**Modeling the zombie invasion**

In my final project I was modeling a hypothetical zombie invasion. I started by designing the core behaviors on paper. Humans and zombies can look within 5 squares of their current location, they can move any of the 8 available directions from their start point. Zombies will look around themselves for the closest human and will charge in their direction, likewise humans will look around for the closest zombie and run in the opposite direction. Also zombies are fast so they will make three moves in the amount of time a human makes one. When a human is within a radius of one of a zombie they are then “tagged” and become a zombie themselves.

The first thing I needed to do when coding this was design a struct for what a player is. I created a player struct that contained the x position, y position, and if the player was a zombie. Then I dynamically allocated an array of these structs so they could be used later for comparisons, I also needed to create a psudo 2D array of ints that I could use to represent the world. Main really was based off of two functions, setup world and run simulation, setup world takes the players and the world and randomly distributes the players across the world and afterwards chooses one to be a zombie. Then runsimulaiton takes the populated world the array of players and the number of steps that we should run the simulation for and calls the move functions for humans and zombies the number of times we want the simulation to run, checking that for humans to be tagged after every move. In order to represent the zombies moving 3 times for each human move I just call the zombie move 3 times per loop while the human move only gets called once.

Currently there are a few flaws in the way I have designed some functions that I want to go into now. One big one is that my movement functions preform the search for the closest human or zombie, but the search isn’t necessarily accurate. The search only looks in the 4 diagonals and cardinal directions from the initial location. However this means that anything within a radius of 5, but not in a straight line on any of those paths will be missed. Another implementation I would like to improve on is my random scattering of players at the start of the simulation. Currently I generate 2 random ints between 0 and players (the number of players), however because it is just random numbers it is possible to generate the same set twice at some point. A potential improvement to this would be to find a way to randomly select numbers from a list of remaining avalible numbers.

In order to parallelize this I elected to use OpenMP. I originally was planning to use MPI to split up the world into sections and send and receive players whenever they changed local worlds, however after thinking about it I decided that OpenMP made more sense. I used openMP to parallelize the for loops in the two movement functions in zombies.c. Initially I was going to put the pragma calls in the run simulation call, but this would have been a mistake as the entire section would have needed to be a critical section since the result of parallelizing that for loop would have resulted in clobbered results. By placing the parallel calls in the movement functions I should be able to cut down the run time significantly, as they are the longest calls.

In this project I managed to take several input variables, including custom world size, custom player numbers, custom run time, and custom thread number. For most of these I didn’t need to do any kind of sanity check, but for the custom number of players I needed to be careful since I could end up with a custom value for number of players that exceeds the avalible space in the world. In order to make sure this doesn’t happen I check if the number of players exceeds the avalible space in the world, and if it does I lower the player number to the maximum space in the world.

Unfortunately I ran out of time to fully test the code, so although it compiles it instantly segfaults on running even the serial version. So obviously the first task would be to fix this, I expect it is because I never to a check in the movement function that prevents a player from leaving the world. If I had time to test it I would make sure that any attempted movement didn’t leave the world in advance of actually preforming it. Once this improvement was made I would like to parallelize different sections of the code outside of the movement functions. One function that would benefit from parallelization would be the PlacePlayers function. In this section I could use another pragma omp for call to split the for loop up so each thread could be placing a subsection of the players. This would take a bit more thought since there could be conflicts where a space ends up with 2 players in it at once, so at the very least the setting coordinates needs to be a critical section, but even then that doesn’t fix the full problem since if it passes the while condition in the code I will already have a conflict. Chances are attempting to make this code parallel would require me to rethink how I have built the function.

Another improvement I could make to the code would be combining my two movement functions, they take the same inputs and are almost entirely identical, so rewriting the code to combine those two into a single function would improve the memory usage of the program, and most likely improve the runtime. It would also mean I would have less code to worry about adjusting when checking for the walls of the world.

While I am talking about changes to move, in my initial design I was thinking about updating movement to remove the limit of looking in a 5 square radius, and instead give both sides to look at the spread of players across the entire world. This would mean I could make the behaviors of the humans and zombies more interesting since they could look at patterns across the entire world. But this was much too complex for the scope of this project. A final problem that I had with this code, and an improvement I would like to make would be improving random, currently it doesn’t always pick values in the right range, and this results in things like no zombie being chosen. It is a big bug that I have no idea how to deal with.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10x10 10 players | 1 thread | 2 threads | 4 threads | 8 threads |
| Humans | .0000003 | Seg fault | .0000003 | Seg fault |
| Zombies | .0000005 | .0000001 | .0000001 | .0000001 |

Since it segfaults if something gets gets to the edge of the map it is difficult to assess this program, I wanted to use the omp\_get\_wtime function to record timing data for improvements in the movement functions, I managed to do this and have a very small amount of data for how long it takes and the improvements that openMP provide. But getting anything useful out of it is not really possible as the problems with rand prevent the actual move behavior from taking place properly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100x100  100 players | 1 thread | 2 threads | 4 threads | 8 threads |
| Humans | .0000022 | .000025 | .000022 | .000021 |
| Zombies | .0000001 | .0000002 | .0000001 | .0000001 |
| 100x100  100 players | 1 thread | 2 threads | 4 threads | 8 threads |
| Humans | .0000022 | .000025 | .000022 | .000021 |
| Zombies | .0000001 | .0000002 | .0000001 | .0000001 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1000x1000  1000 players | 1 thread | 2 threads | 4 threads | 8 threads |
| Humans | .000424 | .000327 | .000249 | .000226 |
| Zombies | .000004 | .000004 | .000004 | .000011 |

Things only really start to become apparent for the humans when we get to numbers like 1000 players, at that point we see a clear improvement in the human times. However the zombies don’t show this for a few reasons, first is that the zombies always start at 1 and I am just testing the first move. And another problem is that frequently the random number generator doesn’t even select an actual zombie so they take no time to move.

Overall I am happy with how this came out, the code isn’t complete, but when I run it I do see an output of some sort, and the parallelization actually seems to be working. I did run into a few problems that I have covered earlier in this write up, but given the complexity of the project I don’t really find that surprising. I came very close to meeting what I initially expected to accomplish at the start of this project, and that was a basic behavior for a parallel version of a zombie invasion, and I came very close to that with the exception of the segfaults from players being able to leave the world. When the world prints we can see that players are placed in the world and will move around, and if we make the world smaller we can clearly see how things move around.

**Self Grade**

I am giving a letter grade to how I think I did on each section of the project possible grades are A,B,C,D,F

Scope/Challenge – I think this was a project that was large in scope and very difficult: **A**

Completion – The project was close to initial completion, but still has oversights and bugs, segfaults a lot: **B**

Implementation – I did use OpenMP effectively, but not to the full extent it could have been used: **B**

Assessment – data collection is minimal and doesn’t give much information**,** however is fairly extensive: **B**

Wattage – Did not use a significant portion of the union college computing resources: **N/A**