Evacuation Project

Harth Majeed – 100 896 761

Introduction

This project is about 2 robots evacuating from a circular room, 3 cases, 1) Both start in the middle, 2) One in the middle and one randomly placed, 3) Both placed randomly. I have made a GUI animation of this process using Java.

Implementation Details

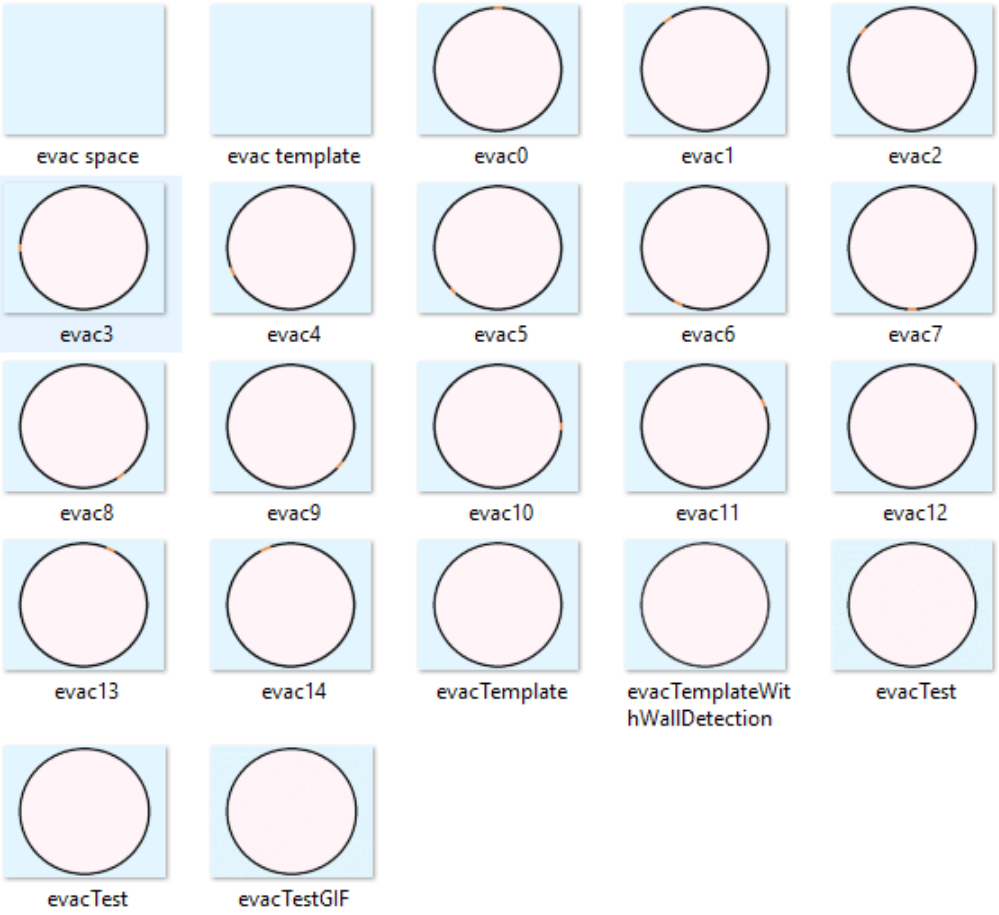
I have 3 different classes, Presentation, EvacAnimation, and Robot.

Presentation

Presentation is responsible for several tasks. Presentation initializes the window GUI, all the Jpanels, objects, and Jframe to build the rest of the window. There is also an “actionPerformed” function that works based on the Timer object created in main. This function checks to see if one of the robots found an exit and if so then it alerts the other robot of the exit location, then the simulation functions are called on the robots and the screen is then repainted to display new robot locations. This function is tied to an object created in the main, its called a Timer objects and at every specified interval of time it calls the actionPerformed method. To start the program in main the timer’s start() function is called and it runs forever until the user closes the window.

EvacAnimation

EvacAnimation is responsible for creating the animation area. It start off in the constructor by loading in 15 images, these 15 images contain the same circular image but with 15 different exit locations, each location was made manually, they are in the resources folder.

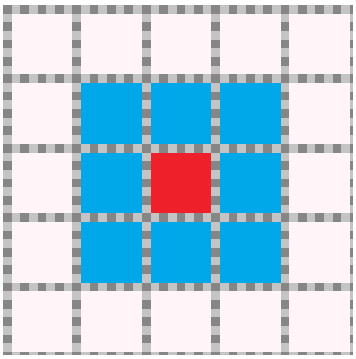

evac space
evac template
evac0
evac1
evac2
evac3
evac4
evac5
evac6
evac7
evac8
evac9
evac10
evac11
evac12
evac13
evac14
evacTemplate
evacTemplateWithWallDetection
evacTest
evacTest
evacTestGIF

Two robots are then created in the center, after that a setRandomImage() functions chooses an image and loads it into the robot as they need it. And the final function is simulateAnimation which calls the simulate function on both robots.

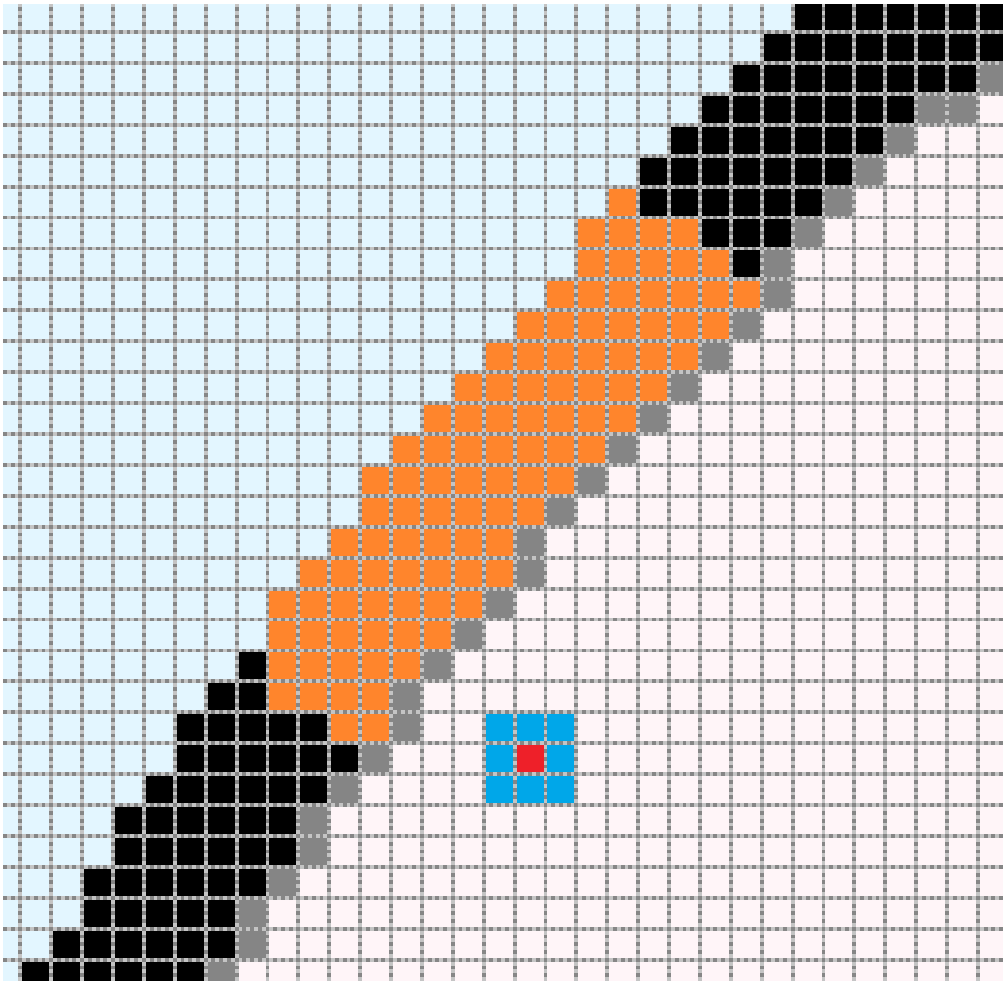
Robot

Finally, the meat of the whole project. The entire algorithm is in the Robot class, as intended this is a distributed computing system, so my approach was to make sure each robot was totally independent of each other and the only thing it can be altered of is an exit's coordinates when it's found. There is a lot of logic here so it's best to separate it into sections for better readability.

FindWall(). The first step is to find the wall, there are several steps to this process. First the function will need to select a suitable direction, this direction is chosen randomly, it can choose to go in any of the 8 directions, since we are using pixels here.



The robot then follows this direction until it detects a wall, wall detection is based on the color of the pixel at its current position, so continuous check of its surrounding is very important.



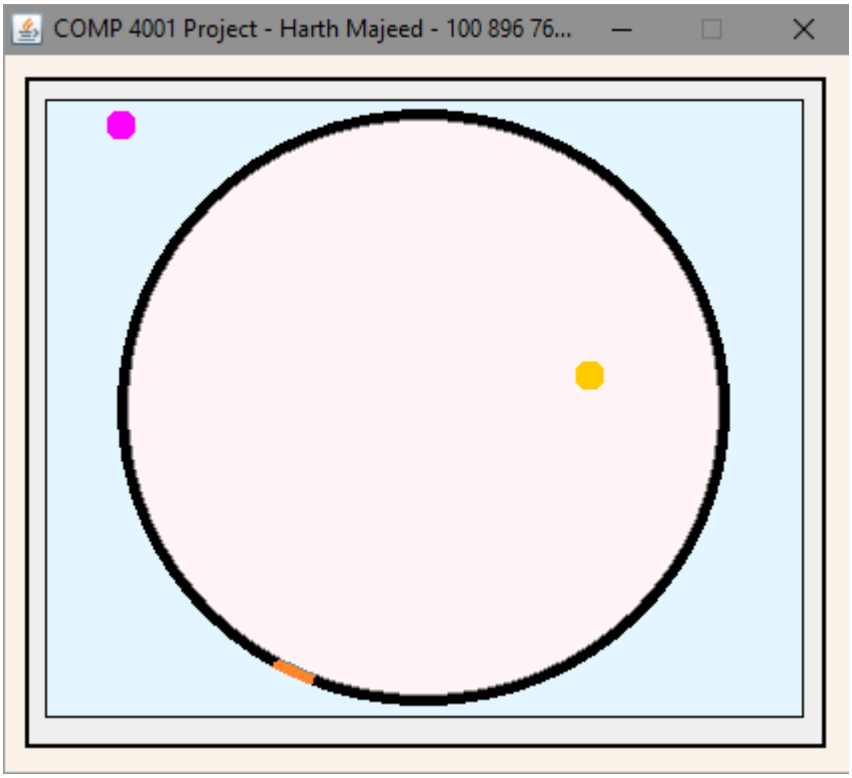
Checking within a pixel's surrounding must check for a wall, and also it must check if it might accidentally skip the wall as this was a major issue. Once the wall is found variables are set so that it won't be called again.

Walk(). Yes, walk(), the function now takes a hold of the robot's movements around the wall, this was a very tedious and tricky issue as many factors need to be taken into place. The function must check the surrounding pixels for an exit, if the exit is there then leave and the other robot will be alerted, but if the exit is not there then keep moving. This is where it gets tricky, in my implementation every last position is saved so the robot doesn't accidentally go back, so the robot must check for the squares that have the path color(grey) and walk along that path without going on the last position. Now since it must check for 2 Grey path positions it must be careful and take that into account, sadly this situation

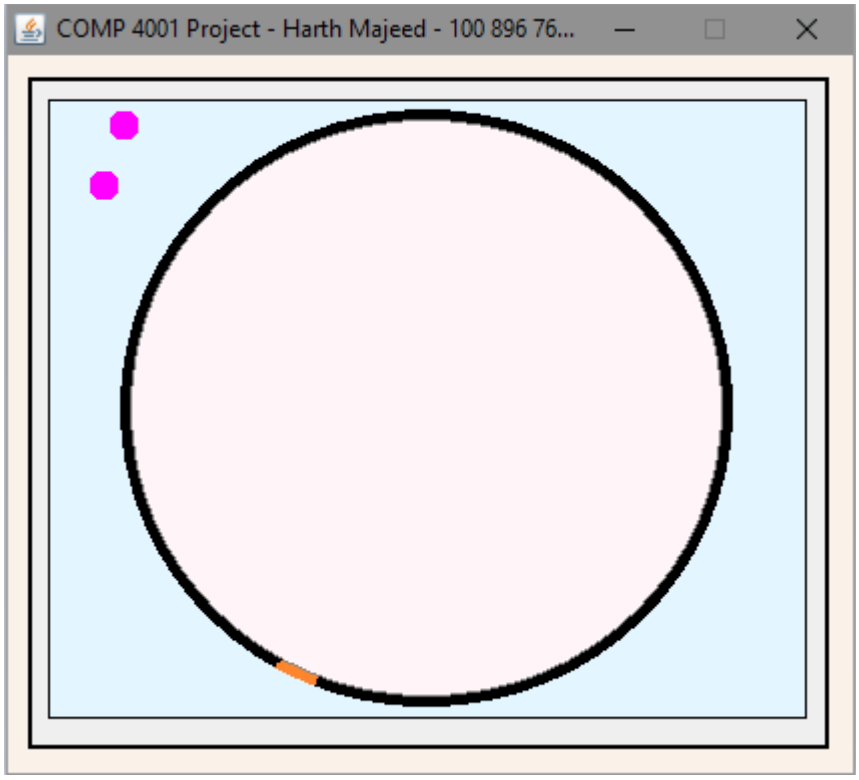
here is where it is the most buggy and is still a major issue as the robot will get “stuck” by moving back and forth from its new position back to its old one.

GotoExit(). GotoExit() function is very simple, it checks to see how its X and Y values compare to the exit, and decrements/increments accordingly until it reaches the exit. This function is only called when another robot already evacuated, and once a robot is evacuated it can never be bothered again.

Successful Runs



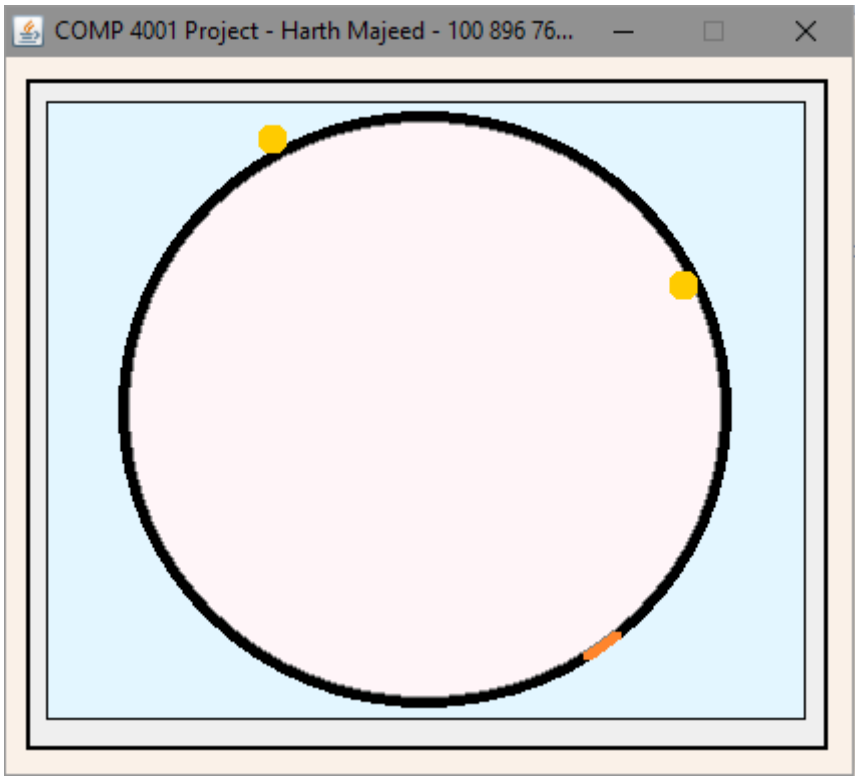
A robot cutting across to the exit.



Both Robots evacuated.

Main Issue, robot getting stuck.

An issue where both robots are getting stuck.



There is only one way “un-stuck” a robot and that is when the other one finds the exit, the robot will then go straight to the exit. The “stuck issue” is the only issue preventing me from completing the simulation successfully, I spent a few days on this issue trying to resolve it but I had no luck, I got close, by using a DirectionSetter function but that still didn’t fix the issue. I also coudn’t obtain any results since the simulation couldn’t complete half the time.

Conclusion

Overall, despite the simulation issues I really enjoyed creating this program. It made me appreciate the difficulty and time that goes into making the algorithms we learned in the lectures. Doing something that seems so simple can be very difficult especially since there is so many implicit cases that we run along implementation. My goal was to see how the robots would score with random exits and all 3 cases, it would have been very interesting to see how randomly placing the exit would affect the results. My original hypothesis was that having the exit randomly placed would actually produce faster evacuation time as the direction of he robots would also be random. This was an interesting learning experience as a lesson in distributed computing and making an animation out of it was a well deserved learning experience as you experience the difficulty of dealing with pixels.