Reflex Agent

Chase Hiatt

For this project, we were tasked with programming a virtual robot to maneuver around obstacles and reach a goal. As a windows user, getting this lab to work properly was quite a challenge. After hearing about the issues running it on windows, I first tried running it on WSL, but ran into a whole host of issues with the U.I. drawing correctly. After that, I decided to just switch to raw windows and hope it was working better. After getting gcc and g++ installed on my windows (which was quite a process) I ran into issues with my java version. I eventually figured out that if I ran javac –release 8 …. then compilation and run would work great. After I was able to get all programs compiled, I ran into issues with the U.I. not updating in worlds outside of world 1. When I would start the program, world 1 would launch and updates to agentCpp.cpp would be reflected after I pressed 'u'. However, switching worlds and pressing 'u' would not result in any updates to the potential fields. Even my most basic function implementations would refuse to update in all worlds outside of 1.

I eventually gave up on having nice potential fields outside of world 1 and just focused on having a good program. I decided to combine 3 primary potential fields for my robot. Firstly, I have an ambient potential field which points in the direction of the goal. This affects all squares equally, but not with particular strength. My next field is my tangential field. This field is strongest between 5-10 distance units from an object. This field serves to direct the robot around objects. My final field is my perpendicular field. This field affects the robot at distances <5 units. This field is quite strong and works to push the robot away from walls so that it doesn't hit/get stuck.

The ambient field is simply accomplished by finding the x and y offset from the goal, then initializing the result vector with those values. I normalize the result vector prior to calculating the next fields so that this field is weak and can be overwritten. For my tangential field, I iterate through each distance sensor and see if the distance is less than 10 for that sensor. If it is, I calculate an angle 90 degrees greater than the sensor's angle and add some force in that direction. The amount of force applied is inversely proportional to the distance (i.e. as distance decreases, the force increases). My perpendicular field is almost identical to my tangential field, except with a smaller threshold, more strength, and a 180 degree offset (to push perpendicular instead of tangential from the object).

Prior to finding this combo, I tried only using a 'goal' field and using a 'goal' field and a 'tangential field'. I found that without all 3 fields, the arrows would sometimes point directly into the wall of the obstacle, which is not a productive potential field. After tinkering around with thresholds, constants, and strength formulas for each of the 3 fields, I believe I found a combination that is fairly effective and efficient.

I wish my computer would have let me visualize my algorithm in all the available worlds, as I believe it would have performed well even on world 4. After a few hours of fiddling, this screenshot is the best I have to show for my efforts. My robot would not run, so I'm not sure how long it would take to reach the goal.