPU Time

Snapshot — Thread Dump

Hot Spots - Method	Self Time [%]	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)
MainPanel.**runContinuous** ()		7,470 ms (53.9%)	105 ms	13,862 ms	6,497 ms
Cell.**<init>** ()		5,529 ms (39.9%)	5,529 ms	5,529 ms	5,529 ms
MainPanel.**calculateNextIteration** ()		373 ms (2.7%)	373 ms	446 ms	446 ms
MainPanel.**backup** ()		314 ms (2.3%)	314 ms	5,945 ms	5,945 ms
Cell.**setAlive** ()		101 ms (0.7%)	101 ms	101 ms	101 ms
MainPanel.**displayIteration** ()		73.3 ms (0.5%)	73.3 ms	73.3 ms	73.3 ms
RunContinuousButton$GameRunnable.**run** ()		0.000 ms (0%)	0.000 ms	13,862 ms	6,497 ms

Method Name Filter (Contains)

Figure 3. CPU performance of the "Run Continuous" mode after refactoring convertToInt().



Overview | Monitor | Threads | Sampler | Profiler

GameOfLife (pid 6672)

Sampler

Sample: CPU Memory Stop

Status: application terminated

CPU samples | Thread CPU Time

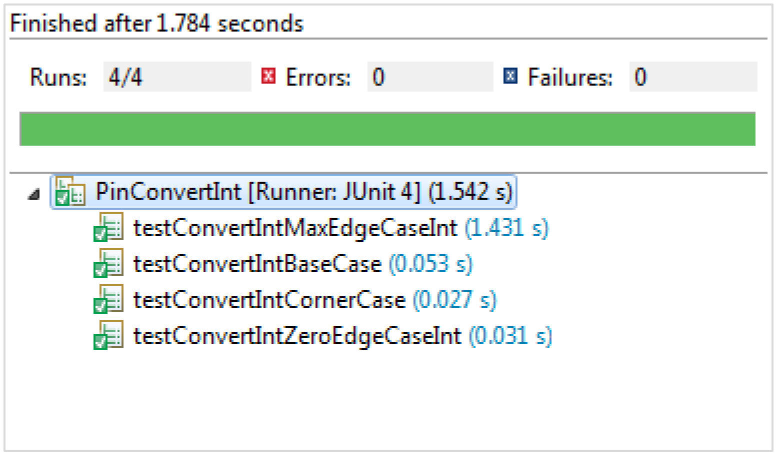
Snapshot — Thread Dump

Hot Spots - Method	Self Time [%]	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)
MainPanel.**runContinuous** ()		6,134 ms (52.9%)	78.1 ms	11,602 ms	5,545 ms
Cell.**<init>** ()		5,120 ms (44.1%)	5,120 ms	5,120 ms	5,120 ms
MainPanel.**displayIteration** ()		162 ms (1.4%)	162 ms	162 ms	162 ms
MainPanel.**backup** ()		94.0 ms (0.8%)	94.0 ms	5,214 ms	5,214 ms
MainPanel.**calculateNextIteration** ()		90.8 ms (0.8%)	90.8 ms	253 ms	253 ms
RunContinuousButton$GameRunnable.**run** ()		0.000 ms (0%)	0.000 ms	11,602 ms	5,545 ms

Method Name Filter (Contains)

Figure 4. CPU performance of the "Run Continuous" mode after refactoring runContinuous().

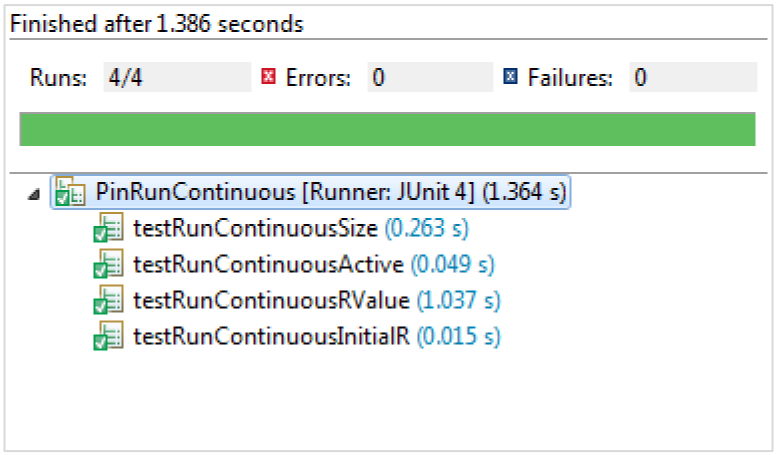
Repo: https://github.com/brandonhang/SlowLifeGUI



```
Finished after 1.784 seconds
Runs: 4/4   Errors: 0   Failures: 0

PinConvertInt [Runner: JUnit 4] (1.542 s)
    testConvertIntMaxEdgeCaseInt (1.431 s)
    testConvertIntBaseCase (0.053 s)
    testConvertIntCornerCase (0.027 s)
    testConvertIntZeroEdgeCaseInt (0.031 s)
```

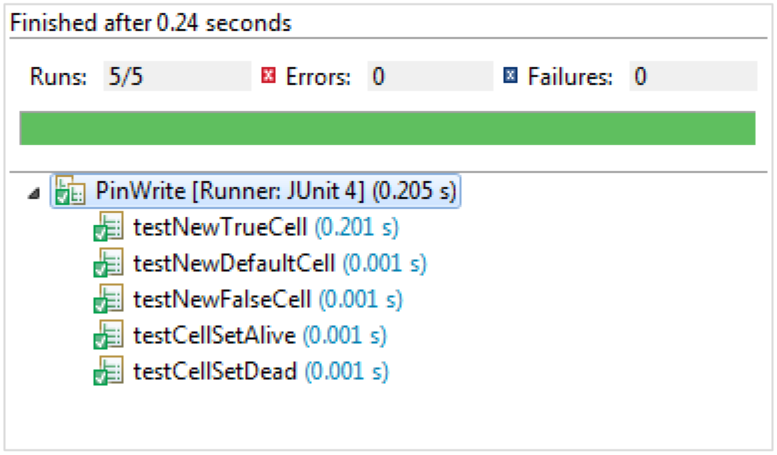
Figure 5. Pinning test for the MainPanel.convertToInt() method



```
Finished after 1.386 seconds
Runs: 4/4   Errors: 0   Failures: 0

PinRunContinuous [Runner: JUnit 4] (1.364 s)
    testRunContinuousSize (0.263 s)
    testRunContinuousActive (0.049 s)
    testRunContinuousRValue (1.037 s)
    testRunContinuousInitialR (0.015 s)
```

Figure 6. Pinning test for the MainPanel.runContinuous() method



```
Finished after 0.24 seconds
Runs: 5/5   Errors: 0   Failures: 0

PinWrite [Runner: JUnit 4] (0.205 s)
    testNewTrueCell (0.201 s)
    testNewDefaultCell (0.001 s)
    testNewFalseCell (0.001 s)
    testCellSetAlive (0.001 s)
    testCellSetDead (0.001 s)
```

Figure 7. Pinning test for the Cell.toString() method

Repo: https://github.com/brandonhang/SlowLifeGUI