CAP4053 Assignment 1
Assigned: February 2, 2015
Due: February 16, 2015

1. In Unity, create a world with a two-dimensional map. In a two-dimensional map, the coordinates of agents in the world are specified with two dimensions (x,y). However, note that Unity is three-dimensional.

2. Create at least three agents: The *subject* (i.e. the individual who will do the sensing) and at least two other agents. The additional agents can be static (i.e. they do not need to move) or they can move according to simple rules that you create. *However, it must be possible to place static agents at desired locations during testing*.

3. Each agent (including the subject) *must* always have a **position** (x,y) and a **heading** (theta).

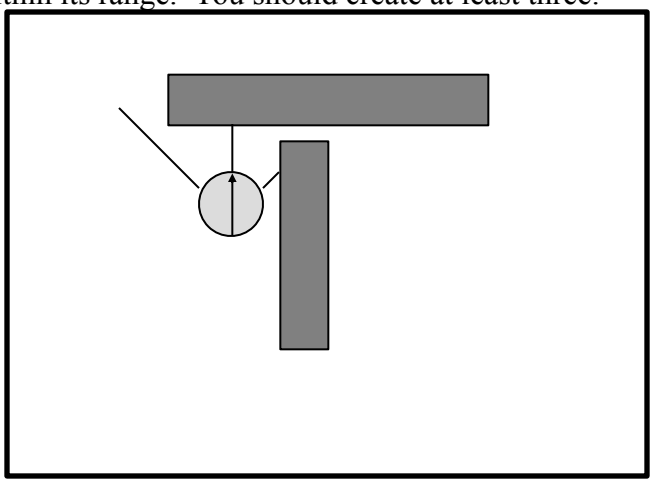
4. Create at least two walls. The walls should all be significantly longer than any single agent's width.

5. Implement keyboard controls for the subject agent. Note that these controls are for debugging and not for entertainment, so they are not designed to be fun. The controls must be very specific:

    a) Rotate-left and rotate-right change the agent's heading but do not change its position.
    b) Move-forward causes the agent to move in the direction of its heading.
    c) Move-backwards causes the agent to move in the *opposite* direction from its heading
    d) You may control speed in any way you choose: Continuing pressure may cause motion or rotation to accelerate. Or, speed may be constant. The only rule is that there must be a way to *stop* the agent from rotating and from moving, and it should be easy to do so.

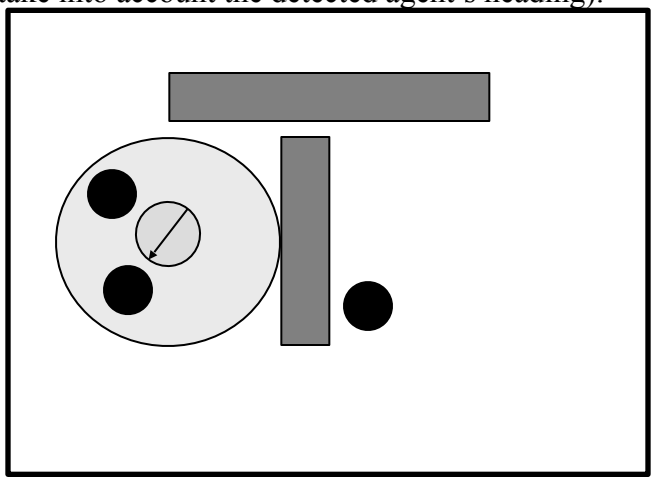
6. Implement three types of sensors:

   a) **Wall sensors (i.e. rangefinders or "feelers")**. These project a ray *relative* to the heading of the robot, from the location of the robot outward to a maximum length (the maximum length is the sensor's *range*). Each wall sensor returns the distance before it intersects with the nearest wall, if one is within its range. You should create at least three.



Wall Sensors (Rangefinders)
Return: distance

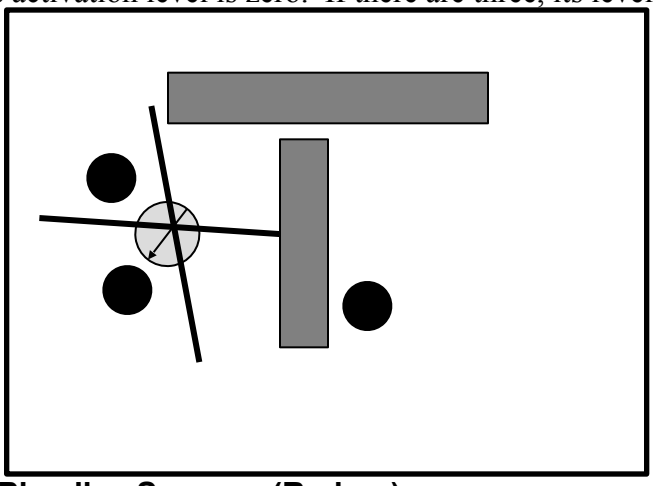
   b) **Adjacent agent sensor.** Returns a list all agents within a fixed radius (the sensor's *range*) of the agent. Each entry in the list should identify the agent, and specify its distance from the subject and its *relative* heading, i.e. the angle from the subject's heading to the detected agent (you do not need to take into account the detected agent's heading).



Adjacent Agent Sensors
Return: relative angles, distances

   c) **Pie-slice sensors (i.e. radars)**. Divide the world around the subject into four pie slices in space. Each pie slice is represented by two angles that

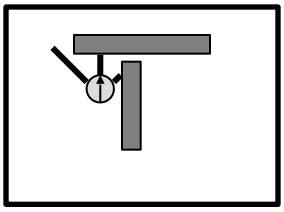
are relative to the subject's heading. These angles are its *boundaries*. When the subject rotates, the boundaries of each slice rotate with it. Each pie slice can see ahead a maximum *range*. For each pie slice, return an *activation level* proportional to the number of agents within the slice (i.e. bounded by its boundaries and its range). For example, if there are no agents, its activation level is zero. If there are three, its level is high.



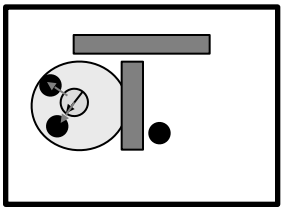
Pie-slice Sensors (Radars)
Return: activation levels

Implement debugging displays, which should output the following information updated in *real time*, as the world is displayed at each tick. Note that (a) through (c) below may be displayed as graphical representations or as text. Graphical representations are better and easier to understand quickly, but text may be easier to implement depending on the engine.

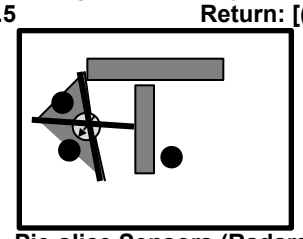
a) Display the position and heading of the subject.
b) Show the output of each wall sensor.
c) Show the set of adjacent agents, including distance and heading.
d) Show the activation level of each pie-slice sensor.



Wall "Feelers" (Rangefinders)
Return: 5,2,0.5



Adjacent Agent Sensors
Return: [(0,1),(270,1)]



Pie-slice Sensors (Radars)
Return: 1,0,0,1

Example Outputs: Note that graphical output does not need to be superimposed right on top of the agent. It can be shown on the side instead.

Turn in a completed report all in *hardcopy* except for code (which you will turn in online):

e) Paragraph about chosen game engine (see #1 above)

f) Turn into the class webcourse at http://webcourses.ucf.edu *code that you wrote* (you do not need to turn in existing engine code).

g) Include in the hardcopy report a *summary* of your code (not just code comments) that explains how it works and any special properties or features.

h) Include screenshots with debugging outputs on-screen:
   i. Subject in at least 3 different positions, in each case with a different heading. Debugging output should clearly display the position and heading of the subject.
   ii. Subject facing agents in various relative locations. You must at least show separate tests where the subject faces north, south, east, and west. In at least one example, there should be more than one agent visible to the subject. Adjacent agent sensors and pie-slice sensors should indicate correct values. If it is not clear how they are displayed, include an explanation.
   iii. Subject facing a nearby wall to the north, east, south, and west. Wall sensors should indicate correct values. If it is not clear how they are displayed, include an explanation.

i) Conclude with a short summary of lessons learned. What was harder than expected? What did you learn about the engine that you did not realize at first?

7. Make an appointment with the TA during the week of February 16th to show him your completed demo working. He will check your debug output against the agents on the screen to make sure everything is correct.