Brendan Rice, Jake Murphy
CSC 412
Final Project Report

Design

A huge part of our design process was using Github / Git in order for us both to complete different tasks. With that came a handful of merge conflicts, which were always resolved rather easily.

(V1) Looking at version one, we created a singular thread that called our base function moveTraveler. moveTraveler is essentially the host function for each thread, which moves the traveler until we find the exit or terminate the program. We use a function getDirection which is used to get a random direction, which uses a bounds checking function to find open spaces around the traveler. If no directions are returned, the loop breaks thus indicating that there is nowhere to go. If the direction is valid, we call a function updateSegment. This function checks if we should add a new segment to the traveler, while also updating the grid's spaces to match the changes we make. Finally we check to see if the traveler has found the exit, and if so we free the traveler data and terminate the thread.

(V2) We added in a vector of threads, in order to run multiple travelers at a time. The logic for finding the exit / moving the traveler is identical to that of V1, however we simplify some lines in order to take up less space / improve efficiency. We also made sure to join the threads during the termination of the program. We used C++ threads because we both were more familiar with them and believed them to be easier to use.

(V3) In this version of the program we needed to add in a mutex lock in order to prevent travelers from accessing shared resources at the same time. We added our lock within the moveTraveler function, and locked the grid before grabbing our newDirections vector. We then unlock the grid after we are done performing operations to it.

(V4) This version of the program involves travelers moving the partitions around the board. This is where we see the largest influx of functions, and operations to perform as we are changing the grid far more than in the previous versions. We call the function checkIfSpaceIsPartition, which is used to check if we're heading in the direction of a partition. This function is rather large, and has a series of conditionals to check the direction in which we're heading. We then check and see if the space we're moving to is a vertical or horizontal partition. If not, we exit the function and update our segment like usual. But if we do find a partition, the findPartitionIndex function will grab the index of the partition we are approaching. This way, we can make changes to the block list, as well as the positions on the grid. Next, we call one of the movePartition functions (eg.

moveHortizontalPartitionToTheNorth). Say the partition is a vertical partition to the North of the traveler. The function called is movePartitionNorth. After making the change to the partition, the traveler will either update, or find another direction as it still might not be able to move after the partition has moved. The logic we chose for the movement of the partitions is rather simple. If the partition is the same direction that the traveler is trying to go (a traveler moving east with a horizontal partition in its way), we move the partition once and move on. However, if the partition is vertical rather than horizontal, we move it up or down until there is a free space. If there is not free after moving it both upwards and downwards, it tries to find a new direction.

(V5) This version was a breath of fresh air compared to the other ones. Instead of using one lock and locking before each movement made by the traveler, we created a 2D vector of locks with the same dimensions as the grid. Then, we lock those grid spaces before gathering the new possible directions for the traveler, and unlock them after moving it.

(EC1) We implemented the progressive disappearance extra credit 1 in version 2. Since we used vectors for our travelers, this consisted of a few simple changes in our finishAndTerminate function such as sleeping for a split second in between removing each part of the traveler.

Limitations

Our programs had no limitations in regards to the project tasks, however Jake uses MacOS while Brendan uses WSL. This was slightly troublesome, as there were some rare cases where the program would throw one of us a warning, but would work just fine for the other user.

Difficulties encountered

There were 2 large difficulties to face, or more specifically tasks that took significant portions of time to complete. A smaller issue was that both of us made edits on Version2 of the program that were supposed to be made for Version4, leading us to spend some time reverting back to previous commits while also trying to salvage the functionality that we had implemented in later ones.

The first big difficulty occurred in Version1, which was the process of moving a traveler. Although it was simple looking back on it, we really had trouble with these three subtasks within the problem: changing the grid, adding a new segment, and the moving of the traveler itself. Initially, we didn't pick up on the fact that we had to change the grid space at a position from free → traveler, and vice versa. When trying to add a new segment, we struggled with figuring out when we should make that check, and how that process should work as far as changing the grid as well as adding the traveler segment. Finally, when trying to move the traveler itself we had an issue with how the segments were moving, as it seemed as though a traveler and its segment

would move “rigidly.” What we mean by this is that the traveler wouldn’t move like a snake, it would move around the board in the same shape.

The second big difficulty was moving the partitions in Version4. Not only did this task take a long time to plan out, but we also found it was very ticky tacky in terms of errors. Incorrectly checking a specific space could make a partition “eat” a traveler segment, or using the wrong index within a loop could cause a segmentation fault. These errors occurred rather frequently, and were debugged through Xcode, the whiteboard, and hundreds of program runs. These small errors, as well as the need for a lot of conditionals within the program caused a lot of issues for us, but were resolved in the end.