Jarrod Blanchette
CSC 412 Spring 2023
PROG 5 Report

For version 1 I modified main, threadfunc, and created a oneThreadGeneration. The oneGeneration is used to create the threads and assign its values and later on join the threads before swapping grids and incrementing the generation counter. OneThread is what is used to give the parameters for the threads i.e. start row and end row. The main is just where we assign the arguments and then allocate the thread array called info.

For version 2, I got rid of oneGeneration completely since we are dealing with persistent threads this time. OneThreadGeneration remains the same. In thread func i first started using booleans to keep the threads going and later on layed locks on top of that to test it working, and it works as intended according to the output. (NOTE. I will include a readme.txt showing the extra credit that I did, but I will also mention here that I added the EC for locks as well as implemented 2 new rules for the other extra credit in version 2 also.). I did find with the locks that It was in fact in a fully locked state according to the output. So through trial and error I added another unlock at the bottom of threadFu and that ended up fixing it.

For version 3, I just created a randRow and randCol to assign those values randomly and edited the code from oneThreadGeneration to work for these new random coordinates. After that I created a lockArray and set the girls to the random coordinates. One thing that was interesting after a lot of troubleshooting was that originally version 3 was running, but required so much from my VM that after a short period of time, the program was killed. I realized that all of the code used to create the lockArray would be repeated every time that oneThreadGeneration was called so I created a function called initializeLockArray holding that code and then just calling that function in the main before initializing the front end and the application.