David Tran - davtr766
Adam Rohdin - adaro815

Lab5 - Reinforcement Learning

The second part of the lab assignment is to implement a Q-learning controller that will construct the rocket to face up at all times. This requires three implementations in order to achieve this behavior:

- A state space representation and a reward function based on the sensors.
- An action representation.
- Q-learning algorithm.

The objective is to face up, therefore the state space and reward function should rely on the angle of the rocket. We use two different state and reward functions, one for hover and one for angle. For this part of the lab, only one of them is necessary which is the state and reward functions for the angle. The state function that controls the angle are discretized between $-pi/2$ and $pi/2$ into four different states. The reward function is based on the angle where we use an exponential function to calculate the reward. We get better reward values if the rocket is facing upwards.

For the action representation we have four actions as suggested in the lab description which is none, left, right and middle. For the given state we calculate an action which is an integer which then maps to a specific action. This is done so the rocket will choose the most appropriate action for the given state. Implementation is given below:

```
/* Performs the chosen action */

void performAction(int action) {
    /* Fire zeh rockets! */
    /* TODO: Remember to change NUM_ACTIONS constant to reflect the number of actions (including 0, no action) */
    /* TODO: IMPLEMENT THIS FUNCTION */

    resetRockets();
    switch(action){
        case 0:
            break;
        case 1:
                leftEngine.setBursting(true);
                break;
        case 2:
                middleEngine.setBursting(true);
                break;
        case 3:
                rightEngine.setBursting(true);
                break;
        default:
            break;
    }
}
```

In Q-learning, we need to keep track of estimated values for taking an action in a given state. Each time an agent moves the Q-values can be updated by nudging them towards the observed reward and the Q-value of the observed next state. In this lab, a hash map is used to keep track of the Q-values where the key is a state s and the value of the key is the Q-value. To update the Q-values we use the following formula:

$$Q_{new\ state} = Q_{prev\ state} + \alpha * (R * \gamma * max(Q_{new\ state}) - Q_{prev\ state}$$

where α is the learning rate, R is the reward value which is observed in the current state and γ is the discount factor which determines the importance of future rewards. The new Q-value is then stored in the hash map with the corresponding state.

David Tran - davtr766
Adam Rohdin - adaro815

For the third part of the lab the objective is to learn the rocket how to hover. In the state function of the hover feature we create a state variable as a composition of the angle state as well as a discretized values based on the velocity in x and y. The last function is the reward function for hover. In this function we calculate the reward based on the velocity of x and y, as well as on the reward function for the angle. To get a better result we apply different weights to each part because specific parts are more important than other, in our case our conclusion is that the reward for the angle and the velocity in y-axis have a more important role than the velocity in the x-axis.

Question 1: In the report, a) describe your choices of state and reward functions, and b) describe **in your own words** the purpose of the different components in the Q-learning update that you implemented. In particular, what are the Q-values?

a)

Angle State:

```
public static String getStateAngle(double angle, double vx, double vy) {

    int numberOfStates = 4;

    int discretizeValue = discretize2(angle, numberOfStates,-angle_constraint, angle_constraint);
    String state = Integer.toString(discretizeValue);

    return state;
}
```

In this function we discretize the angle between -90 and 90 degrees. By using discretize2 we capture the return value between 0 and MAX.

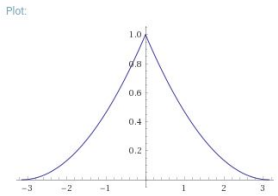
Angle Reward:

```
public static double getRewardAngle(double angle, double vx, double vy) {

    double reward = 1 - (Math.abs(angle)/Math.PI);
    reward = Math.pow(reward,2);
    return reward;
}
```

In this function we base the reward how close the space ships angle is to zero. If the ship is at zero radians (which is the best case) the reward function returns its highest value; 1.0. To get better performance out of the reward function we make it exponential.

Here is an illustration of the function.



Hover State:

```
public static String getStateHover(double angle, double vx, double vy) {

    int discy = discretize(vy, 4, -0.5, 0.5);
    int discx = discretize(vx, 4, -1.0, 1.0);

    String state = Integer.toString((Integer.parseInt((getStateAngle(angle, vx, vy)) + discy + discx)));

    return state;
}
```

In this function we first discretize the velocity in both x and y. We do this to calculate the return state as a composition of the state from x and y, as well as the state given by the angle. We do this since our intuition is that the stability of the ship is related to the velocity and angle of the ship.

Hover Reward:

```
public static double getRewardHover(double angle, double vx, double vy) {

    double vyRew = 0.0;
    double vxRew = 0.0;

    if(Math.abs(vy) < 0.5)
        vyRew = 0.5 - Math.abs(vy);

    if(Math.abs(vx) < 1.0)
        vxRew = (0.5  - (Math.abs(vx)/2.0));

    double reward = 1.5*getRewardAngle(angle, vx, vy) + 1.5*vyRew + vxRew;

    return reward;
}
```

Here we start by calculating the reward based on the velocity in x and y. This is done in a linear fashion, something which might not be optimal. In the best case we want to do this as an exponential function since it creates a more distinct value difference. After this is done we calculate the final reward value as a composition of the reward value for velocity x and y as well as the reward for the angle. We also weight the values differently depending on how much of an impact we think they have on the stability of the ship. In this case we value angle and vertical velocity.

b)

As mentioned before, the Q-learning update algorithm's definition is as follows:
$Q_{new\ state} = Q_{prev\ state} + \alpha * (R * \gamma * max(Q_{new\ state}) - Q_{prev\ state}$
The components of the Q-learning update:

- α is the learning rate which essentially is the step size that is taken towards the solution which should converge to a solution. The time it takes to find a solution is depended on the value of the learning rate.
- R is the reward value which is observed in the current state.
- γ is the discount factor which determines the importance of future rewards. Depending on the value of the discount factor, the agent can consider current rewards over future rewards instead.
- $Q_{prev\ state}$ is the computed Q-value in the previous state and $max(Q_{new\ state})$ is the estimation of the optimal future Q-value.

Question 2: Try turning off exploration from the start before learning. What tends to happen? Explain why this happens in your report.

When turning of exploration from the start the algorithm tends to get 'stuck'. By this we mean that the algorithm does not improve with any significant rate. This is due to the fact that the algorithm does not try new approaches to solving a problem nor might it find itself in any new cases when trying to learn. This can result in a bad and non-complete state table with few or bad chosen actions.

Question 3: For simplicity, we here constructed a discrete (table) approximation for a continuous problem, which can be a pretty rough approximation. As mentioned in the RL lecture, *Fitted* Q-iteration is a technique where supervised learning is instead used to approximate Q(s,a) inside the Q-updates. Suggest some sufficiently powerful model/representation that could be used to learn an arbitrary Q-function given examples from the update.

One fitted Q-learning algorithm that could be used is 'Neural Fitted Q-learning'. This algorithm tries to minimize the *error* by estimating the next Q-value.

David Tran - davtr766
Adam Rohdin - adaro815

Question 4: For this question, assume you had plenty of computational power rather than a weak lab computer. Like above you want to approximate Q(s,a), but instead of s being simplified state like angle and position, let s be all raw pixel values from the rocket simulator graphics that you are shown. What would a suitable representation for the Q-function be for supervised learning it from image inputs? (Hint: The ML lecture on deep learning might be useful)

Convolutional Neural Network(CNN) is a model where Q-function can be applied. In supervised learning we have a data set of input images where some of the input is the solution. We learn the CNN which has pooling layers, fully connected layers and ReLu filters (the architecture of the CNN is depended on the purpose of the CNN). The result are Q-values that corresponds to the given solution in the image, in this case we want the rocket to hover in the middle of the environment. Therefore, the Q-values will be higher in the middle suggesting that the goal is in the middle of the environment.