Atta Ebrahimi

Rebecca Hwa

CS 1571

Project 3 – Flappy Bird Q Learning

**Related Files:**

* flappybird.py, game driver
* Learner.py, implements the q\_array, agent, state, and action classes for Q learning
* q\_array is the file with the learned states

**How to run:**

* “python flappybird.py q\_array” this runs the learned bird
* “python flappybird.py” this runs with a fresh q\_array

Note:In order to save the q\_array you must pause while it is running and this will write the bytes to the q\_array file. Allow 5-10 seconds for this after a pause.

**Training and Implementation:**

My state was represented with five things:

* dx = pipes x coordinate + pipes width – bird y height
* dy = window height – bottom pipe y height – bird y height
* Boolean value if bird is dead or not
* Boolean value if bird is jumping or not
* Pipe window location, whether pipe window is in the bottom, middle, or top third of the window

**Actions:**

I had my agent choose an action and learn every single frame as long as the transition would land into a new state. This proved troublesome as the time it took to learn was ridiculous, but I didn’t want to limit the states in the way discussed in class as I had already mostly finished the project. If a state had equal Q actions for both actions, I biased the bird to jump if the position was lower than the pipe and to not jump if it was not. This prevent a lot of bad deaths where the bird would fly through the top or fall off the bottom. If this was not the case an action was chosen based on whichever one had the higher Q exploration value. max(Q(s, jump) + 5/N-jump, Q(s, idle) + 5/N-idle) returns the action associated with the max. This function allows for more exploration of infrequent actions in the earlier learning stages.

**Q\_array:**

This is an encapsulated python dictionary object where the key is a state object and the value is an ActionSet object containing the Q values for each action at that given state. It is dumped to file in bytes with pickle, hence the weird characters.

**Learning:**

For my alpha value I just picked random numbers between .1-.9 and .9 seemed to work really well. Gamma was done the same way and .8 worked well. After a state transition that produces a new state, I updated the previous state, action pair using the following equation:

Q(s, a) = Q(s, a) + alpha\*(reward + max(Q(s’, a’jump, Q(s’, a’idle) ) - Q(s, a) )

Where Q(s, a) is the previous state and action that lead to Q(s’, a’) which is the new state with a reward.

**Reward:**

If the bird dies from flying off the top or bottom the reward does -11000. If it is done by a pipe or collision it is -3000 + the Euclidian distance from the pipe. If the action is produced in which the bird lives the reward is 1 + 15 / (euclidian distance from pipe). If the bird makes it through the pipes the reward is 8000.

**Game speed modification:**

In order to cope with the fact that I naively took a state from each frame, which makes the state space ridiculously large, I increased the game speed by a factor of 10 to increase the learning. In order to learn at this speed, the constant OVERALL\_SPEED, and FPS must be 10 and 60 respectively. Training in this way caused the Q learning to not function properly. In order to fix this, I had to limit the frames to 6 so that it would consistently get above 10-20 pipes. The program still works learning under regular conditions, however I did not have time to relearn under that case and just left it as is at 6 fps, this still proves that it learned so I decided to leave it.