This lab is worth 3% of your final AI4G grade.
Due Friday March 24th 17:00

Understanding Reinforcement Learning

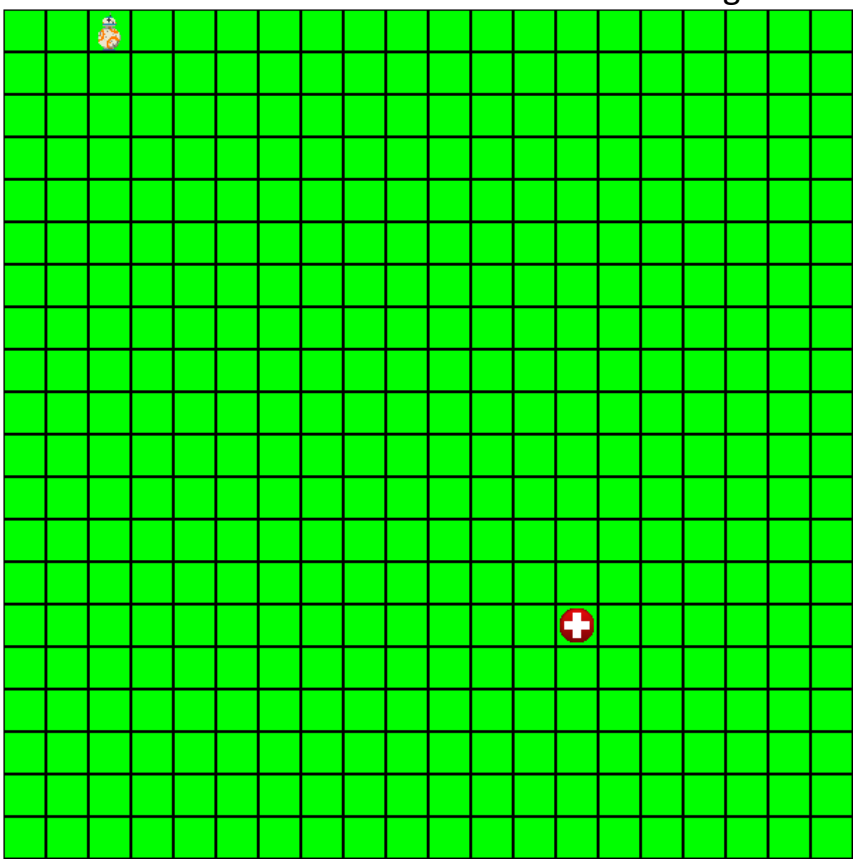
Lab 10

Enter your answers in this document and submit this to blackboard

Enter Your Name here:
=>

Gridworld Explained:

In my C++ implementation of Gridworld, the Robot (Droid) must find the most efficient path to a Healthpill, but it has no idea where it is. Its world is a 20 x 20 grid:



If the Droid's "learning" attribute is set to **true** then the Droid's update method will execute the Reinforcement Learning algorithm (Q-learning) and produce a q-table which can then be used by the Droid to navigate to the Healthpill.

The algorithm utilises the **Environment class** to perform the RL. It simulates the Droid's environment and allows us to experiment with it. We can create an **Agent** in the environment and tell the Agent what to do. The **doExperiment** method is where we run the RL algorithm. In here we will get our Agent to try lots of different moves in the Environment and see what happens by examining the reward we get back.

As per the q-learning algorithm this is an iterative process, so we will try and get the Agent to traverse the grid to the healthpill thousands of times (in fact I have set it to 100,000 episodes). **Note:** we do not make any suggestions about how the Agent moves.
During each episode the Agent takes a number of steps throughout the grid ("looking" for the goal node). To ensure the Agent does not get stuck doing useless episodes (not reaching the goal in a reasonable amount of time), we limit the number of steps in an episode (I have set it to 5,000). Each step will either be a random move or the best move from the current state as recorded in the q-table (exploration v exploitation). We use a probability to decide which to do.

As we complete each episode and build more knowledge about our environment, we want to explore less and exploit our knowledge more. So, we gradually reduce our Alpha and Epsilon parameters:

```
decayAlpha();       // Our learning rate
decayEpsilon();     // Probability of choosing a random move
```

Once we have completed all episodes the resultant q-table is written to a file specific to the droid e.g. qTable_D1.csv, where "D1" is the name of the droid. This file can then be reused by setting the Droid's learning attribute to **false**.

To do:

1. Interrogate the **doLearning()** method of the Droid class. It utilises the Environment class to perform the RL. The resultant q-table is written to a file. This file can then be reused by setting the Droid's learning attribute to false. Run the application in learning mode. How many states are there in the q-table file?
   Answer =>

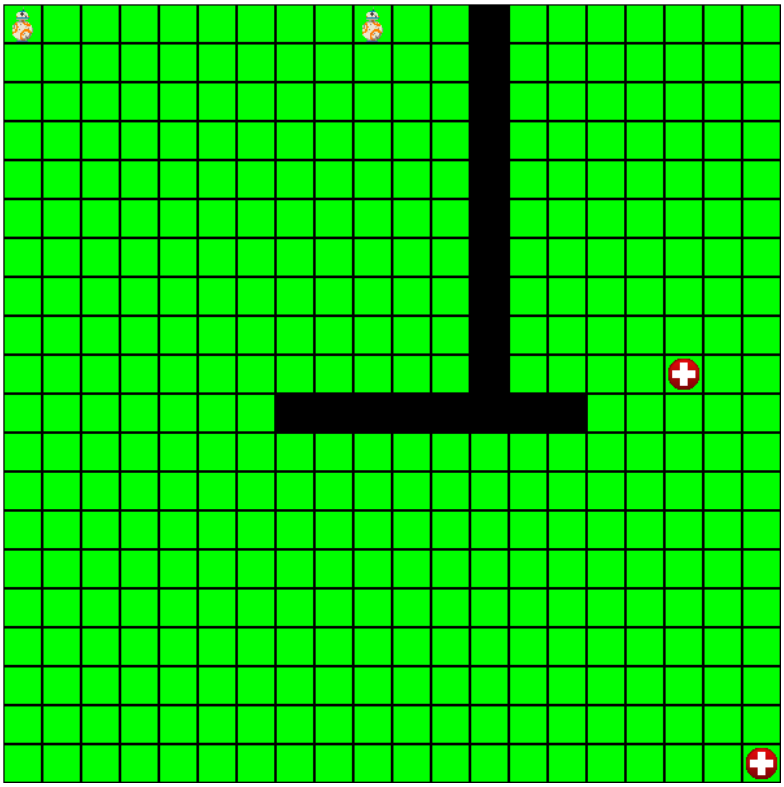
2. Reposition the droid to start at position (1, 1) and the Healthpill at (20, 20) and re run the learning. Is the q-table different? Explain why or why not.
   Answer =>

3. Now let's examine having multiple Healthpills and how the Agent can learn the best routes to both:

a. Create a second Healthpill at position (10, 18). Do this in the **initGridObjects()** method of the Game class.
b. Set the Droid's **goalNo = 2** this time
c. Rerun the program with **learning = true** and confirm it reaches the second new Healthpill.
d. Set the **learning = false** and the **goalNo = 1** and rerun the program. Does the Droid reach goal 1 this time?
   Answer =>
e. How many states are there in the q-table now? Why?
   Answer =>

4. Next, let's examine having more than one Droid.
   a. Create a second Droid instance called "D2" at position (10, 1)
   b. Set the **learning = true** and the **goalNo = 2** (each droid must do its own learning if it has a different start position, and will produce its own q-table.
   c. Run the program. What are the names of the q-table files created?
      Answer =>
   d. Do both Droids now travel to their respective Healthpills?
      Answer =>
   e. Turnoff the learning for both droids, swap their target Healthpills. Do they reach the other Healthpill without relearning? Explain why or why not.
      Answer =>

5. Finally, lets add in some obstacles. The trick with 'learning' about obstacles is to treat them as if they were simply invalid moves, just like we do if the Droid chose to move off the grid. In other words, the Droid gets a negative feedback and simply stays where it is. It will eventually learn that it is not a good policy to do that.
   a. So, add some obstacles in a pattern of your choosing (see my example below). Probably easiest to hardcode some into the **initGridObjects()** method. I have already created a **bool containsObstacle** in the Node class which you can set, and maybe modify the Node **draw()** method accordingly. Then simply check this during each Timestep of the Learning process.



b. Demonstrate this working by attaching here a screen recording of your 2 droids avoiding obstacles and navigating to different Healthpills.

Screencast =>