University of Pittsburgh

CS 1699 – DELIVERABLE 4: Performance Testing Conway’s Game of Life

Test of JavaLife

Rinat Gatyatullin

CS 1699

Bill Laboon

31 March 2015

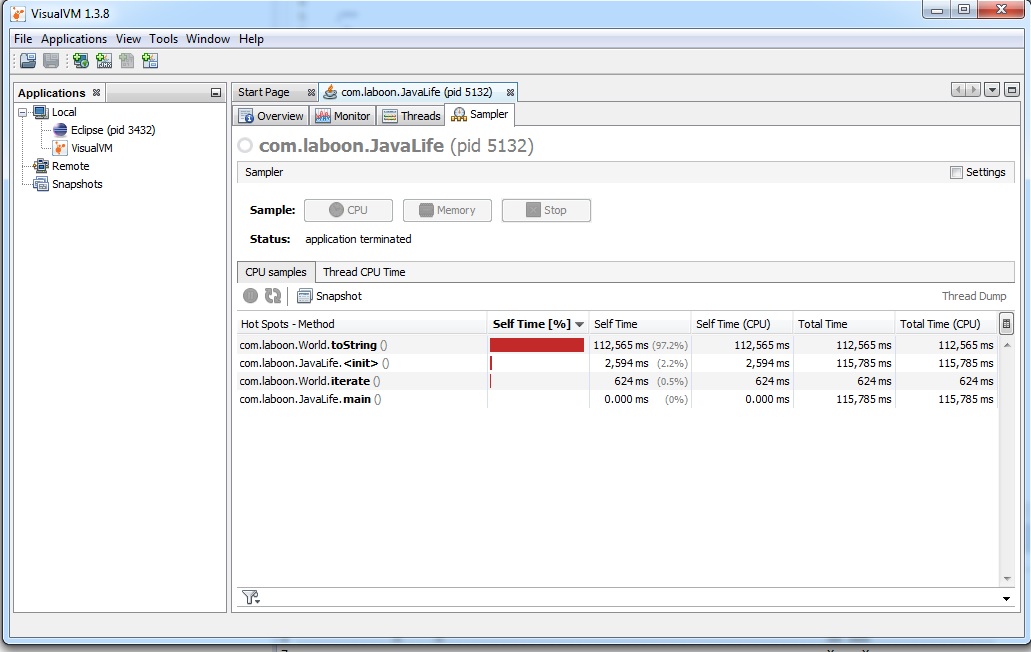
To profile JavaLife, I used the Visual VM profiler to track the CPU usage of the various functions inside of the project. I ran JavaLife with the following arguments: size = 100, seed = 10, percent = 50, and iterations = 3000. When I ran the profiler, I noticed that the majority of CPU time in this program was dominated by the World.toString method. In fact, once the execution was finished the profiler showed me that the toString method took up 97% of all CPU time used by the application. At that point I knew that the toString method was going to be the method that I had to refactor to increase the performance of the application.

When I looked at the code for the toString method immediately I noticed how many times the String was being changed inside the method and I knew that was a problem because Strings are immutable in Java. Which meant that every time the string was being changed the CPU had to create an entirely new String. I knew that StringBuilder and StringBuffer were both ideal for this type of situation and I decided to go with StringBuilder because there was no need for optimization for threading in this application. StringBuilder keeps a sequence/list of changes to the String and doesn’t create the String itself until you call its toString method. By refactoring the function to use StringBuilder it would cut down significantly the CPU time that the function occupied. That proved to be the case, because when I ran the Visual VM profiler again the total CPU usage of the function was cut down to a little over 3% of the entire application run.

For the unit tests I wasn’t entirely sure how to test the World.toString() method, but I decided to go with a few different equivalence classes. I tested the toString using a world of size 0, a living world, a dead world, an alternating world, and finally one of my test cases involved seeing how many String tokens there would be if I used split() on the String returned by toString on a world of a certain size. I tried to hit as many edge cases and values I could think of, but I’m not sure if I did well enough on that part. The unit tests were successful on both versions of the function code.

Some of the biggest challenges involved with this deliverable were definitely the setting up of Visual VM and coming up with unit tests to test the toString method. The set up took a little longer than I expected because I never really installed add-ons to Eclipse before so I had to learn how to do that. Once I finished though, it was pretty easy to use Visual VM itself. Just had to change around some Run configurations. At least now, I will be much quicker when installing any kind of future add-ons for Eclipse. Coming up with unit cases for the toString method also proved to be difficult and I’m not entirely sure if I did well enough. I tried to think of the various kinds of values the toString method might encounter and tried to enumerate those in my unit tests.

Before refactoring screenshot:



After refactoring screenshot:

