Student Name Student Number

Joni Niemela H296319
Kavan Mäkelä 50106323
Andrew Gergis 150905291

What concurrency problems were identified in base code

The running variable in world file was shared between files

How they were corrected in your submission

We added a std::mutex running_mutex; for the running inside the next_generation function in the world.cpp file
for the locking we used a unique lock lock(world::running_mutex)
and the unlocking lock.unlock()
both are inside the the if condition checking the status of the running variable.

Explain your overall strategy to use multiple threads in world generation and how this strategy was implemented.

The overall strategy for using multiple threads in world generation is to divide the world into smaller chunks and process each chunk concurrently in separate threads. This allows potentially speeding up the overall world generation time.

The implementation involves the following steps:

1- Determine the number of rows and columns in the world, and create a barrier to synchronize the threads after each generation.
2- Create an array of threads to process the world in parallel. The number of threads is determined by the constant THREADS
3- Calculate the number of rows to process in each thread by dividing the total number of rows by the number of threads. This ensures that each thread has an equal workload.
4- Partition the data processing into multiple parts and run them in separate threads. Each thread processes a subset of rows, from start_row to end_row, where start_row and end_row are calculated based on the thread index and the number of rows per thread

5- Within each thread, iterate over the columns for the specified range of rows. Perform the necessary data processing for each cell in the world, including checking the neighbors and updating the next generation of cells based on the rules of Conway's Game of Life.
6- Use a mutex to lock the world::running flag before checking its value to ensure thread safety. If the simulation is not running, return early to avoid unnecessary processing.
7- Wait at the barrier after processing each generation to synchronize the threads and ensure that all threads have finished processing before proceeding to the next generation.
8- Join the threads after they have finished processing to wait for them to complete.
9- Finally, swap the current and next generations of cells, and output the time taken for generating the next world.