**Part II: Q-Learning Angle Controller**

This requires you to implement three parts:

1. A state space representation and a reward function function based on the sensors (inputs)
2. An action representation (possible outputs)
3. The actual Q-learning algorithm.

To do this you will be editing two files, QLearningController.java and StateAndReward.java. This will later be extended for the final hover controller in part III below.

As the only objective is to face up, the state space and reward function for this part only needs to rely on the angle of the rocket. The states need to be discrete in this simple table version of Q-learning.

**Tasks for part II:**

1. Implement the state and reward functions for the **angle** controller in StateAndReward.java. Test it by flying around manually and watch print-outs of the state to make sure all the important states are covered.
2. Implement the action representation in the method performAction(int action) in QLearningController.java. Test it by temporarily setting it to perform some action in tick(), the main decision loop.
3. Implement the Q-learning algorithm in QLearningController.java according to the instructions above. Run it for a sufficient number of iterations until it stabilizes, <100k should be enough for most representations with the angle controller. Remember to turn off the exploration while evaluating it (see instructions for the simulator above). If it calms down and faces upwards it has converged to a good policy, otherwise let it continue for a while more or revise your choices above.
4. Experiment with turning off the controller on 'P' and manually maneuvering it into bad situations before turning the controller on again on 'O'. Make sure that it seems to recover eventually.
5. In the report describe your choices of state and reward functions, and describe **in your own words** the purpose of the different components in the Q-learning update that you implemented.
6. Try turning off exploration from the start before learning. What tends to happen? Explain why this happens in your report.

**Part IV: The Full Q-Learning Hover Controller**

Extend this to full hover control by simply implementing state and reward functions that correspond to the task of hovering. The state and reward function calls in the tick() function need to be changed to point to these new hover versions.

**Tasks for part IV:**

1. Implement the state and reward functions for the hover controller in StateAndReward.java and point the corresponding function calls in the tick() method in QLearningController.java to your new functions. If you cannot get behavior that is even close to hovering after 500k iterations you need to either revise your state and reward choices according to the instructions above, or there may be a bug in your Q-learning implementation. A good idea can be to turn off the controller on 'P' and maneuver it off-target and then turn it on again to see how it recovers like you did with the angle controller in part II. Do not forget to turn off exploration on 'E' when evaluating it!
2. Demonstrate your solution to the lab assistant and hand in a report describing your solutions to part II and III, including the two questions in part II. As it can be difficult and time consuming to get perfect hover behavior we are fairly lenient as long as it can be seen that it is at least trying.