Red/Green Light Method

The goal of traditional Reinforcement Learning algorithms is to understand a specific environment and to pick the best **action** for a given scenario. For example: if a Reinforcement Learning agent is trained to understand the environment of Super Mario Bros., Mario's possible actions are walk left, walk right, run left, run right, and jump. If Mario was standing right next to a goomba, the agent would say that the *best* action for the scenario would be to jump so that he avoids touching the goomba and stomps on his head.

The "Red/Green Light Method" has a similar goal, but it is more of a combination of Reinforcement Learning and Classification Models. The Red/Green Light Method is binary—its only goal is to decide if an action is a good (green) idea or a bad (red) idea. Instead of having an agent to choose the best possible action in a particular environment, we look at each action as more of a **skill**. If we trained Mario using the Red/Green Light Method, he would ideally refine his walking skill, his running skill, and his jumping skill. Then, his jumping skill would tell Mario that jumping is in fact a good action when he is next to a goomba.

To reiterate, the end goal of the Red/Green Light Method is to create classification models for specific skills, such as Mario walking, and to have it classify whether executing that skill is a good idea or a bad idea. The inputs to the model are the environment—the coordinates of where Mario is, if/where a goomba is present, if/where a coin is present, etc. During training, Mario would be put next to a goomba. In one scenario, he jumps and avoids the goomba. That's a green light for Mario, he did good. The Mario and the goombas positions would be recorded and would be labeled as a 1 for training. In another scenario, Mario doesn't jump, the goomba hits him, and Mario dies. That's a red light for Mario, he did bad. Everything would be recorded again, but this time it would be labeled -1 for training.

Training Method

In this simulated Matlab environment, there exists two separate objects. One is a **Robot** object that has a floating point (x, y) coordinate as well as a pointing angle (0° - 360°) that can be turned clockwise or counterclockwise. The Robot can also move a given distance in the direction of its current pointing angle. The other object is a **Destination** object that only has a floating point (x, y) coordinate. The goal of this is to get the Robot to navigate itself to the Destination using Machine Learning.

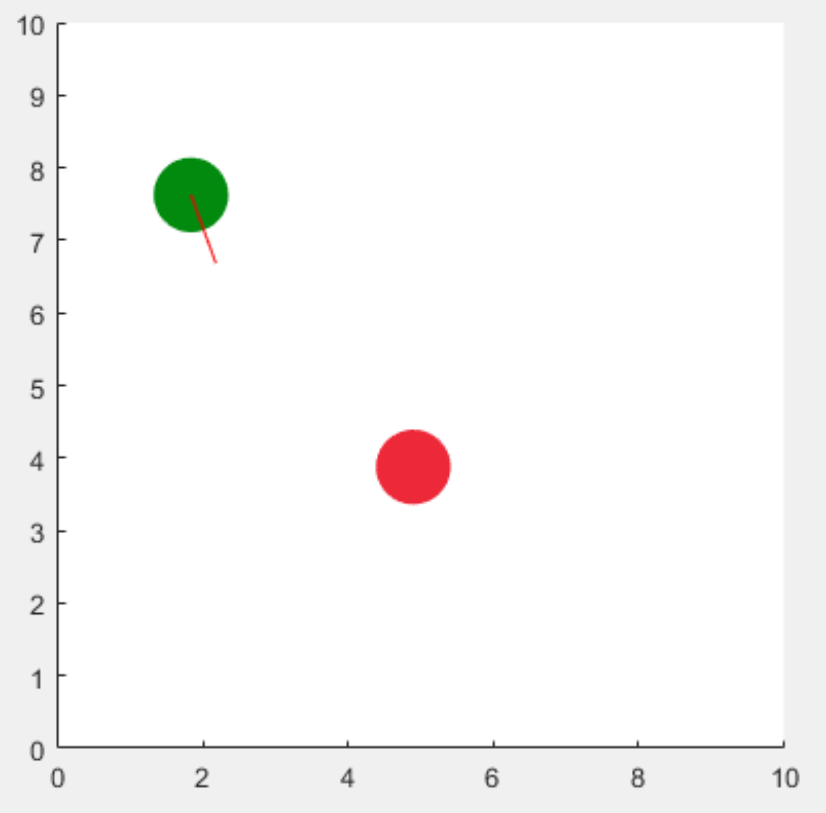

Figure 1: Simulated Matlab environment showing Robot (green) and Destination (red)

The robot is trained to have two separate skills—a turning skill and a moving skill. The turning skill model predicts whether turning clockwise or counterclockwise is a good idea. The moving skill model predicts whether moving forward is a good idea. Both skills are trained using a combination of Reinforcement Learning ideology as well as standard classification model algorithms. Labels for training are either 1 (green light—good) or -1 (red light—bad).

The turning skill is trained by generating 1000+ random locations for both the Robot and Destination as well as initial pointing angles (in other words, 1000+ unique environment instances). The computer calculates the angle between the Robot and the Destination—the angular difference between the Robots current pointing direction compared to where the Destination is (see Figure 2). The computer then has the Robot turn clockwise *and* counterclockwise from this initial position. If turning a direction reduces the angle between the Robot and the Destination, it is labeled 1—green light/good. Otherwise, it is labeled -1—red light/bad. The whole environment is recorded as features—the x & y distance between the Robot and the Destination, the Robot's current pointing angle, the current angle between the Robot and the Destination, and the turn direction (CW/CCW). The good/bad labels are also recorded for the training purposes depending on whether turning CW/CCW reduced the angular difference for the environment. The training data is then passed through a Decision Tree Classification algorithm to train a model.

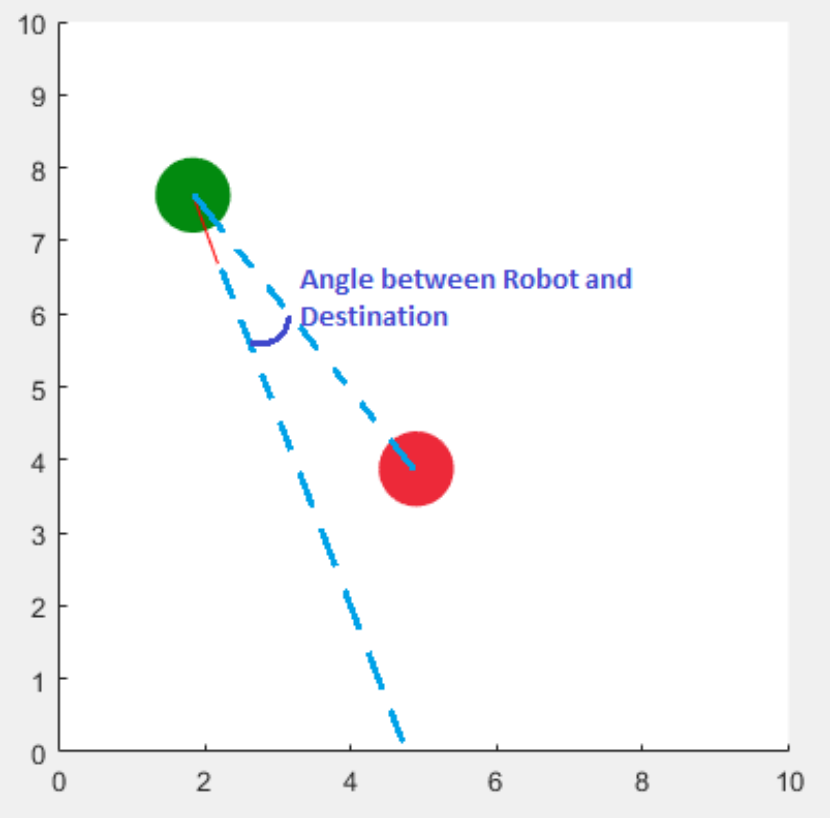

Figure 2: Depiction of what the "angle between" the Robot and Destination is

The training of the movement skill is essentially the same except it is trained to determine whether moving the Robot forward will reduce the distance between the Robot and the Destination, as opposed to reducing the angle between. The x & y distance between the Robot and the Destination, the Robot's current pointing angle, and the good/bad label are all record and a Decision Tree Classification algorithm is used to train a model.

Using the Skills

To *use* the skills, the computer creates a feature vector of the current environment for each possible action: turning clockwise, turning counterclockwise, and moving forward. For example, let's say we have an environment with a Robot at (1, 1) pointing at 0º and a Destination at (5, 5) (see Figure 3 for environment depiction).

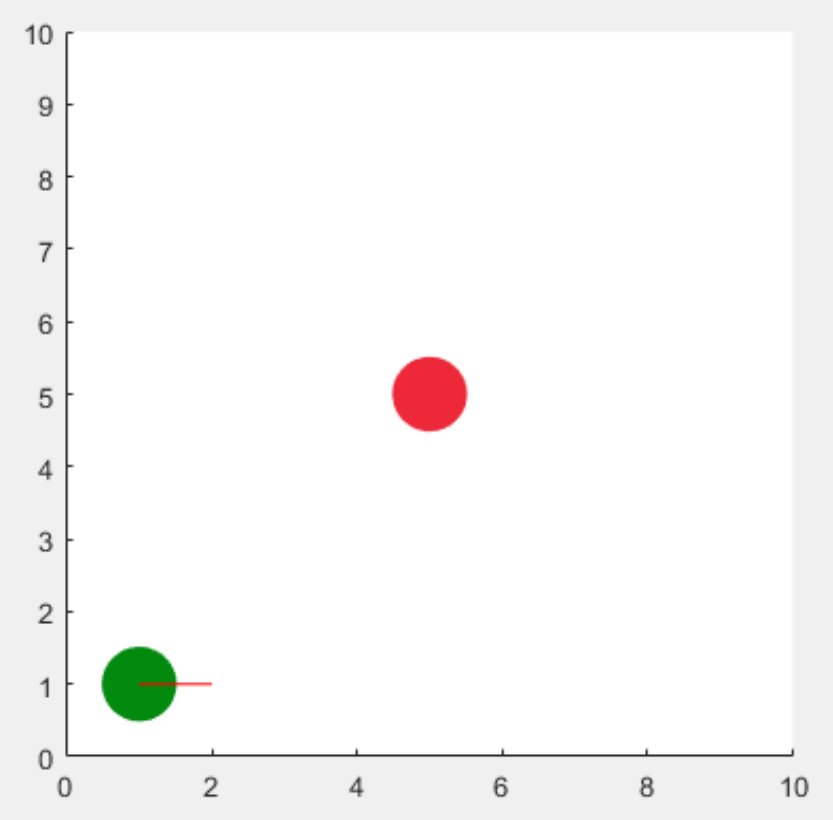

Figure 3: Depiction of example environment

The feature vectors for the environment would look like this:

Skill	Delta X	Delta Y	Robot Angle	Angle Between	CW/CCW
Turning	4	4	0º	45º	1 (CW)
Turning	4	4	0º	45º	-1 (CCW)
Moving	4	4	0º	N/A*	N/A*

*Not necessary information for Moving skill

We can now ask the Turning Skill model if turning CW is a good idea. The Turning Skill would hopefully predict that turning CW is a bad idea as it would point the Robot further away from the Destination. What about turning CCW? The Turning Skill should say that turning CCW is a good idea. Take note that we have to ask the Turning Skill model to make a prediction twice—once to check CW and another to check CCW since turning has two possible directions. The Robot only moves forward, so we only have to ask the Turning Skill model whether or not to move once for this iteration. The Move Skill would say moving forward is a good idea because it would get the Robot closer to the Destination.

After executing a 1º CCW turn and a 0.1 movement forward, the environment has changed. The Destination is still at (5, 5), but the Robot is now at (1.1, 1) and pointing at a 1º angle. So, we must update the feature vectors for both the Turning Skill and Moving Skill and ask the models to predict whether an action is good or bad all over again. Repeating this loop over and over again should get the Robot to the Destination.

Skill Hybridization?

Currently, the two skills are working towards the same goal—getting the Robot to the Destination—however, they aren't exactly aware of one another. I wonder if a valuable, further step would be to experiment with **hybridizing** the two skills together—to create another model whose skill is to understand the best way to get the separate skills to work together in sync.