29 sep - 2016
Gustav Jannering - gusja113
Rasmus Johns - rajso813

Different parts of Q-learning

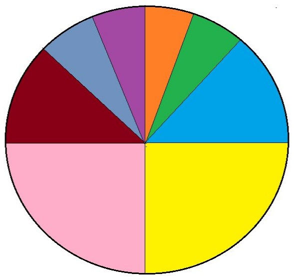
The components of the Q-learning in our implementation was:

- State: The different states in the state space.
- Actions: The actions which can be performed.
- Q-lookup table: The combination of state and actions. “If in state S, perform action A”.
- Reward: The calculated value used to incentivize good behaviour.
- Alpha: The rate at which old knowledge will be replaced by new knowledge.
- Gamma: How long- /short-sighted the program will be.
- Max QValue: The value multiplied by gamma, resulting in a weight representing the future.

State and Rewards

Part II

We used the following states (where each color represent a state):



In order to promote angular equilibrium we used the following reward-function:

$$reward = Math.pow(Math.PI, 2) - (Math.pow(angle, 2));$$

This promoted the agent to learn that “good” angles gave a good reward, while a poor angle resulted in low reward. On top of this, we also found that $\gamma = 0.5$ gave us a good result.

If exploration is turned off when the program is started. Since the program does not try any new actions, no action is performed, meaning that the ship will fall endlessly.

Part III

For this part, we kept all code from part II. On top of that, we we extended our spates:

$$state = angle\_state + " : " + discretize(vx,\ 3,\ -3,\ 3) + " : " + discretize(vy,\ 10,\ -3,\ 3);$$

This extended our state space to support several new states, covering the velocity.

However, in order to utilize these states, we needed to adjust our reward-function from part II. We chose to create a reward-function as *the sum of three rewards*: the angle reward from part II, a reward for x-velocity and a reward for y-velocity. Since we already had our angular reward prepared, all we had to do was create a reward-system for velocity.

Unlike the reward given for angular accuracy, the velocity reward could not utilize subtraction in its formula (seeing how the upper velocity limit was blurry, whereas the angular limitation was $\pm\pi$). We therefore improvised the following reward-function:

$$vy\_reward = Math.pow(Math.PI,\ 2)\ /\ Math.pow(2,\ Math.abs(vy));$$

$$vx\_reward = Math.pow(Math.PI,\ 2)\ /\ Math.pow(2,\ Math.abs(vx));$$

Our goal when constructing this part of the reward-function was to reward good behaviour with huge rewards. However, we also wanted the reward-function to rapidly decrease the reward sizes as velocities increased (thus discourage floating).