AI Agents for Flappy Bird

Tai Vu, Leon Tran

OVERVIEW

Game: The bird moves according to in-game gravity; the player must avoid pipes at random points by tapping on the bird to make the bird jump vertically.

Problem: Choose the action for each game step that maximizes score.

Solution: Use Q-Learning to let the bird estimate the optimal action in each position.

MODELING

State: (x_distance, y_distance, y_velocity)

- x_distance = bird to the nearest pipe.
- y_distance = difference of the bird's y-position and the lower pipe's position.
- y_velocity = how fast the bird is falling.

Action: {0, 1}

- 0 is flap, 1 is don't flap.

Reward:

- -1000 if bird dies.
- 5 each time the bird passes the pipe.
- 0.5 each time the bird survives a time step.

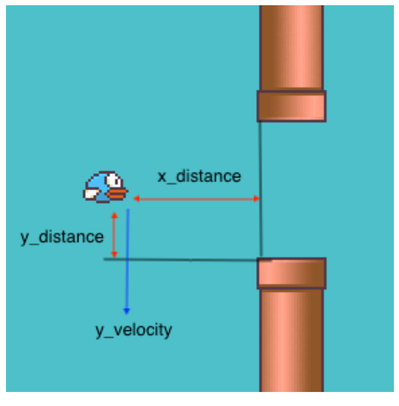

Figure 1: Representation of State.

APPROACHES

- **Q-Learning** allows an agent to estimate expected future reward for any tuple (state, action).
- The agent may then choose the action that yields the greatest expected future reward.
- We modify Q-Learning in the following ways:
  1. Use an **epsilon-greedy** approach.
  2. **Discretize** screen into 5x5 and 10x10 grids to limit state space size.
  3. **Forward vs Backward** Q-Learning (updating Q-values from least to most recent vs most to least recent).

Algorithm 1 Q-Learning

Initialize all Q-values to 0
for iteration = 1, N **do**
  Initialize memory $\mathcal{M}$
  Set initial state s_1
  Set $t = 1$
  while s_t is not terminal **do**
    With probability ϵ select a random action a_t from $\{0, 1\}$
    otherwise select $a_t = \max_a(Q(s_t, a))$
    Perform action a_t and observe next state s_{t+1} and reward r_t
    Store observation (s_t, a_t, r_t, s_{t+1}) in $\mathcal{M}$
    Increment t
  end while
  for observation (s_i, a_i, r_i, s_{i+1}) in $\mathcal{M}$ **do**
    Set $Q(s_i, a_i) = (1 - \eta)Q(s_i, a_i) + \eta(r_i + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1}))$
  end for
end for

Figure 2: Our Implementation of Q-Learning

- We choose **hyperparameter values** $\eta = 0.9, \gamma = 1,$ $\epsilon \in \{0, 0.1\}$.
- We run the program for 3250 iterations.

FUTURE WORK

- Fine-tune hyperparameters.
- Experiment with different discretization levels.
- Construct neural networks and CNNs for function approximation.
- Implement Experience Replay.

RESULTS

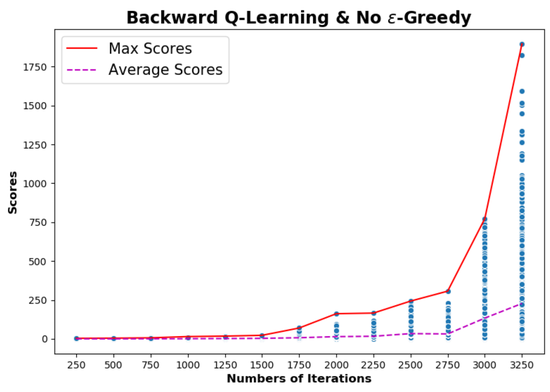

Figure 3: Training curve for Q-Learning with 10x10 discretization.

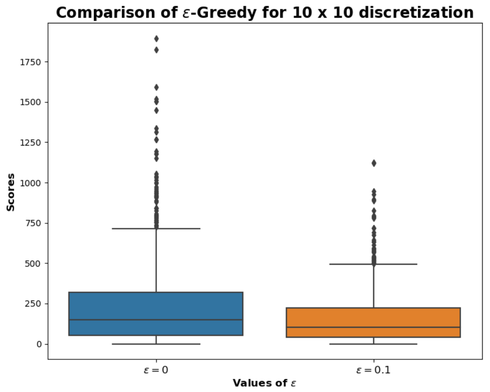

Figure 4: Comparison of ϵ-greedy approaches with 10 x 10 discretization at 3250 iterations.

Discretized?	Training Order	Epsilon	Mean Score	Std	Max
None	Forward	0	0.225	0.4800	3
None	Forward	0.1	0.251	0.5060	3
None	Backward	0	0.307	0.5734	4
None	Backward	0.1	0.408	0.6925	4
5x5	Backward	0	1.999	2.2002	14
10x10	Forward	0	10.474	10.5268	75
10x10	Forward	0.1	9.068	10.7047	69
10x10	Backward	0	227.642	256.7441	1896
10x10	Backward	0.1	154.824	160.2136	1125

Figure 5: Comparison of different Q-Learning agents at 3250 iterations.

DISCUSSION

- Discretized Q-Learning performs better because the state space is smaller. Visiting a state in a particular range allows us to generalize the Q-value to neighboring states.
- Backward outperforms forward training because it learns the most important information first (when it hits the pipe).
- 10 x 10 gives the highest score because it estimates the Q-values without overgeneralizing, which happens when the discrete regions are too large.
- $\epsilon = 0$ outperforms $\epsilon = 0.1$ because the discretized state space is small enough to explore almost completely; moving randomly doesn't benefit us.