FlappyBird.AI

Reinforcement Learning applications in games using Q-Learning and Neural Networks

Evan Calzolaio
(GitHub.com/Ecalzo/FlappyBirdAI)
Columbia Engineering Data Analytics Bootcamp

Abstract

This project implements both traditional and modern **Q-Learning** techniques in Python, allowing a program to learn to play the popular mobile game Flappy Bird on its own. This process is a type of **Machine Learning** known as **Reinforcement Learning**. Over time, the program learns to play by taking actions and receiving rewards based on those actions.

The Q in Q-Learning stands for “Quality”, as in the quality of an action taken at a given state. For example, a bad action, such as a crash, yields a very negative reward, while a good action, that results in the player not crashing, will yield a positive reward

Architecture

FlappyBird.AI is a class of the following structure:

`get_state(playerx, playery, pipes, velocity):`
- Returns the current values of the player state, creates a new q-table entry if one does not currently exist

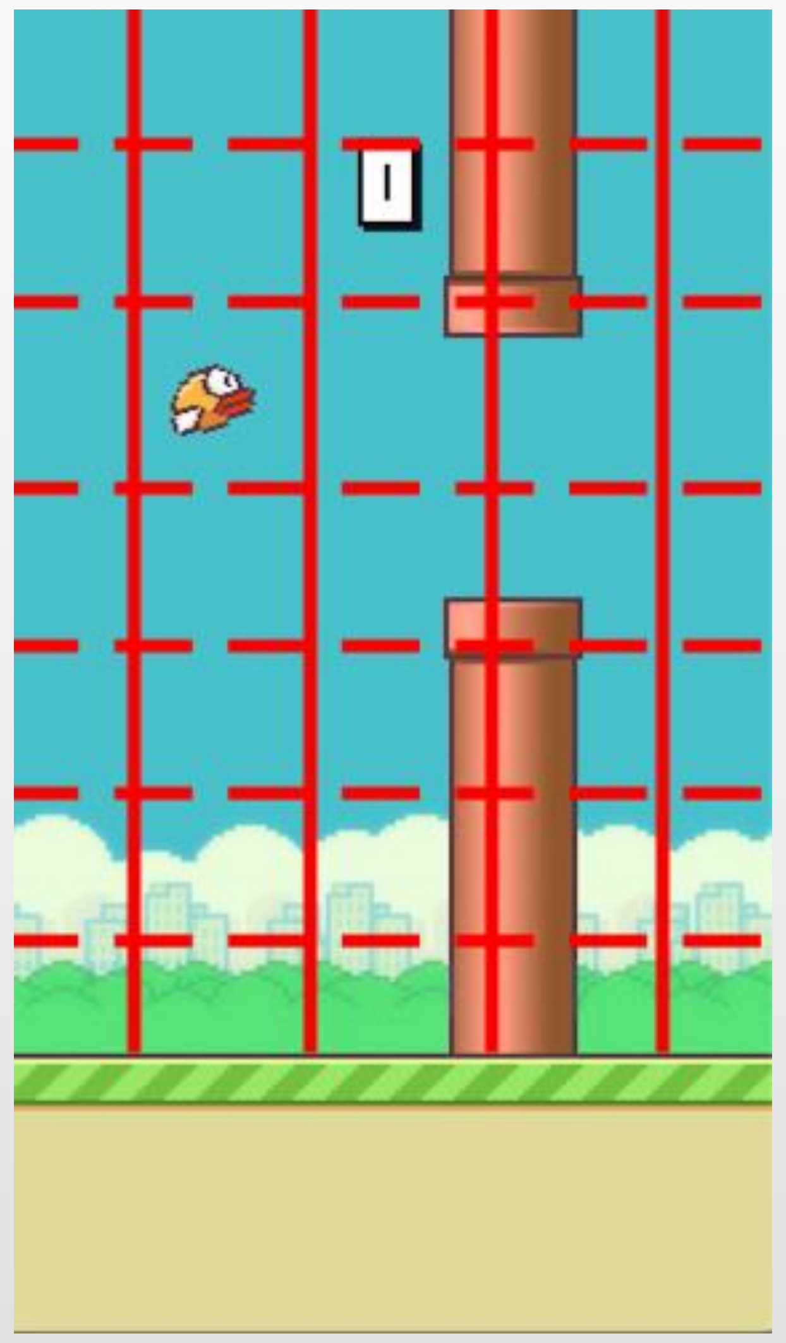
`do_action(state):`
- Chooses the best action based on the current value in the q-table (1 = flap, 0 = don’t flap)

`get_reward(crash?):`
- Rewards the player +1 if the player is still alive or -1000 if the player has died

`remember(state, action, reward, state’):`
- Appends the last SARS’ cycle to a memory array

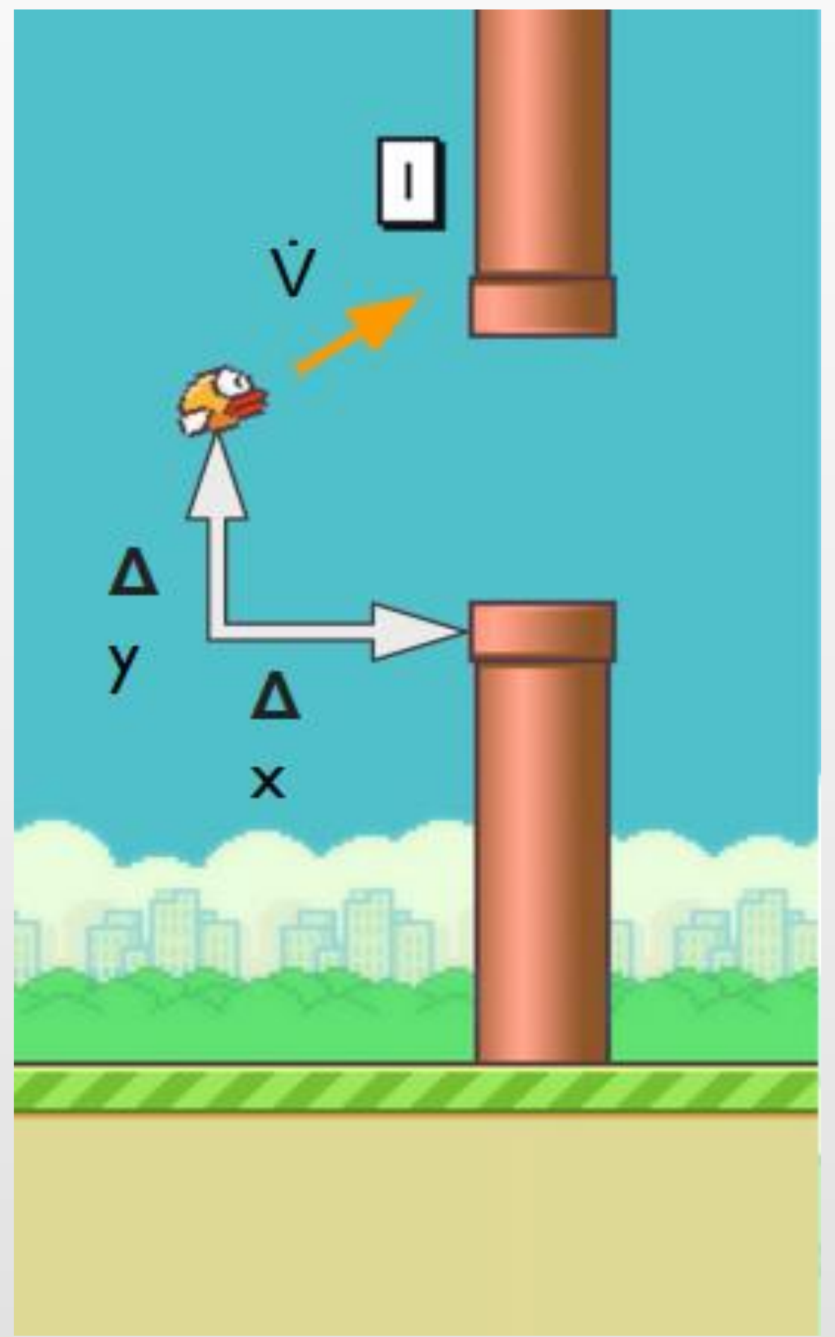
`replay_memory():`
- Cycles through a random batch of 1000 memories and updates the q-table based on the SARS’ outcomes (using the Bellman equation)

Methodology



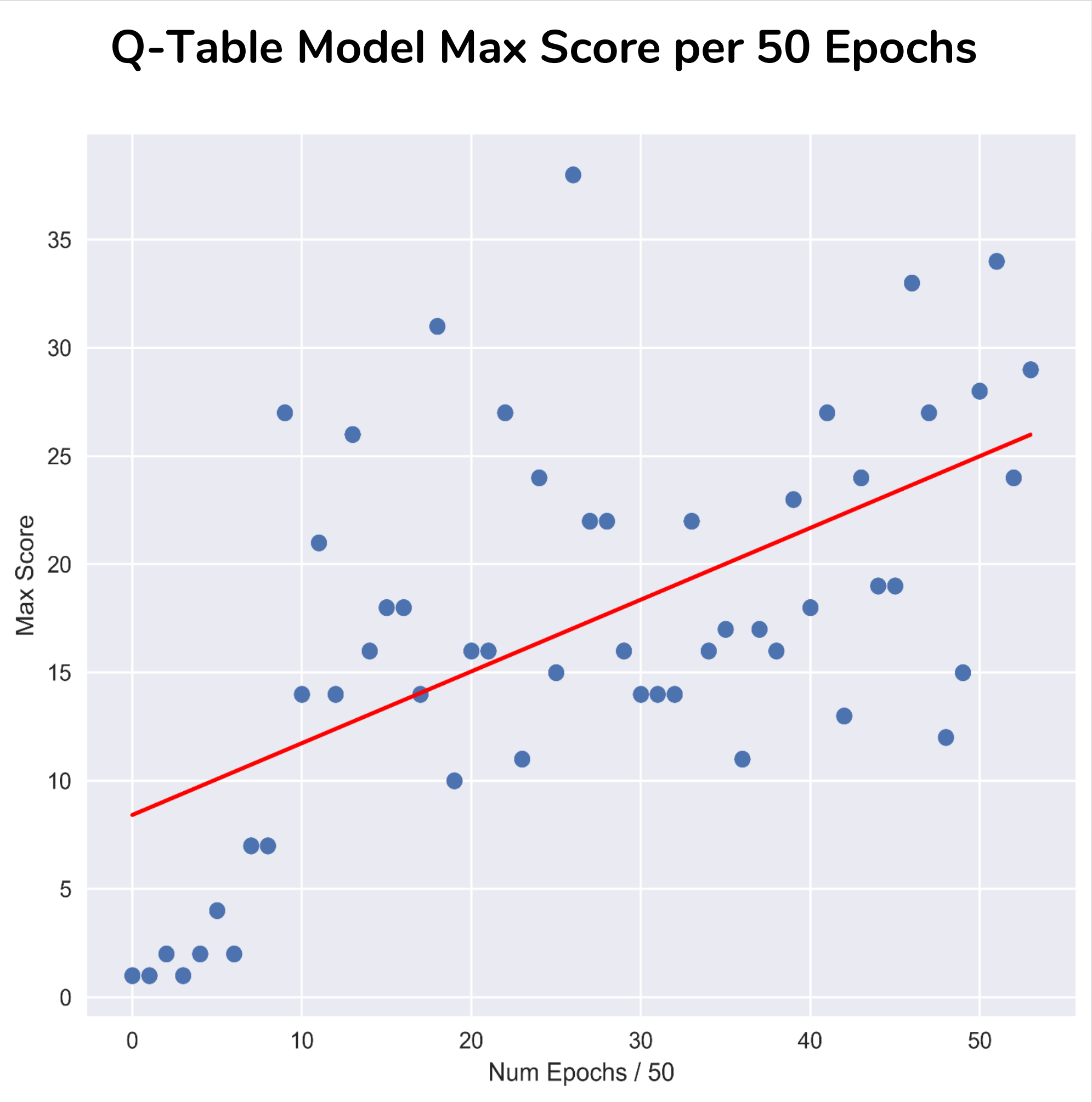
Discretizing the game space (Q-tables)*

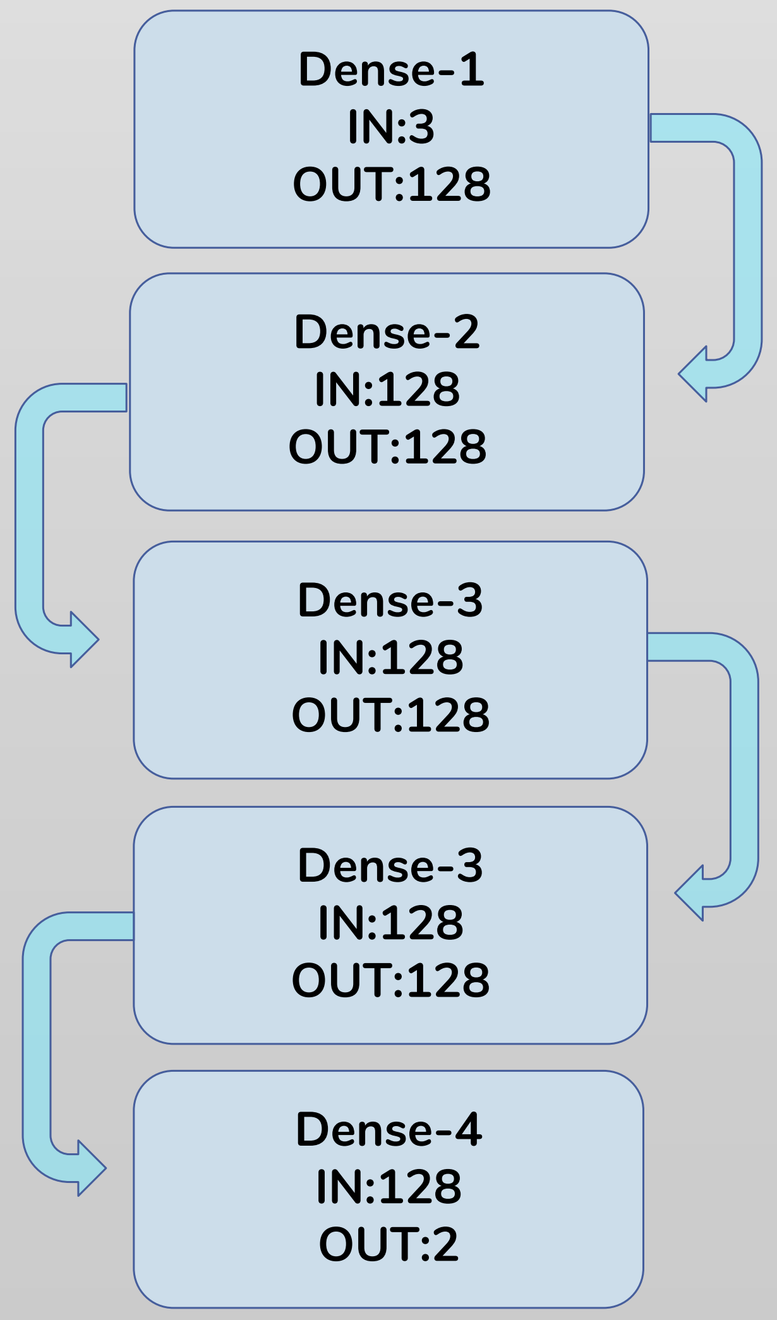
1. Measure the x and y distances from the nearest pipe, record the bird’s velocity
2. Divide the x and y positions into a grid structure (5 x 5), save this information at each state, then store in the model’s “memory”
3. Update the Q-table to determine the probability that an action will yield a favorable reward in the long-term
   - This is done with a **Bellman equation** altered for Q-Learning



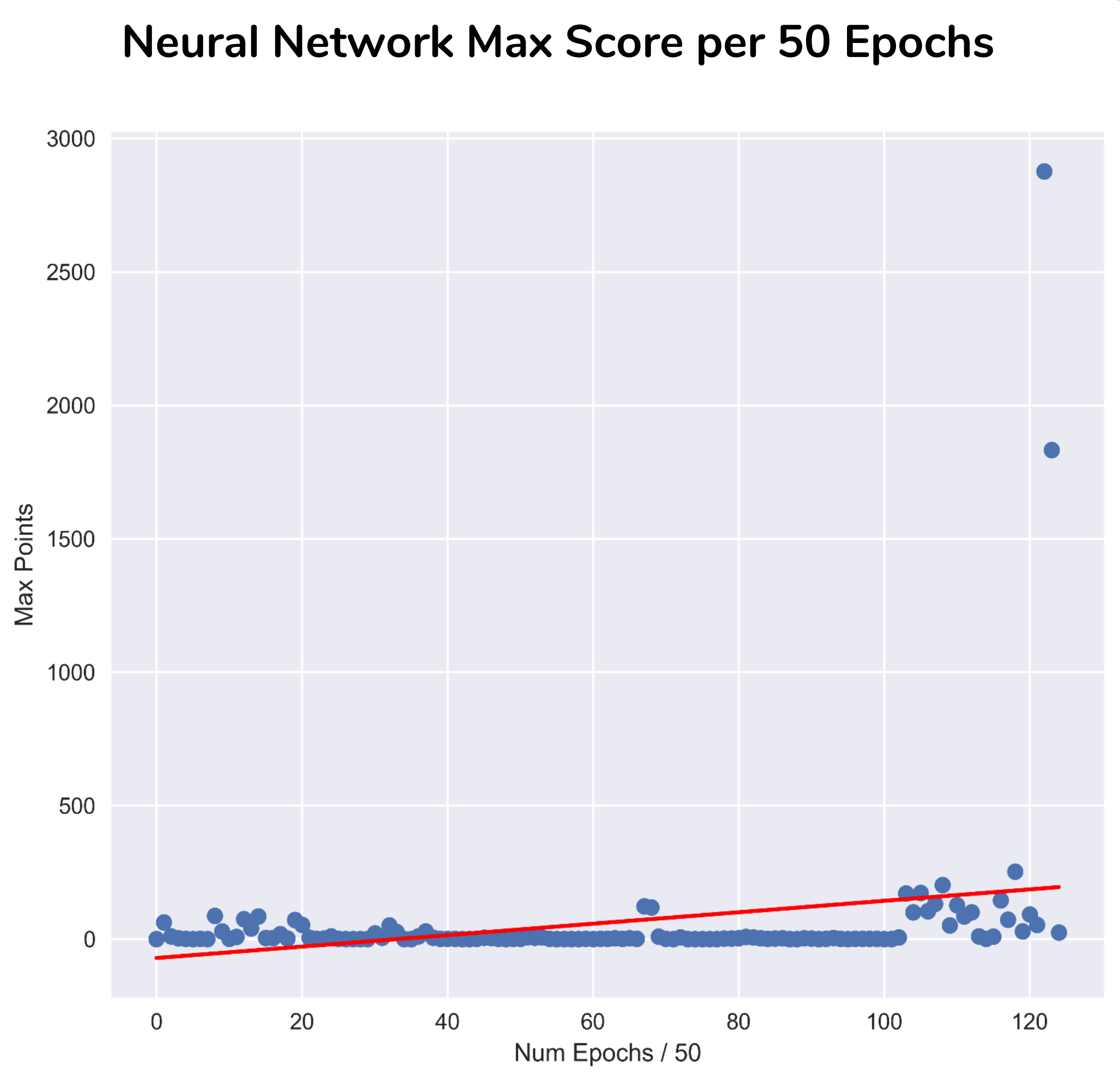
*With a Neural Network, there is no need to discretize the game space

Results


Q-Table Model Max Score per 50 Epochs



Neural Network Structure


Neural Network Max Score per 50 Epochs

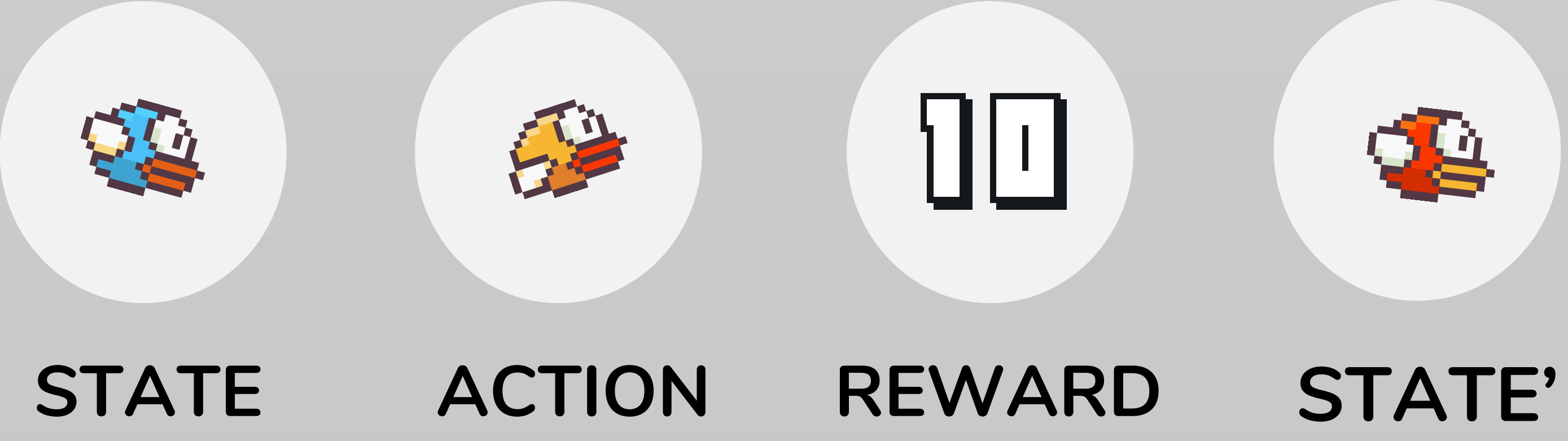
Noteworthy is the fact that, while the traditional Q-Learning model learns at a more apparent, steady rate, the Neural Network is able to obtain far more points overall

Reinforcement Learning

$$\boldsymbol{Q[s,a] = (1-\alpha) * Q[s,a] + \alpha * (r + \gamma * max(Q[s',a]))}$$

Bellman equation

- The **Bellman equation** is a component of dynamic programming, a mathematical optimization method
- The equation combines the immediate reward from some initial action with a “long-term” discounted, delayed reward, or the expected, future reward of the current state, action, reward, state’ values
- In this equation, Alpha is the learning rate, Gamma is the discount rate, and r is the reward received for the state, action pair
- This equation is used to update the Q-table and, ultimately, lead our program to decide which action it should take at any point in time



Acknowledgements

- PyGame port of Flappy Bird: GitHub user sourabhv
- Inspiration for DQN in games: GitHub user maurock and his Snake game
- Prof. Lee Tanenbaum for his continuous support and encouragement throughout this project