My first step was to simply run and profile the application. I hooked everything up, after a bit of struggle with the JMX, and looked at the sampling. I saw that World Iterate was taking the largest amount of CPU while java init was taking about half as much. Most of the memory was held in Cell[] arrays.

My first thought was to look at the World Iterate() function. I immediately noticed that there was what looked like a recursive call inside of it, which had me thinking that perhaps the problem was a large stack size and a lot of processing required to collapse the recursion. However, I soon realized that the second iterate function was actually Cell.iterate(), so it was not recursion. I was a little concerned about the creation of new cells for every single iteration of the loop, but we could only alter one function and there were no public setters in Cell.

From here I began to look at each function call within World iterate(). GetNumNeighbors was first, and last because that was where I found the problem. There was a loop inside of GetNumNeighbors that seemed to have no purpose. The contents of the loop seemed to be static no matter how many times more than one it was run. I tested this to make sure that my idea was correct, and then removed the loop so that the code only ran once. This lowered the number of runs per iterate call for that section of code in a 20x20 grid from 4 million to 4 hundred.

After running the sampler again, I saw the clear results. The iterate function, instead of being almost twice the CPU of init, was so small that it almost did not show up next to java init.