Viet-Hung Le, Skyler Van Horn

V1: For Version 1 of the program we created a program that just takes in one thread or traveler that gets to the exit. We made this easier on ourselves by first hard coding the traveler to be 1 since we are only working with one traveler. We then created a function in the struct of the dataType header file that is called move and take Direction dir and bool to growTail. The Direction dir is something that takes in the direction from the dataTypes which had EAST, WEST, SOUTH, and NORTH. Within this function are conditions that check the direction and add 1 whichever direction is true. There is a for loop that takes the remaining segment and linked them together to follow the head. So whenever the head moves in a direction, all the segment follows it.

The neighbor_checks function is a function that takes in a row and col and the type from the dataType header file which checks if it's near the exit. This is what this function is mainly used for, to check if it's near the exit.

The get_free_space function is a vector of direction which check the square ahead of where the traveler is moving if it is a free square or not. If it is a free square then it can move in that direction. If it isn't then it will pick the next possible direction to move. This also has a case so that it will never move in the opposite direction of where it is heading. So if it goes NORTH, it can't go SOUTH and same for EAST and WEST.

The thread_run function is where everything happens. We have two while loops in here. The first while loop we hard code to be true so that it keeps running. In this while loop, it takes a condition where if it has directions then it randomly chooses from the dirList. If it is true then we call the move function and it keeps calling that until it reaches the exit. We have a condition where we call the neighbor_check to see if it is an exit. Once we do that then we break and check the traveler gray.

V2: For Version 2 we created a for loop in the main function that takes for how many travelers we have. We create threads equal to the number of travelers and continue to do so as long as the boolean is true.

V3: For Version 3 we just created a mutex lock on line 62 call mylock. We started locking at line 538 and release it at the end of the condition at line 552 and 564. We do this so that the travelers will never intersect each other and of their own tail.

V4: For Version 4 the program needed to have the travelers be able to move sliding partitions if they are facing them and hit them.

First I created a function called neighborInfront that would be called in the thread_run function. This function takes in the head of a traveler, the direction it is facing, and the square type you want to check that it is "looking at". I used this to determine if a partition should be moved.

In the thread_run function I check if the neighbor in front of it is a partition. If it is, then I need to get the grid position of that location. I used a neighborFind function I created.

The neighborFind function will take in the row and col of whatever position is passed to it and a square type and check what grid position is at that location and then return it to the call.

Once I had the grid position I used a two-dimensional for loop to check what partition had this position inside it. If I found what partition that position belonged to, I would attempt to move that partition by calling the move function I created in the SlidingPartition struct.

This function first calculates how many spaces the partition will need to move in order to make way for the partition. Then it checks if the partition is vertical or horizontal. I prioritize moving up and left depending on if the partition is vertical or horizontal. From that point, it will check available map spaces that it can move. If there are enough map spaces, the function will then check if those spaces on the map are FREE_SQUARE type. If they are, I then move the sliding partition similarly to how a traveler is moved. I increment/decrement the column/row (depending on the orientation and direction of the movement) and then update the grid accordingly.

This sliding partition move function returns true if it has been moved. Then in the thread run function, the Traveler will keep moving in the same direction as its previous move.

Version 5: For Version 5 we needed to have as many mutex locks as squares on the grid. This is so that whenever we needed to use certain grid spaces we could lock them from being used by another thread. To do this we first used a nested for loop to created the mutex locks. Then whenever a partition needs to check available spaces to move and then move to those spaces, I would use a for loop to lock those grid spaces. I did the same thing for the traveler. If the traveler is moving to a new square, I acquire the lock for that grid square. Then once the move is done I release the lock.

Problems:

We had a problem where the Travelers would never leave the grid if hitting an exit. We solved this by deleting the vector AND making all of the spaces it occupied FREE_SQUARE.

In Version 4 there were a lot of problems getting the partitions to move correctly as well as segmentation faults. This was fix with help from professor Herve. The issue was that it didn't like how we subtract the partitionSize by the blockIndex without stating the type it was. Once we state that it was an int that's when the segfault disappear. Another issue in Version4 is that the traveler keeps eating the partitions or sometimes move the partition correctly. It does this but mainly you will see the travelers eating the partition. We did not know what the problem was so we could not fix it sadly enough. We searched through the partition_move function. I mainly think it might be the blockIndex or perhaps it was in the partition_move function but we just never saw it.

We also had an issue where it would never reach the debugger we created that print out a statement Dying 2. It wasn't because it was not reaching it but that because we had a usleep that we needed to wait for it to reach the print function of Dying 2.

Things that we could've done was to have the versions take arguments rather than hard coding it. We didn't do that because it was optional. Professor Herve said it was fine to hard code it so we didn't do that. Another thing we added in was a debugger to move east when space is press on the keyboard. This was created in version 1 to make sure that it moves correctly. We figured out by adding in the case for w-a-s-d that we could take control of the traveler. So in version 1, we did the extra credit of taking control of the traveler.

Another thing we had with the help of Chris was the transparent square over our traveler. Thanks to this were we able to tell if our traveler terminated and deleted from the GUI. If we didn't have this then we wouldn't know that there were invisible segments left by the traveler after it hit the exits. The code that helped us was coded in the gl_frontEnd.cpp that we were supposed to touch but was very useful in the end thanks to Chris.

Version 5 has some deadlock issues. For some reason sometimes the partitions attempt to lock and then move and then when they try to move again the traveler (or thread) is stuck and will not move anymore.