JavaLife

Hugh Pedley

CS 1632 – DELIVERABLE 5: Performance Testing Conway’s Game of Life

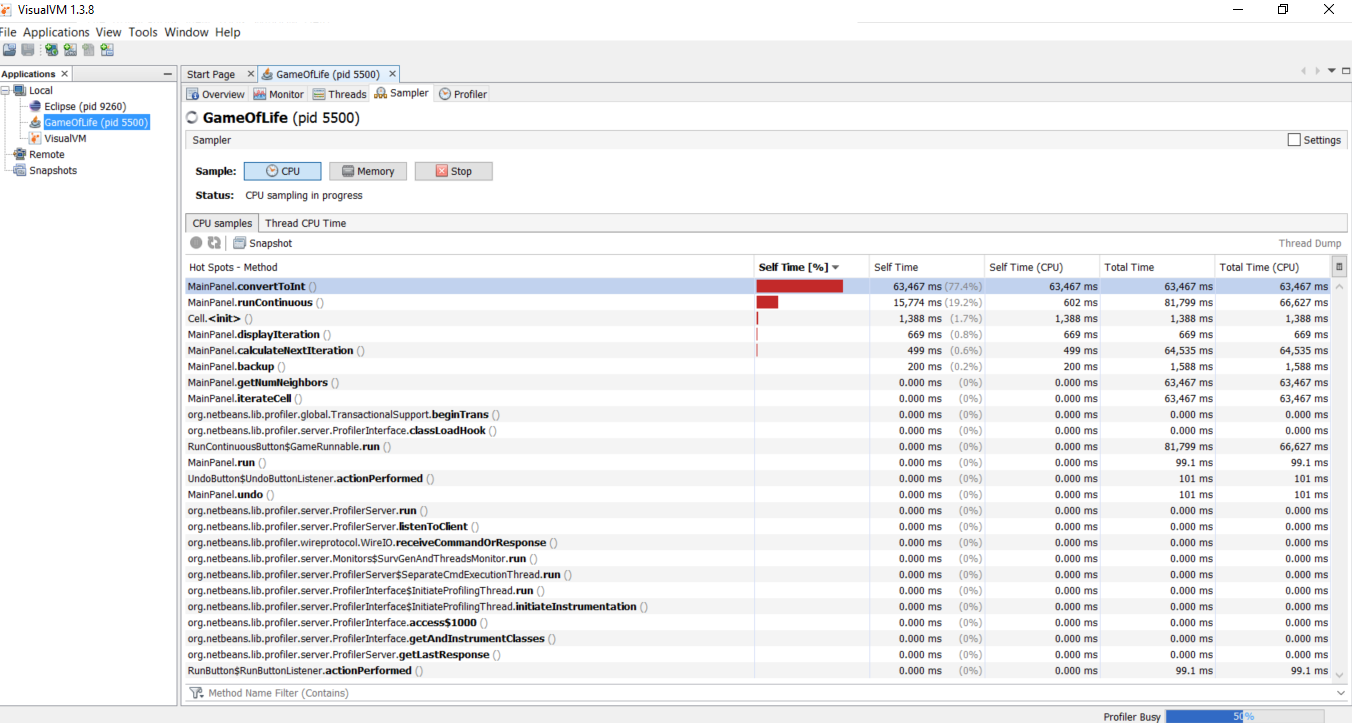
GitHub repository with Deliverable 5 code: <https://github.com/hughpedley/cs1632_deliverable5>

At first glance using only the ‘run’ and ‘run continuously’ buttons on this iteration of Conway’s Game of Life, one method immediately jumps out as using a massively unnecessary amount of CPU time. The convertToInt(int x) method was eating up and inordinate amount of time. Upon looking at the method itself, it was obvious that it simply returned the exact value passed in. Because of this, it was easy to modify without changing its behavior. After refactoring the convertToInt method, it did not even show up on the CPU time sampler.

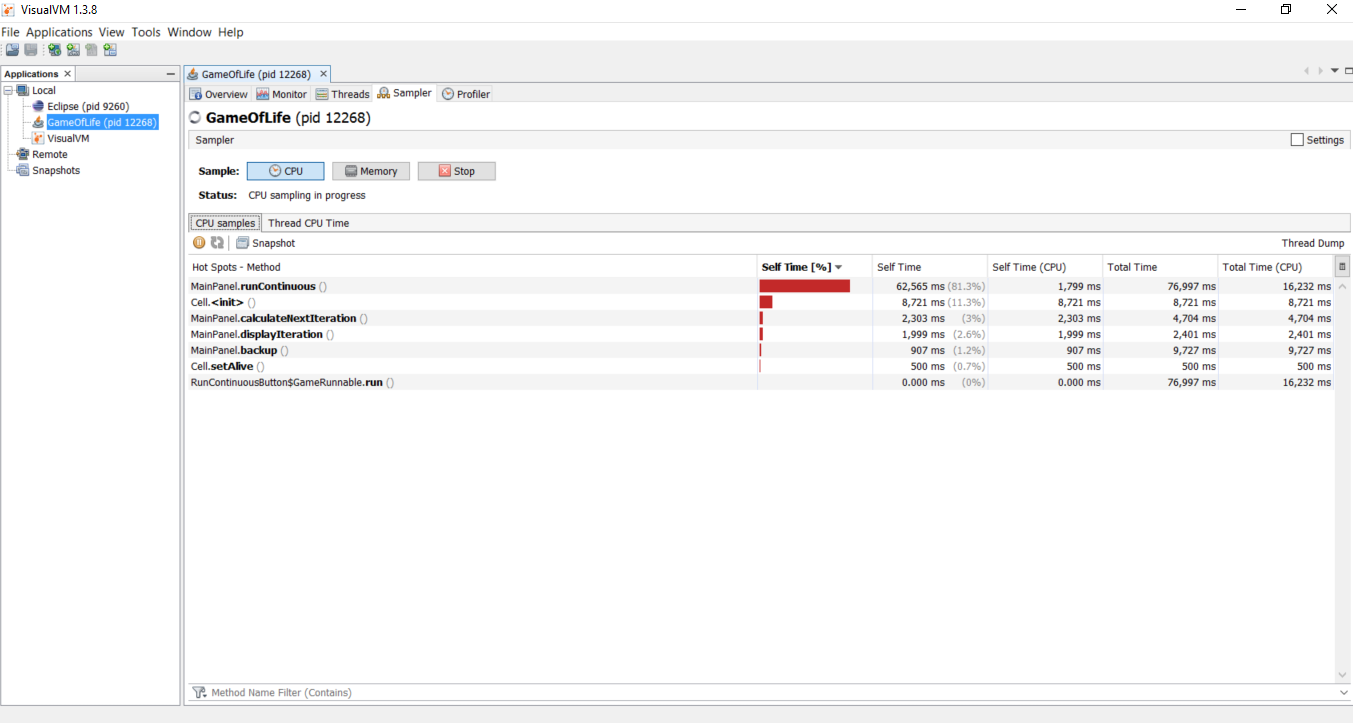
The next method I tackled refactoring was the toString() method in the Cell class. This was a fairly obvious choice because it consumed over 15% of my processor time alone on one click of the Write button, even though Conway’s Game of Life had been running continuously for some time. After refactoring, the toString() method did not clock any significant amount of processor time, indicating that the refactor successfully cut down on wasted actions.

The third method that I decided to change was the runContinuous method. I initially decided not to tackle this method because it made a certain kind of sense that a continuously running method would eat the most processor time. However, it should not have very many actions per cycle, and after seeing how much I was able to reduce the processor time used by other methods, I decided it was worth a shot to reduce the waste in runContinuous. Immediately upon looking at the code for that method, you can see that there is an unnecessary for loop that iterates 10000 times for each full iteration of the while loop. Removing this for loop significantly decreases the self time of the method on the CPU. It should also be noted that I had to do a manual test plan for this refactoring. I have no idea how I would be able to write automatic tests for this. However, I am able to fairly easily write manual pinning tests for this function by ensuring that the display on the Game of Life window matches the original.

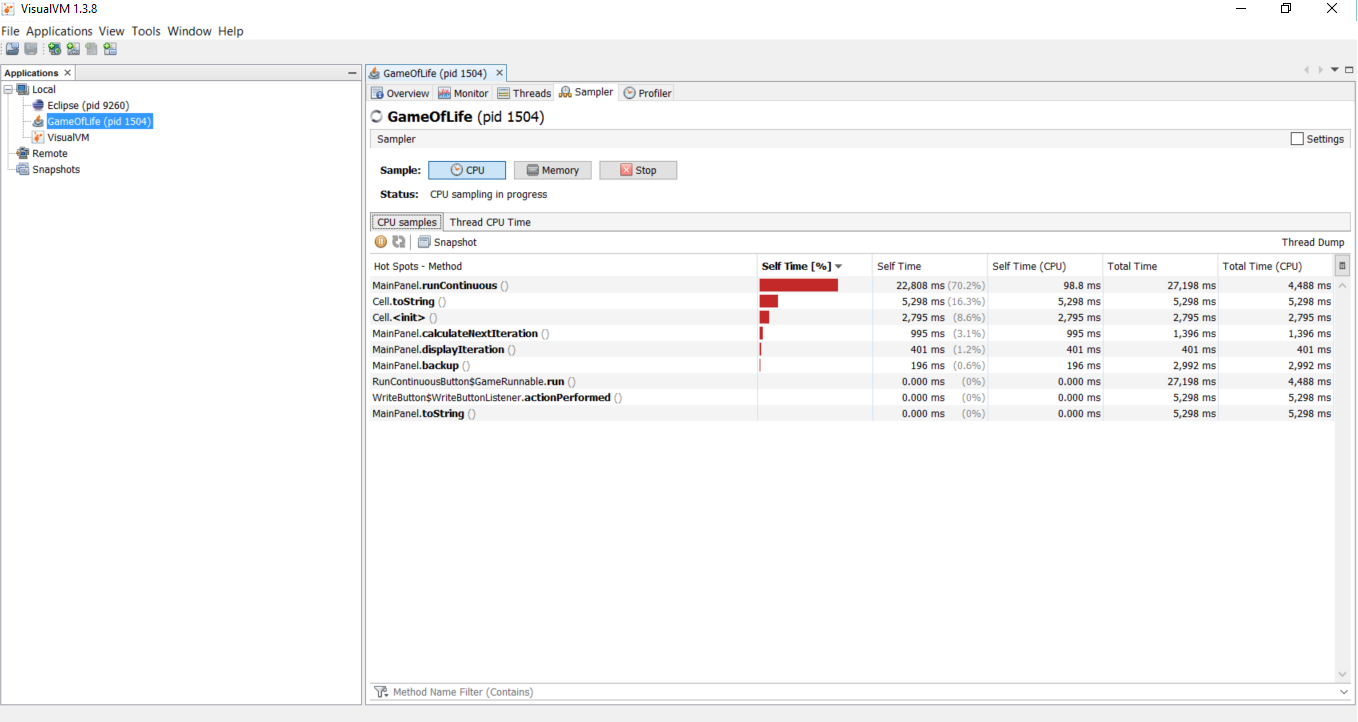
Before convertToInt Refactor:



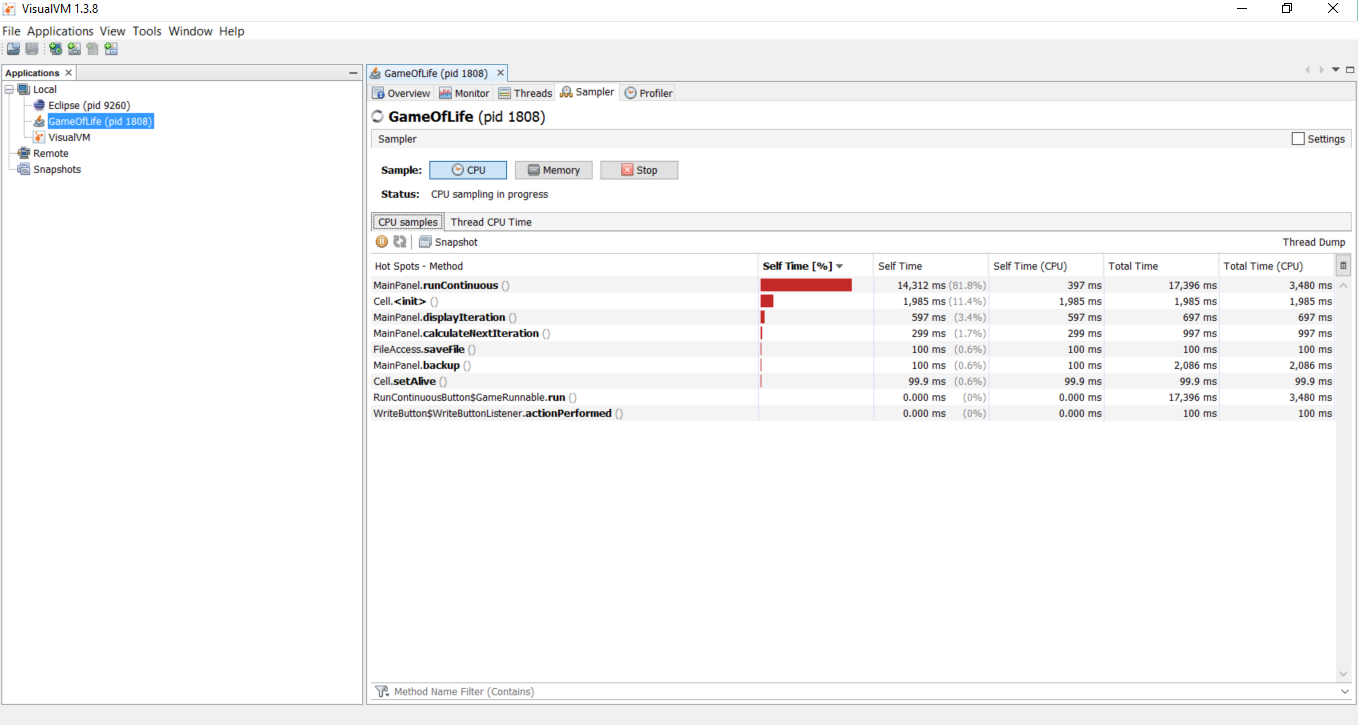
After convertToInt Refactor:



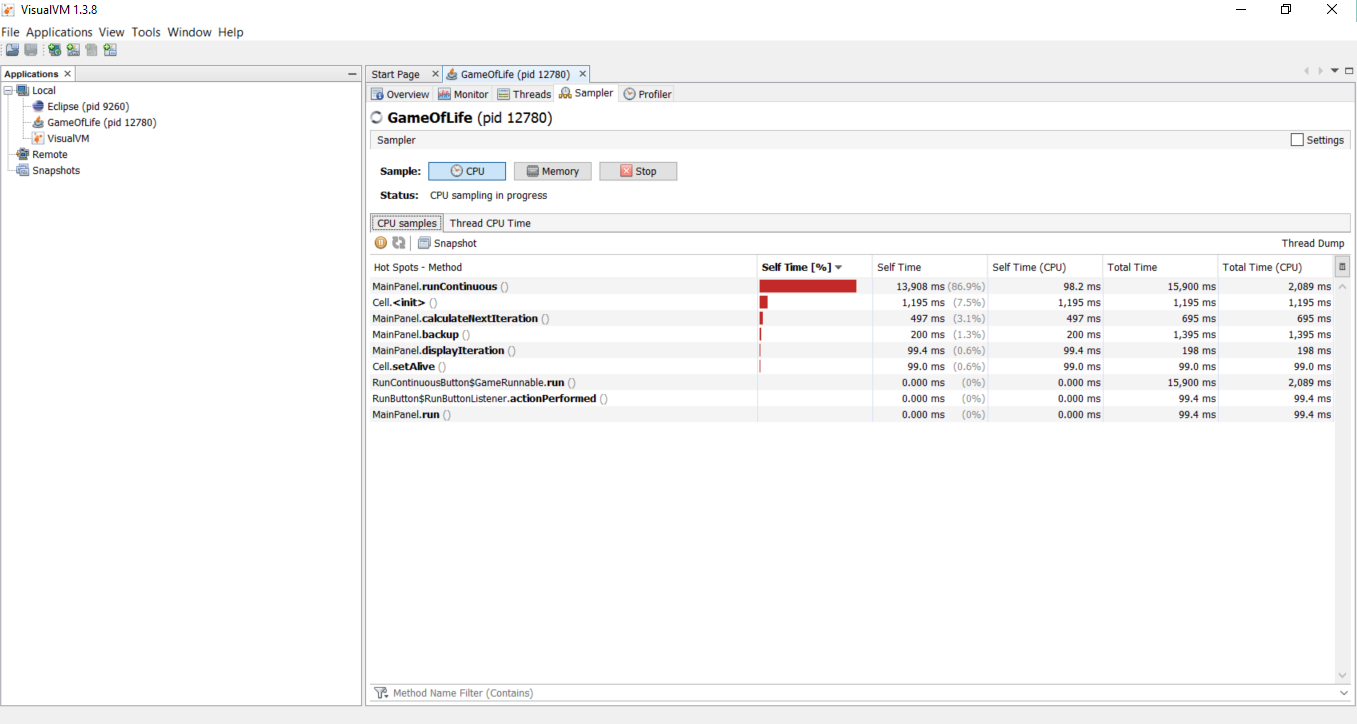
Before toString Refactor:



After toString Refactor:



Before runContinuous Refactor:



After runContinuous Refactor:

