COMP3712 ASSIGNMENT 3

The Game of Life

Timothy Foong foon0020
Foon0020@flinders.edu.au

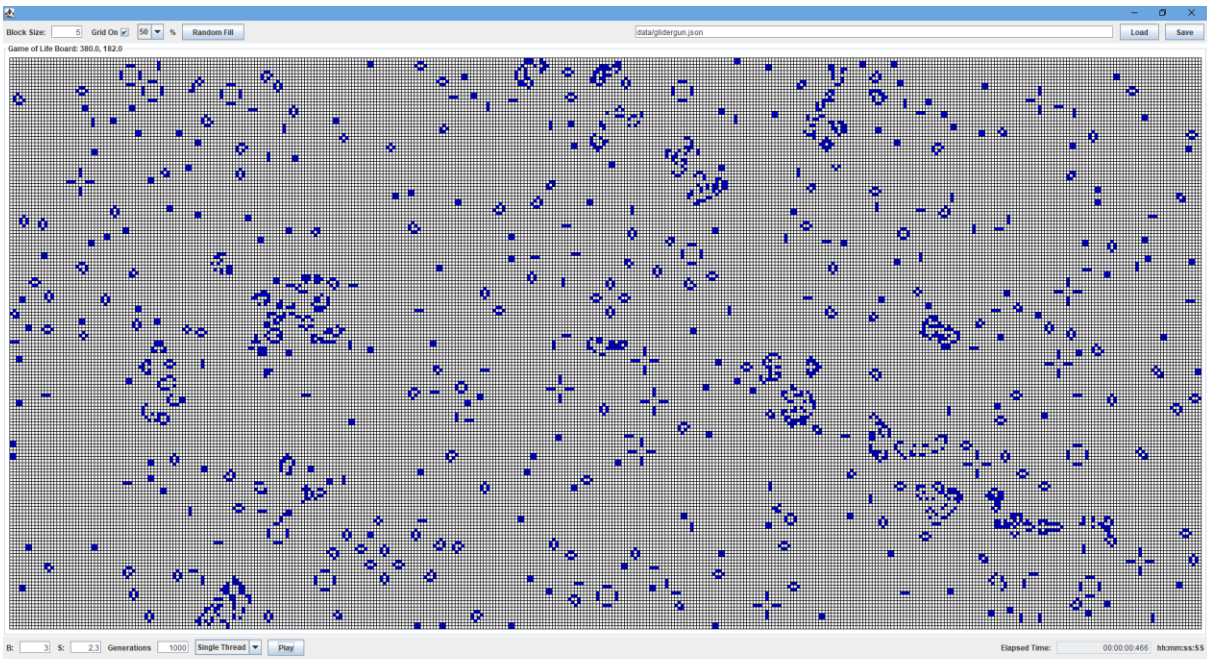
Objective 1: A Responsive GUI

I was unable to modify the GUI to make it more responsive, nor was I able to add a progress bar to the GUI. Looking at the code I could not understand how it worked, I assumed that the listeners in the GUI code tracked the buttons being pressed and the changes in variables that accompanied it, but I was unsure of how to modify it and my attempts did not result in any changes.

Objective 2: Multi-Threaded Java Life Simulation

To multi-thread the simulation I used ExecutorService to create a fixed thread pool. I then separated out the board into sections based on how many threads I had, and each thread will work on the columns given. It will submit the LifeThreads function to the pool and add the returned list of Points to the point variable which will update the game board. To test this program, I ran it at full screen to have a large consistent game board. The larger the board the more pronounced the effects of multithreading would be. The game board was 380 by 182 squares, and the simulation was run for 1000 generations. The computer that ran the simulation is running an AMD Ryzen 5 5600X 6-Core Processor that runs at 3.70 GHz and 16 GB of RAM. Below are some screenshots showing the program running with 1, 2, 4, 8, 16, 32, 64 and 128 threads.

1 thread

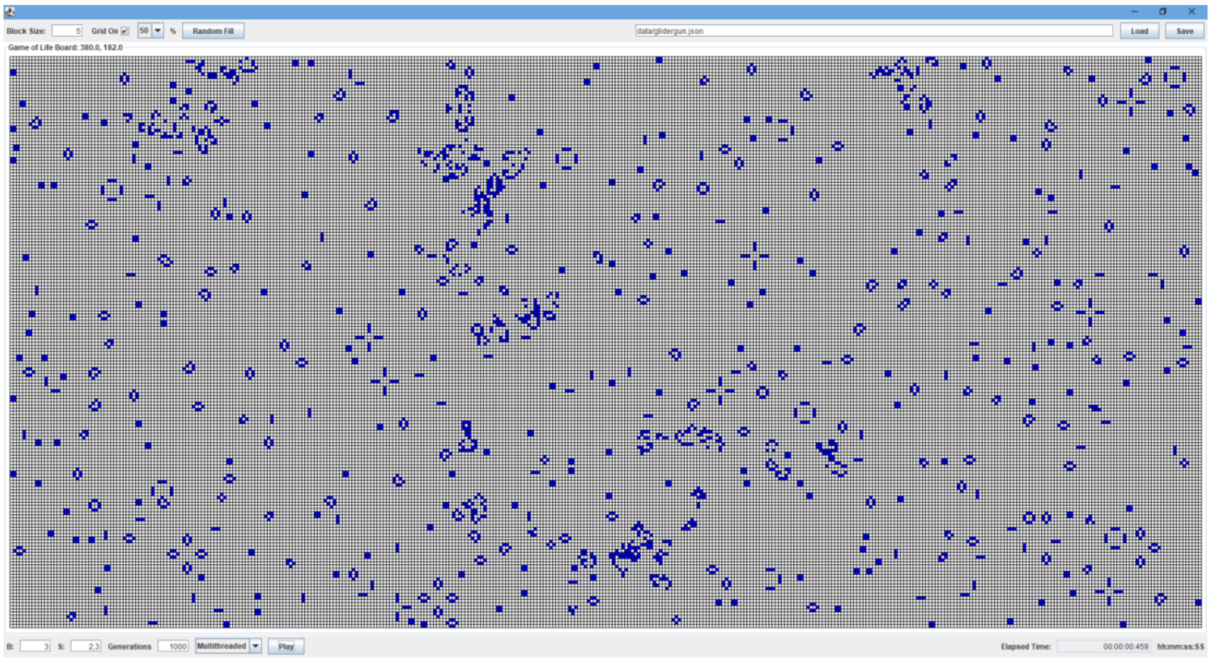


Block Size: 5 Grid On ☑ 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Single Thread Play Elapsed Time: 00:00:00:466 hh:mm:ss:SS

2 threads

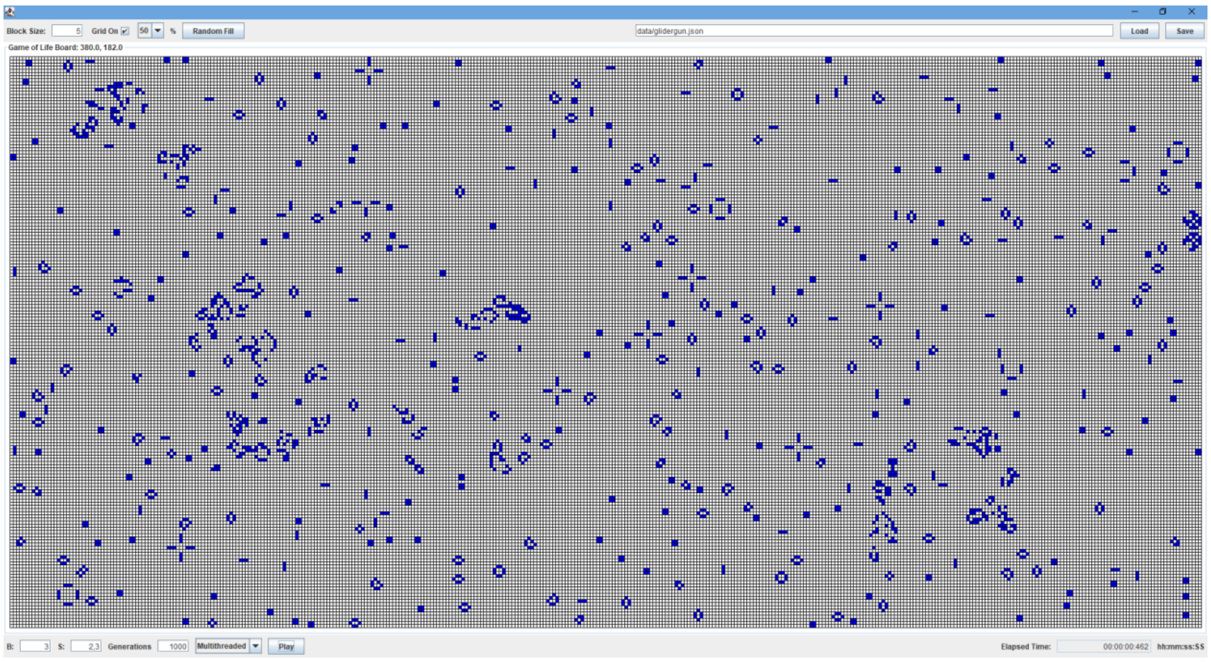


Block Size: 5 Grid On ✓ 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Multithreaded Play Elapsed Time: 00:00:00:459 hh:mm:ss:SS

4 threads

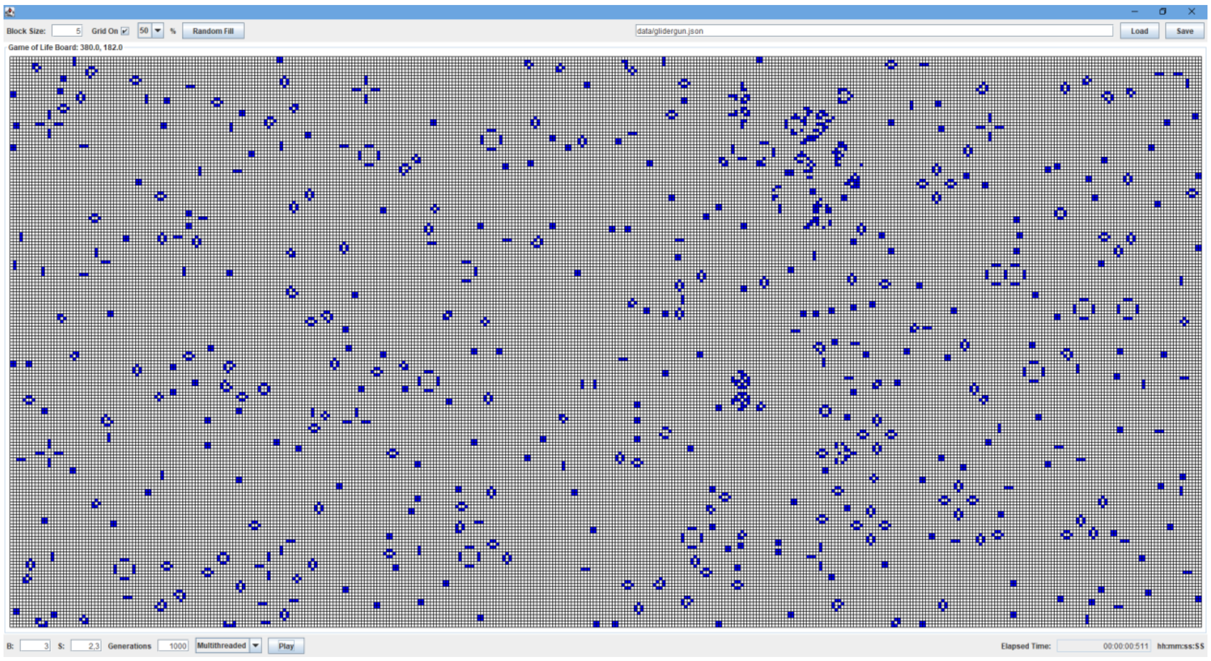


Block Size: 5 Grid On ✓ 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Multithreaded Play Elapsed Time: 00:00:00:462 hh:mm:ss:SS

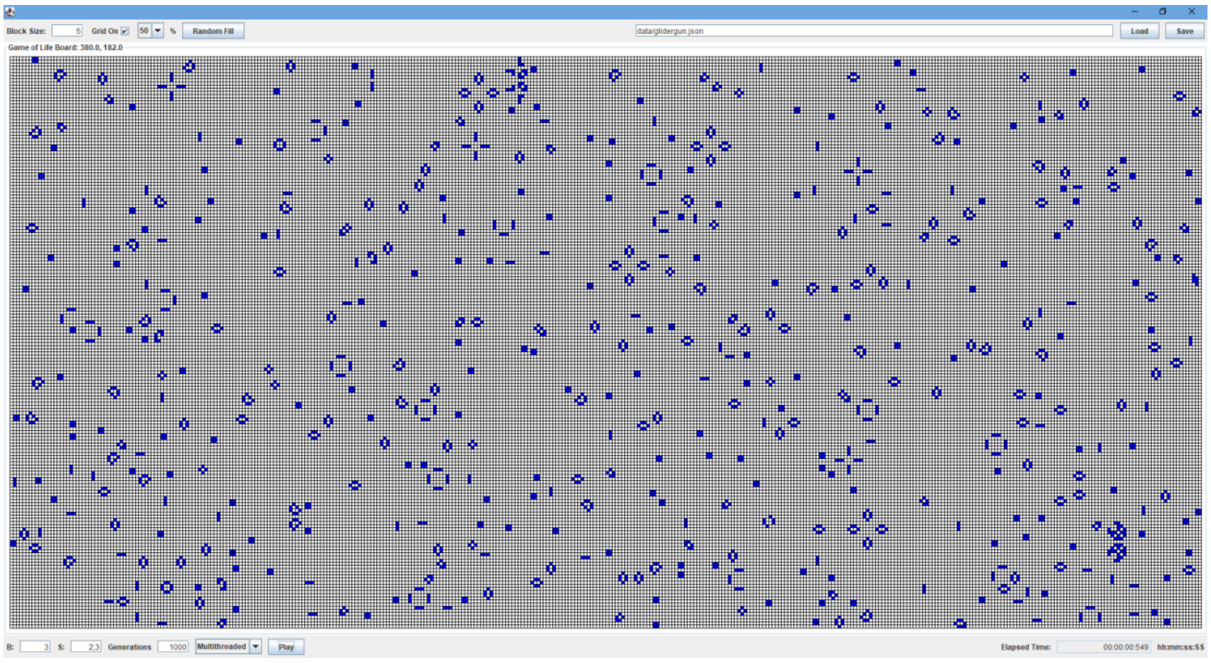
8 threads


Block Size: 5 Grid On [✓] 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Multithreaded Play Elapsed Time: 00:00:00:511 hh:mm:ss:SS

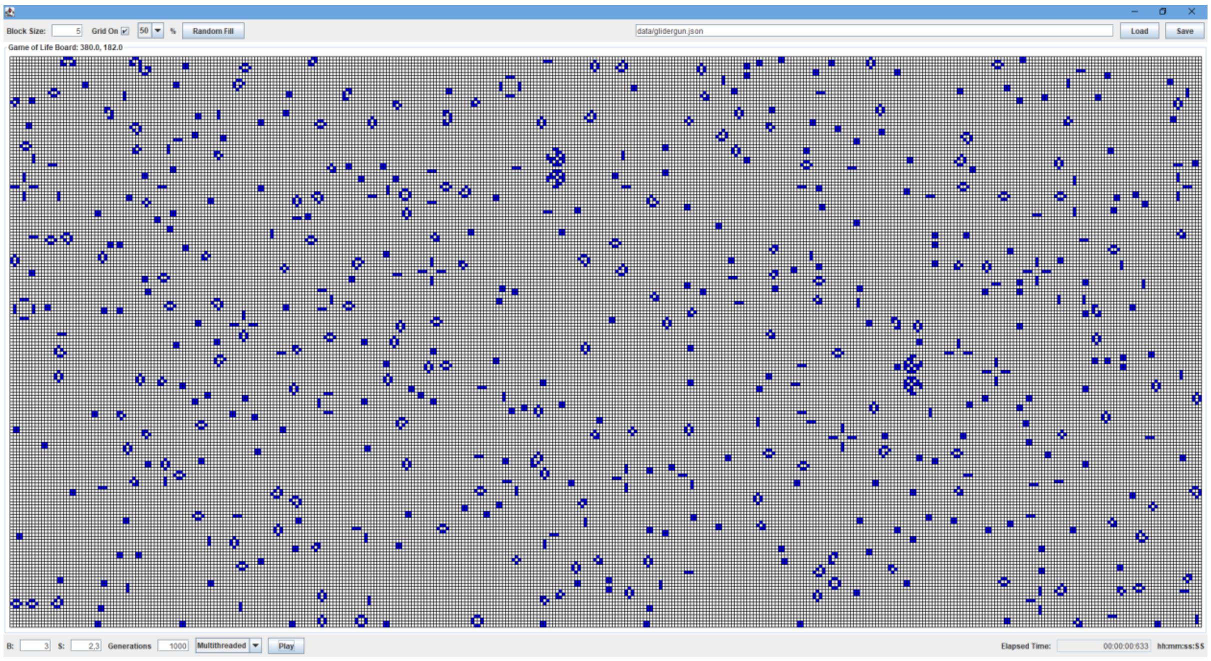
16 threads


Block Size: 5 Grid On [✓] 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Multithreaded Play Elapsed Time: 00:00:00:549 hh:mm:ss:SS

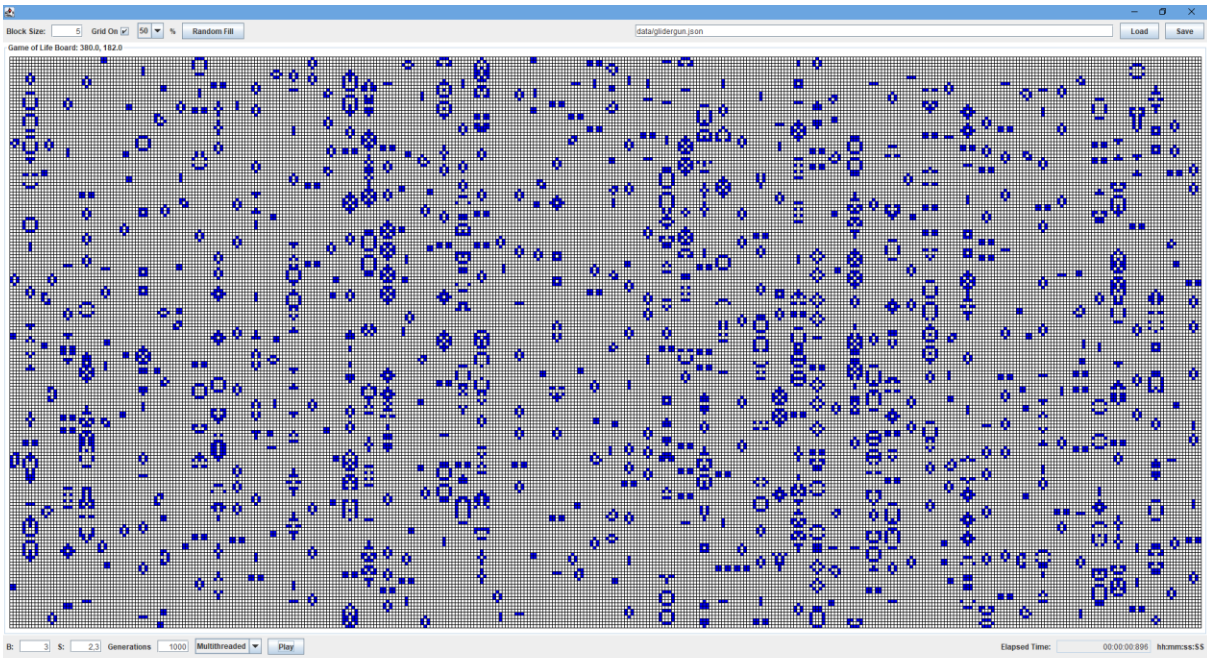
32 threads


Block Size: 5 Grid On ✓ 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Multithreaded Play Elapsed Time: 00:00:00:633 hh:mm:ss:SS

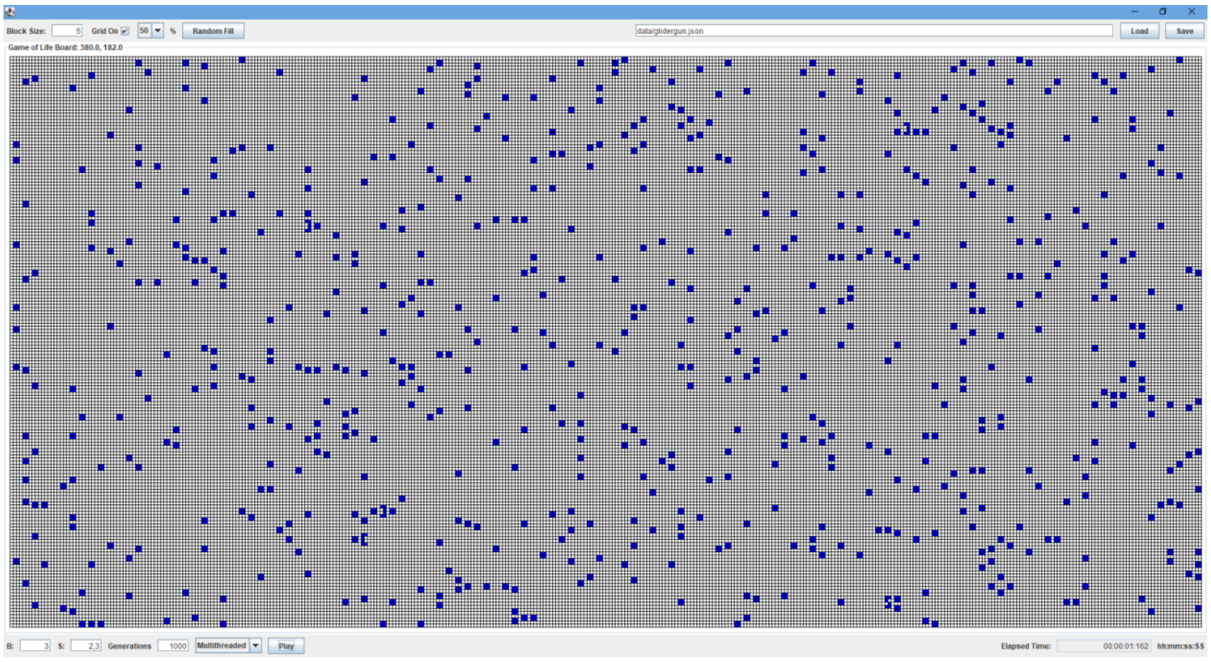
64 threads


Block Size: 5 Grid On ✓ 50 % Random Fill data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

B: 3 S: 2,3 Generations 1000 Multithreaded Play Elapsed Time: 00:00:00:896 hh:mm:ss:SS

128 threads


Block Size: 5 Grid On [✓] 50 % Random Fill

data/glidergun.json Load Save

Game of Life Board: 380.0, 182.0

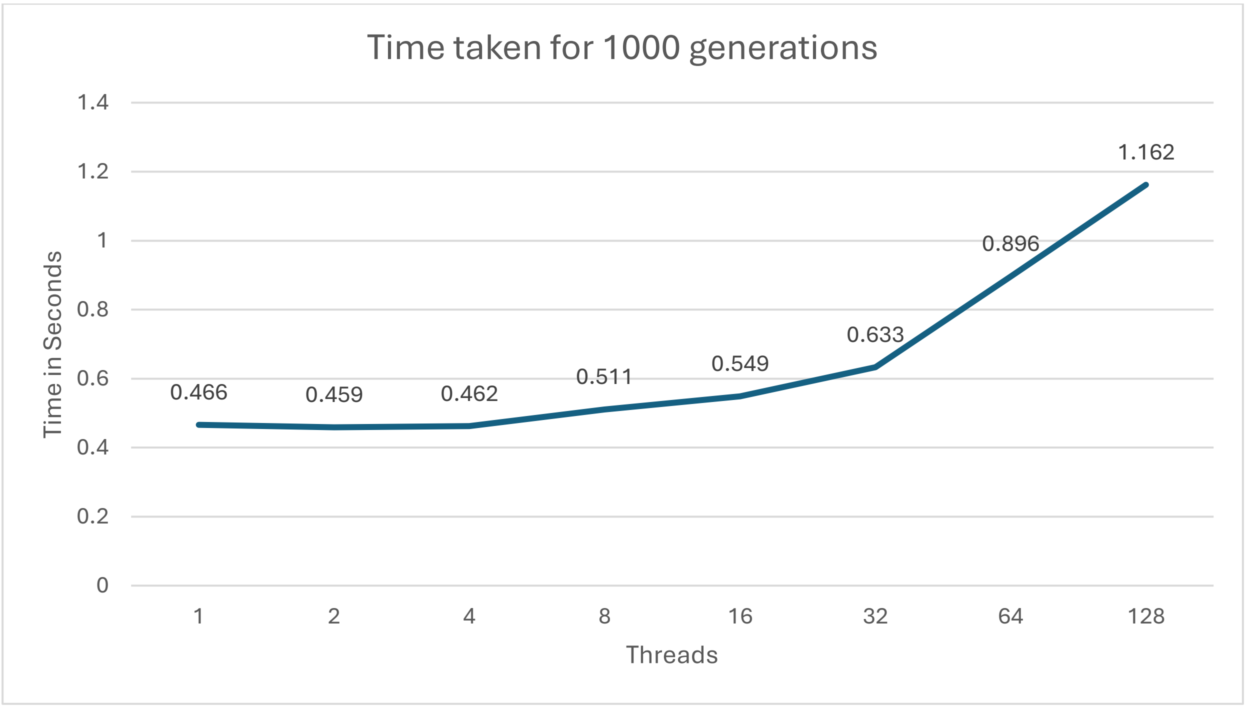
B: 3 S: 2,3 Generations 1000 Multithreaded Play

Elapsed Time: 00:00:01:162 hh:mm:ss:SS

Objective 3: Benchmark and Discussion

Threads	Speed
1	0.466
2	0.459
4	0.462
8	0.511
16	0.549
32	0.633
64	0.896
128	1.162

Above is a table that compares the time taken for each multi-threading setting in seconds. As can be seen 2 threads is the fastest setting but it is only faster than non-multithreaded by 7 milliseconds which is not a significant improvement. Below is a line graph showing the time taken for each run setting of the program.



As can be seen at the 8 threads mark the program starts to run slower and this trend continues up till 128 threads. At that point the overhead cost of creating threads overtakes the time advantage of using multiple threads. At the higher numbers, the program starts to break down somewhat. As can be seen in the screenshot the game starts to ignore some of the rules of the game, ending up in these strange patterns on the screen that will continue to repeat itself through generations. At 128 threads the program breaks even further by only showing this strange box pattern that stays, no matter how many generations go by. This is most likely due to the program not being made properly thread safe. Another flaw in the program is shown when I try to run the glider gun. The glider gun works and can fire the gliders correctly, but the gliders do not go past the first thread section of the game board. This error is caused by the way I split up the graph into its sections, but all my attempts to fix it have resulted in out of bounds errors so the program will be submitted as it. This benchmark is limited by the random fill function of the program. I was using 50% fill to randomise the game board, but this means that there are slight variations in how the game board is populated. These small differences could account for milliseconds worth of time processing. A way around this would be to take the average running time over a high number of runs to get a more accurate reading.