Vishnu Chittari
Yufei “Edward” Du

Project 6: Concurrency Assignment

List of Files:

A61: The version using Threads directly
A61/Life.java
A61/Coordinator.java
input.txt

A62: The version using Executor
A62/Life.java
A62/Coordinator.java
input.txt

README.pdf

How to compile and run
cd A61 or cd A62
javac *.java
java Life -t 4 -s 30000 (--glider) (the numbers for t and s can change)

Extra Credit: java Life -t 4 --file <filename>

A61 Description:

The starter code was provided. We modified the starter code so that instead of only initiate a single Thread, the program will create t Threads and run in parallel. If there are 100 rows and 10 threads, the program will assign row [0, 10, 20, 30, 40, 50, 60, 70, 80, 90] to the first Thread, row [1,11,21,31,41,51,61,71,81,91] to the second Thread, etc. The program uses a CyclicBarrier to keep synchronized. After a generation is done, the CyclicBarrier runs a inline Runnable that sync the board and the temp board.

A62 Description:

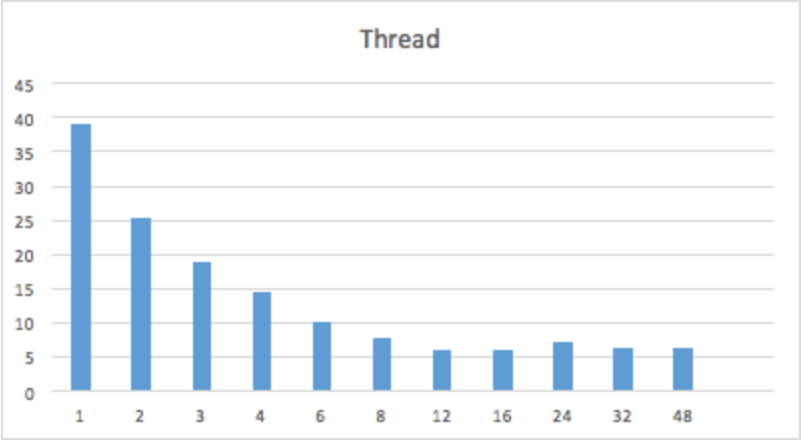
We started our code from using the previous code for A61. We used the built-in executor methods to force all extant tasks to complete before starting the next generation. The way jobs are assigned to each thread is the same with A61. For this program, we used the invokeAll()

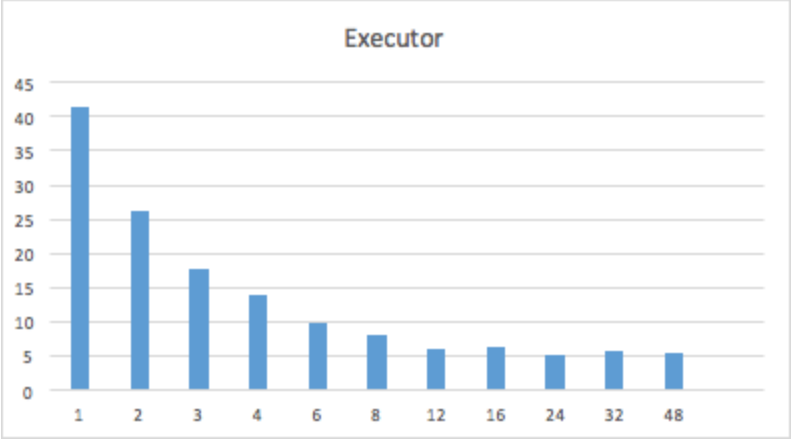
method from the ExecutorService class to keep synchronized. After a generation is done, the program calls LifeBoard's update() method to sync the board and the temp board.

Extra credit:
We added a button that on both the thread version and the executor version so the application will step exactly one generation. The button is called "step".

We also added an additional functionality where the points can be read from a file. In our case our data is in input.txt. See input.txt for the input format

Here we have the graphs of the thread and the executor versions. The graphs are plotted as Time (in seconds) vs number of Threads. We measured each number of Threads twice and took the average. We did notice that when we were testing on node2x14a server, there were other users also logged in, so we think that there may be some inaccuracy because of other people testing their program at the same time (in our time slot).





We found that the executor version was a little faster than the thread version for Thread numbers larger than 3.

Here are the graphs of the run time of the unmodified version divided by the run time of our Thread and Executor implementation (speed up) vs number of Threads. The unmodified version of the program took 41 seconds to run. There are significant speed ups as the number of threads increases, but it does not match the ideal value most likely because of thread creation overhead and the factor that the usage of cache is not optimized.