Members: Gregory Hartranft (gdh12)

CS 1632 - DELIVERABLE 5: Performance Testing Conway's Game of Life

Github: https://github.com/Sindalf/SlowLifeGUI

For deliverable 5 I changed four functions to achieve better performance. The first was ConvertToInt. Testing shows that this was just one of your troll functions that just returns the value that you input making all the code for it completely worthless. Commented that out and returned X.

Next I changed the backup code. Profiling showed that this was the biggest bottle neck in the entire system. It would run at every single iteration of commands. However, the reason it was slow was because for every Cell in \_backup you would create a new cell from scratch. This works very well for making things easy but is slow. Instead I initialized the \_backup array in the main function and then just assigned proper values in the backup function. This cut down the time spent in that function significantly. It still requires some decent time due to how often it is called but its lower than it used to be. Plus, this function time will grow as your grid size grows while many other functions will not grow as fast making this function the most problematic. I'm sure there is another way to handle this function in a more efficient way but I tried to not redesign everything if at all possible. You could change it to only backup when a change in a cell is detected so you don’t have to iterate through the entire array every single time.

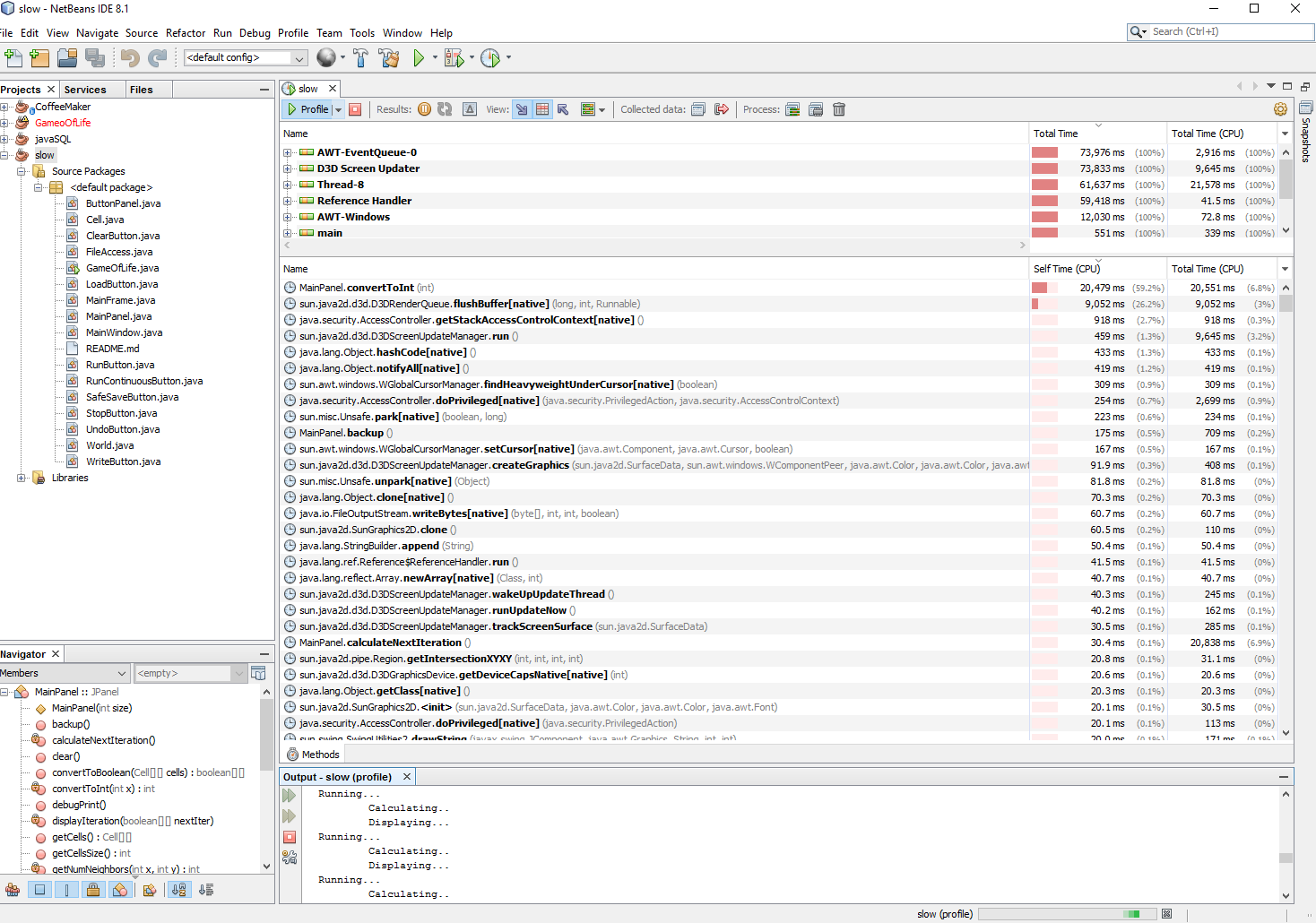
Lastly I noticed toString was rather slow whenever I wanted to write to disk. It's pretty obvious from looking at it that the old tostring code was just another set of pointless clutter to simulate bad code. However, upon at the function and at the cell object overall I noticed there were a lot of string comparisons, mostly to "X". I changed all instances of that to a new Boolean variable that tracks if it’s alive or not. setalive and getalive would eventually begin to take up time due to all the comparisons. However, now it just checks against a Boolean value which is simple enough leading to a good performance increase. This also helped make the toString function fast instead of slow.

One of the things to note that I didn't optimize or spend a lot of time on is load, undo, and clear. Unless these functions proved through much testing to be exceptionally slow like write they were not as much of a priority because they are not running as often as the others. First couple bits of testing showed nothing wrong those functions anyway. At the very end the thing that was taking up the most time was \_backup still due to it running every iteration or the GUI itself. However, I didn't attempt to change GUI elements since I don't have a lot of experience in that area and most of it is just swing and awt anyway.

Screenshot details:

Tests were done running continuously for 1min using an oscillator. I believe the slowness in the GUI is because of all the initialization of GUI elements or due to constantly calling GUI elements such as gettext. This was resolved in my faster implementation by fixing one of these elements. Unsure of what calls D3DRenderQueue but I assume its initialization of GUI elements.

Slow.png



Fast.png

