Flappy Bird using Q-Learning

Jesse A. Pelzar
Department of Electrical and Computer Engineering
Binghamton University
Binghamton, NY 13850
jpelzar1@binghamton.edu
https://jessepelzar.github.io/Flappy_Bird_RL/

1 Introduction

This project is based off of the well known mobile game Flappy Bird. With my strong background in web based technologies, I decided to recreate the game Flappy Bird within a web environment with the ability to not only play Flappy Bird, but the ability to have the the agent (the bird) learn how to avoid hitting the moving obstacles, floor or ceiling. The Reinforcement Learning component to this game was Q-Learning which allows the agent to learn through exploration and exploitation by optimizing the best action to take when in a given state. With a constantly changing environment, the agent must adapt to updated positioning of obstacles in order to avoid them. The importance of this project is to learn and adapt to a constantly changing environment while utilizing similarities such as comparing distances to achieve a different reward. The methods I incorporated are interesting and have the ability to be applied to other environments that may resemble a similar concept but consist of different states and actions.

2 approach

When considering the implementation of Reinforcement Learning, it's important to determine what will make up the environment. this includes the actions, states, rewards and other variable such as a discount factor and exploration probability. Using Q-Learning, I was able to implement a model free learning approach that derives an optimal policy using time difference learning to constantly update the Q table values on each iteration.

2.1 Choosing Actions

In the game Flappy Bird, the bird is locked in x-space but has free range in the y-space. The actions capable by the bird are either jump or not jump. When the birds action is not jump, it will fall down to the ground, which loses the game. On the project website, you will have the ability to configure the type of jump that the agent can perform. The Standard jump mode is an arching jump that accelerates and decelerates, similar to the original game. The Linear jump mode has two variables that represent static jumping and falling velocities. In this project, the bird which represents the agent, must choose what states require a jump and which ones don't.

2.2 Determining Environment States

In order for the agent to know its position in the environment, the agent must always be located in a state (s) which is a member of a discrete number of states in a set (S). States can be represented in a variety of ways including positioning in an environment, location relationships of obstacles, or even a combination of the two. In my development, I decided to represent the states by utilize the characteristics of the moving obstacles within the environment. These obstacles are referred to as the top bar, represented as TB and the bottom bar represented as BB, where TB is on the ceiling while BB is positioned directly below but on the floor.

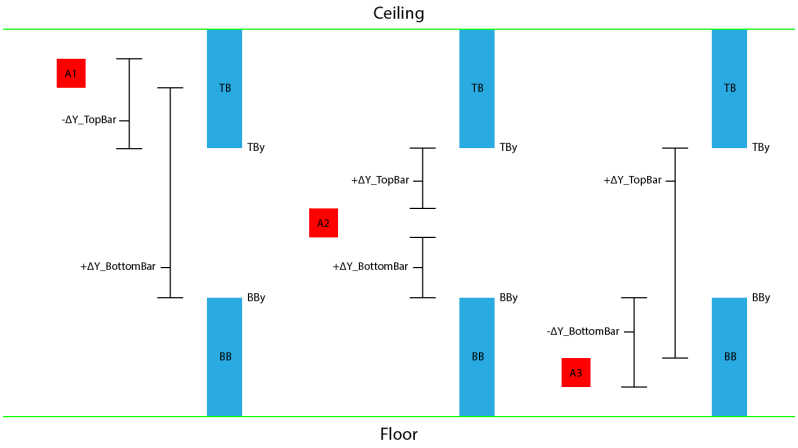

Figure 1: Vertical Differencest

2.3 Vertical difference between agent height and obstacles

When considering the states, rather than only using a single variable to represent the current state of the agent, the method I developed utilizes two variables. The first variable is derived from the vertical difference between top of the agent's height and the vertical position of TB's lowest part (TBy), whereas the second variable is derived from the vertical difference between the bottom of the agent's height and the vertical position of BB's highest part (BBy).

Represented in figure 1, the agent is shown in three different positions, A1, A2, and A3. In position A1, the vertical difference between the top of the agent to the TBy scales negatively where as the vertical difference between the bottom of the agent to the BBy scales positively. With position A3 the vertical differences are the opposite of A1's with the vertical difference between the top of the agent to the TBy scales positively where as the vertical difference between the bottom of the agent to the BBy scales negatively. Last the position of A2 shows both of the vertical differences remaining positive.

2.4 Mapping to state indexes

In order to know the true range of vertical difference from the smallest negative value to the largest positive value, it's important to know the maximum height of an obstacle as well as the minimum height. This tells us that the lowest value the state can be is the length of the tallest obstacle multiplied by negative 1 and that the greatest value the state can be is the height of the shortest obstacle subtracted from the height of the environment, in this case the game canvas. This is visually represented in figure 2. Gathering the values from the vertical differences for the purpose of representing states means that when indexing the Q table, all of the indexes must be positive. When running, these values are stored as class variables that are passed as arguments to a function that maps them to positive integers. The mapping of the values also grants the ability to choose the range from zero to the number of desired states represented in figure 3. This variable is modifiable on the project webpage.

2.5 Rewards

Rewards are necessary for telling the agent if a chosen action for a set of states is the best action to take. Utilizing the vertical difference algorithm to do this reduces the need for additional processes and is overall more efficient. Because the objective of the agent is to make it through the gap between the obstacles, veering away from this goal zone must issue a penalty or negative reward. The method I came up with is shown in figure 4 and mathematically represented in the equation below. Taking the absolute value of the difference between TBy and BBy and raising it to a power of some value will exponentially increase as the difference becomes larger. When multiplied by negative one, the reward value decreases exponentially by the power it was raised too. With an exponentially decreasing

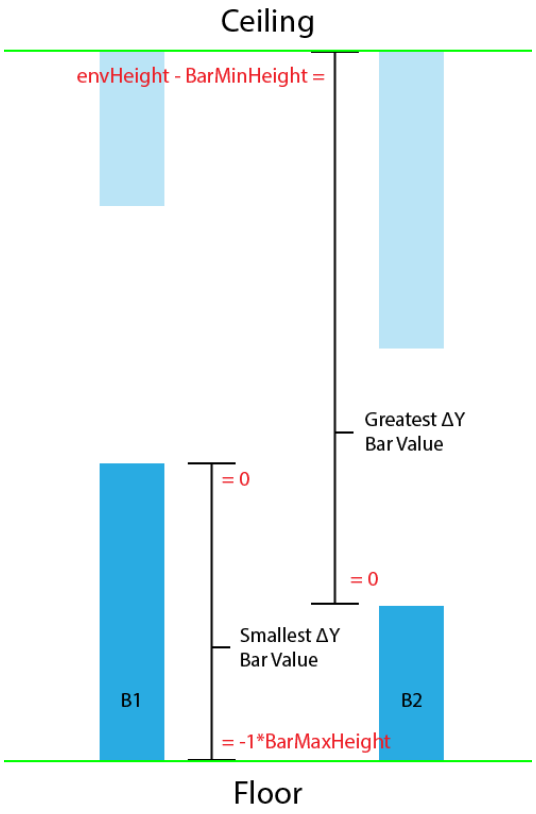

Figure 2: Largest range of vertical distance values

reward, the further the agent moves from the goal zone, the larger the penalty will be.

$$Reward = (|TBy - BBy|^{x}) * -1 \tag{1}$$

2.6 Results

The results showed success with all tests and is fully working to try out in the link on the first page.

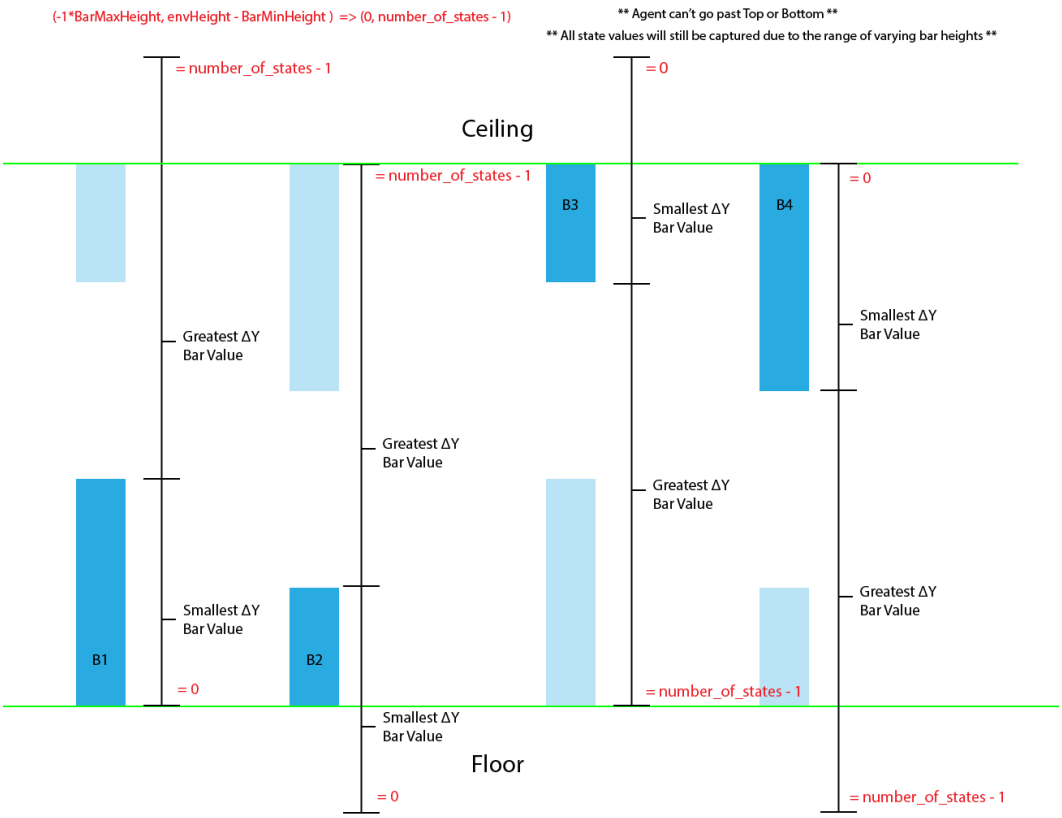

Figure 3: Largest range of vertical distance values mapped

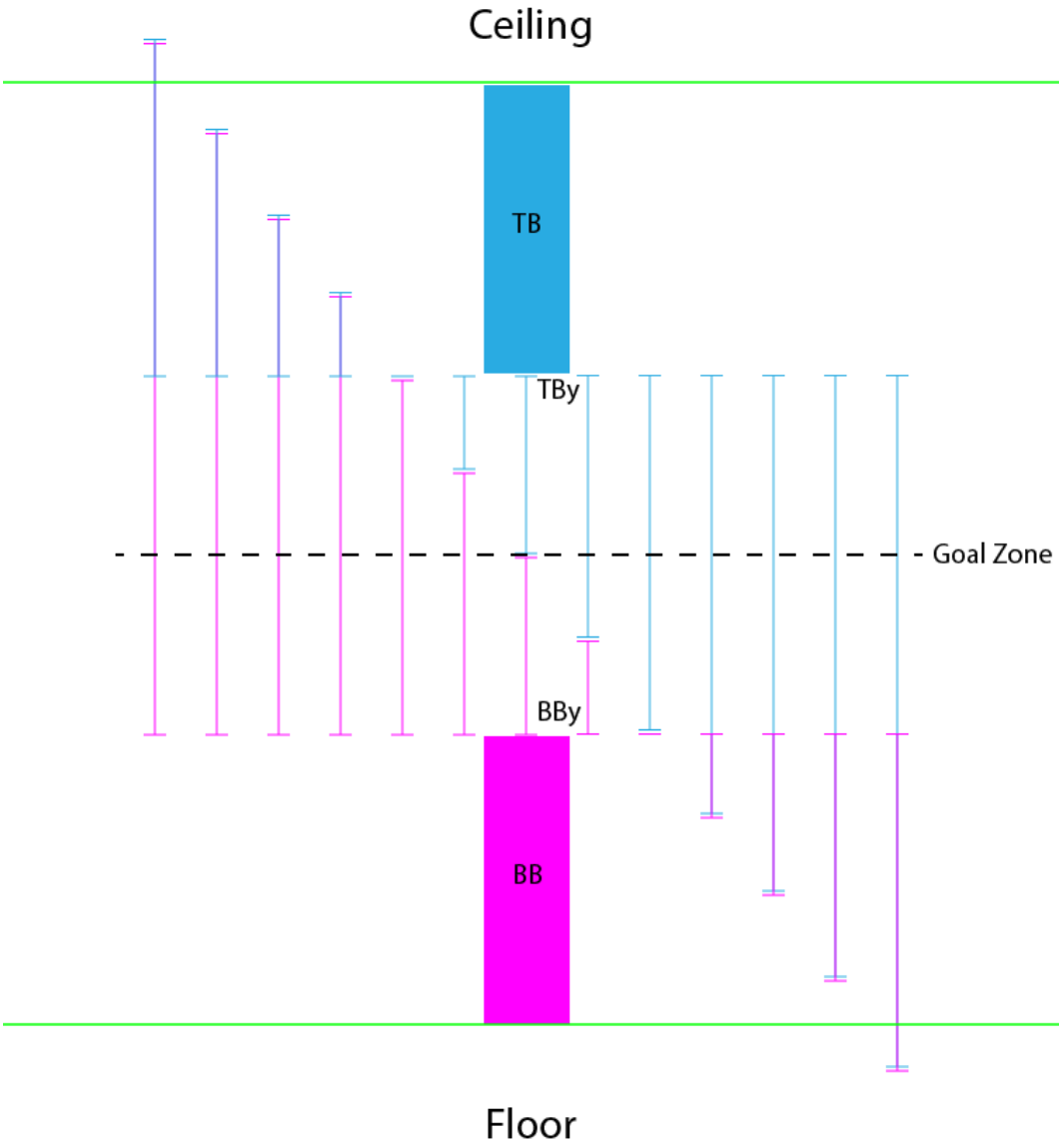

Figure 4: Example of reward zones