CS440/ECE448 Fall 2016

Artificial Intelligence
Assignment 4

Work Distribution:
Part 1: Bangqi Wang
Part 2: Yifei Li
2nd Dec. 2016

Part 1: Q-Learning (Pong)

-Bangqi Wang

This part defines the Markov Decision Tree for Pong Game and uses Q-Learning to create agent to play the Pong.

Markov Decision Tree

The Markov Devision Tree uses states, actions, rewards, initial state, and termination to define the decision process.

State for Pong is defined by the tuple (ball_x, ball_y, velocity_x, velocity_y, paddle_y). It is impossible to define the Markov Decision Tree for continuous moving, so this project divides the x and y domain into 12 parts. The total game board will be divided into 12*12 = 144 different areas. The velocity_x has two directions and velocity_y has three directions. The paddle can move in three directions. The total number of states is 12*12*2*3*3 = 10,369.

Actions for the paddle can only move in y direction and be chosen from moving up, moving down, or stay. Each state can have 3 different actions.

Rewards only have two cases: -1 for missing the ball, and +1 for rebounding the ball.

Initial state for Pong is to set the ball in the center of the screen and the paddle in the middle of the right boundary.

Terminating the game when the paddle fails to rebound the ball.

Q-Learning

Q-Learning is one of the reinforcement learning models that could be used to find the optimal action-selection policy for any given Markov Decision Tree. The Q-Learning follow the algorithm below: (http://mnemstudio.org/path-finding-q-learning-tutorial.htm)

Algorithm

1. Set the alpha, gamma parameter, and environment rewards.
2. Initialize Q value matrix with dimension states * actions.

$$Q : S \times A \to \mathbb{R}$$

3. For each episode (game):
    a. Select possible action and calculate the Q value for states reached.
    b. Find the max Q value from reached states.
    c. Update the Q value for current state with equation below.
    d. Set current state as next state.
    e.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

(From https://en.wikipedia.org/wiki/Q-learning)

Parameter

Alpha is the learning rate for Q-Learning. The learning rate defines the weight ratio between old knowledge and new knowledge. The agent will learn nothing if the factor if 0. The factor of 1 will give balance evenly to old and new knowledge.

Gama is the discount factor that determines the importance of future reward. The agent with less Gama factor will become more myopic. The agent has factor of 0 will only focus on current situation.

Epsilon is the factor describes the "spirit of adventure". The agent with higher epsilon are more willing to take risk action instead of following the Q value matrix.

Part 1.1: Interpretation of Parameter

The parameters in this part directly affect the performance of the agent. In order to find the best performance, I need to define the alpha, gamma, and epsilon parameters one by one.

Gamma: discount factor

This parameter can be defined first. The agent with higher Gamma factor will become wiser in long term. In this case, it is reasonable to make the gamma factor as large as possible. However, if the gamma factor larger than 1, the Q value in matrix may diverge. Therefore, the program needs to find the largest gamma factor in the range from 0 to 1. According to the document:

- For choosing a discount factor, try to make sure that the reward from the previous rebound has a limited effect on the next rebound. In other words, choose your discount factor so that as the ball is approaching your paddle again, the reward from the previous hit has been mostly discounted.

The program needs to choose the largest gamma value that have little influence for the next approaching. The Pong game has 12 states in x dimension and 12 states in y dimension. Therefore, the ball at least changed the state 12*2 = 24 times before coming back again.

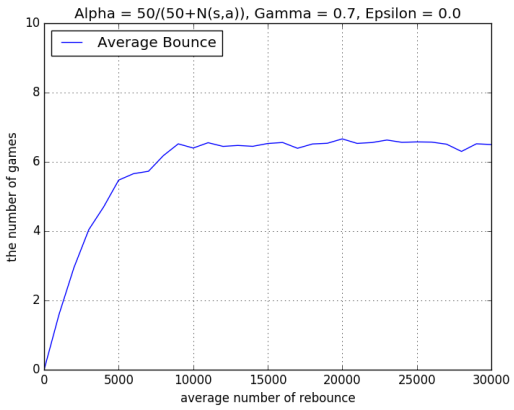
Gamma	Returned Value	
0.9	$0.9^{24} = 0.0798 = 8\%$	
0.8	$0.8^{24} = 0.00472 = 0.5\%$	
0.75	$0.75^{24} = 0.001003 = 0.1\%$	
0.7	$0.7^{24} = 0.00019 = 0.02\%$	√
0.65	$0.65^{24} = 0.000032 = 0.003\%$	

The return value for 0.7 is about 0.02%, which is small enough to be ignored. Therefore, the program will choose **gamma = 0.7**.

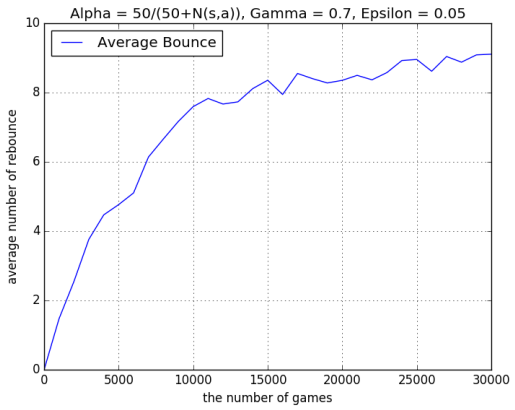
Epsilon: exploration rate

After decided the gamma factor, the program can define the exploration before alpha factor. The epsilon factor stands for the exploration rate. The agent with higher epsilon factor are more likely to explore new action instead of following the Q value matrix. The agent with higher epsilon are more unstable. They may progress or regress significantly. However, if the agent has epsilon factor too low, the agent will have less change to make progress.

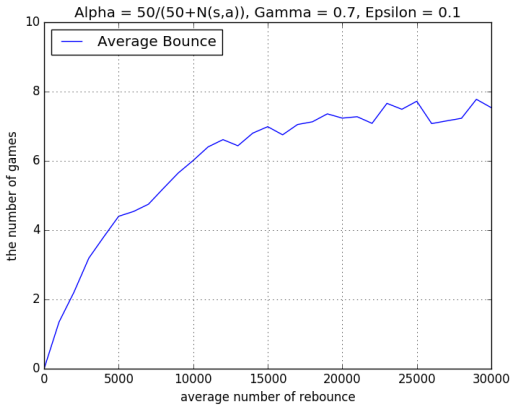
Epsilon in short running (30K)



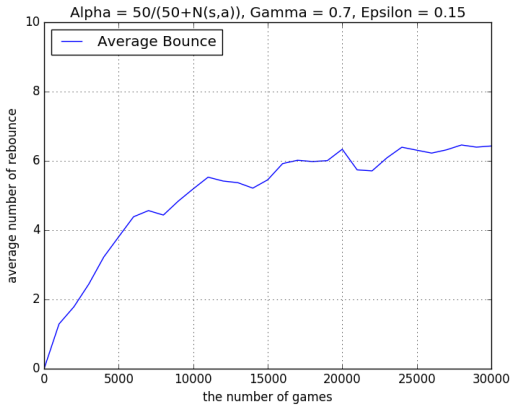
Epsilon = 0.0



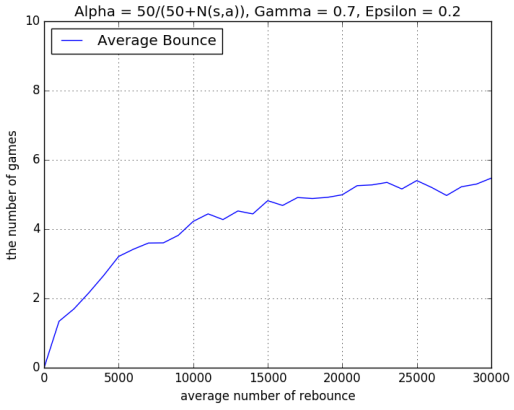
Epsilon = 0.05



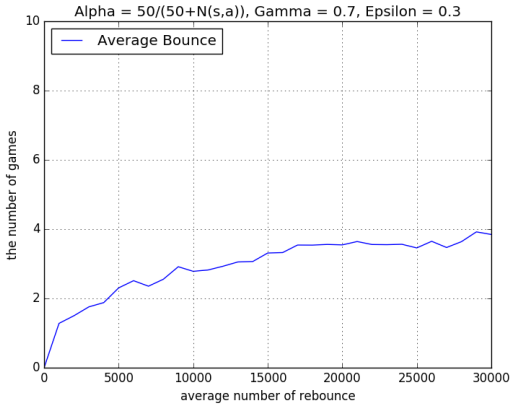
Epsilon = 0.1



Epsilon = 0.15



Epsilon = 0.2



Epsilon = 0.3

The agent with higher epsilon will improve slowly but they may have more breakthrough in long term. According to the table above, the agent with lower epsilon will increase faster but have less opportunity to breakthrough. The agent with epsilon 0.05 has 9 rebounds in 30K games. Therefore, we will choose **epsilon = 0.05**.

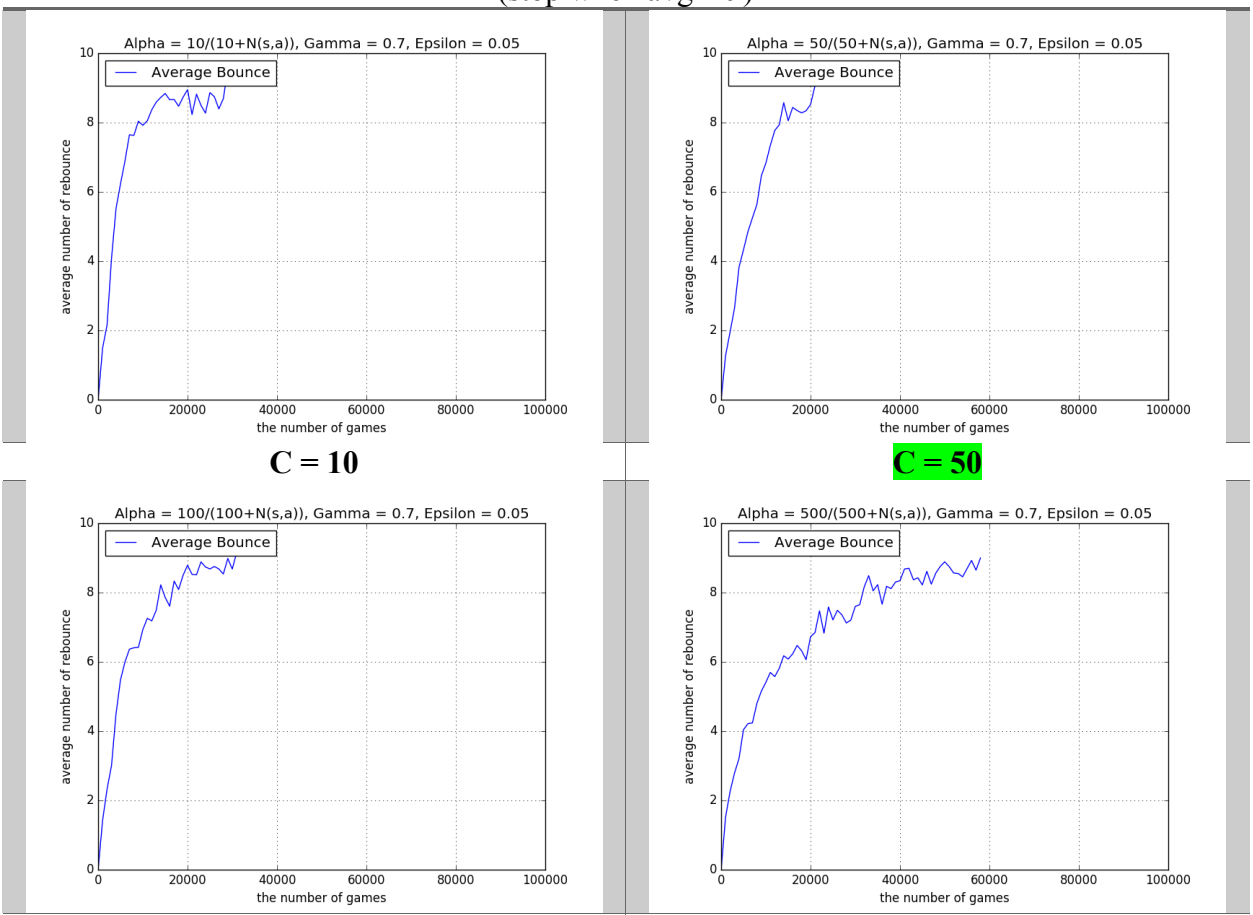
Alpha: learning rate

After determining all other factors, the program needs to determine the learning rate. For the exploration, the program needs to require that each state action be performed infinitely often. The program needs to modify its estimation alongside the learning process. The learning rate controls how fast the program modify its estimation. The agent with higher learning rate will decay faster but has less change to learn and improve.

Learning rate in long running (100K)

Alpha = C / (C + N(s,a))

(stop when avg > 9)



C = 10 | **C = 50**

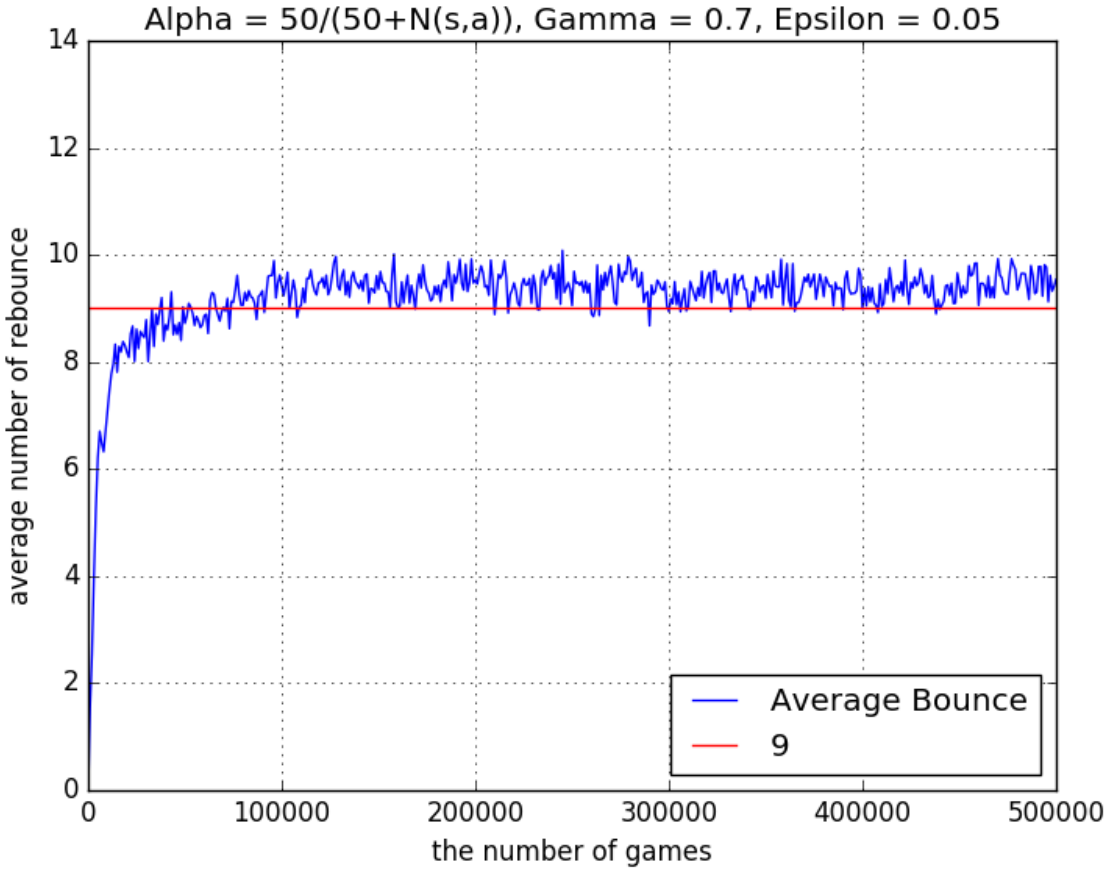
C = 100 | **C = 500**

The learning rate will decay to half after saw the state C times. The total running for optimal solution is about 100K and the total states is about 30K. According to my observation, the number of states is around 20k, because some states is extremely hard to reach. Therefore, each states will show about 5 times in the training process. However, states have different possibility to appear. According to the table above, I believe that C = 50 is the most reasonable factor,

because agent with alpha = 50 / (50 + N(s,a)) reaches optimal (average number of rebound is 9) fastest. Therefore, I will choose **C = 50**, and **alpha = 50 / (50 + N(s,a)).**

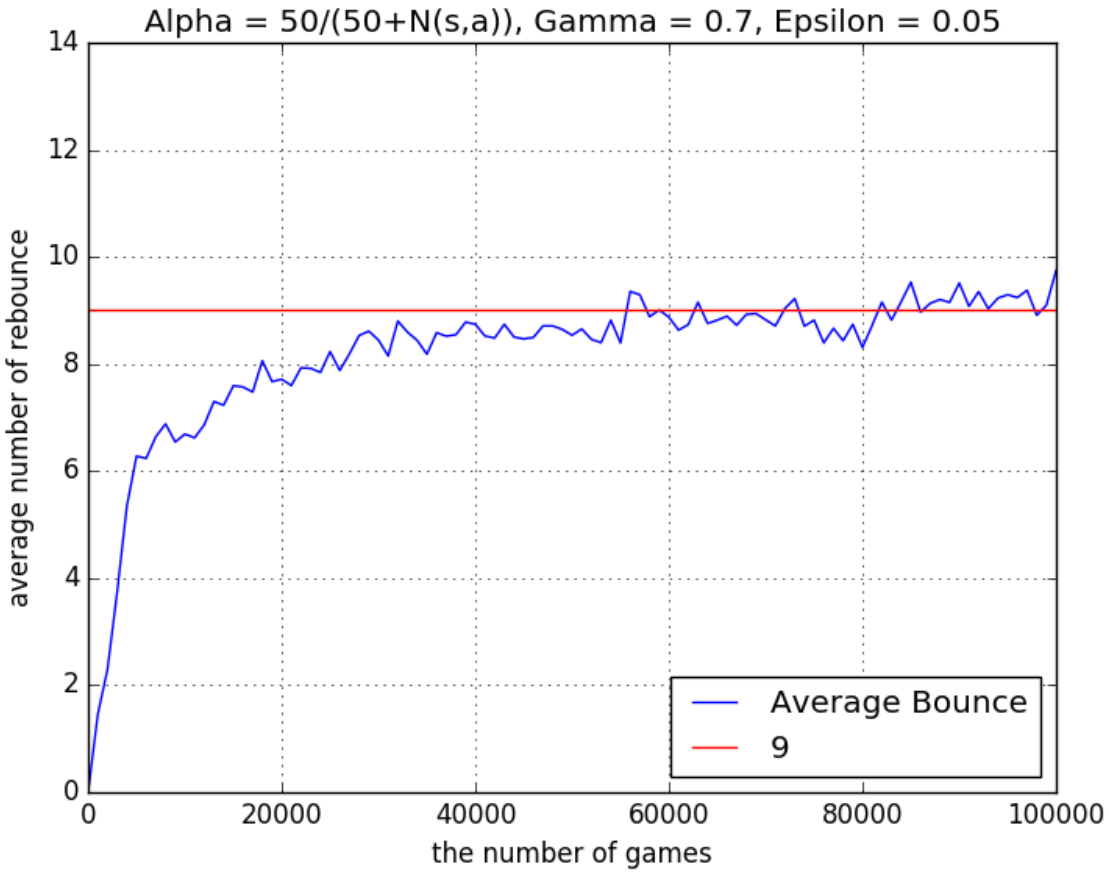
Long term running

I will run the program with C = 50, gamma = 0.7, and epsilon = 0.05 for 500K times. The performance reaches optimal in about 100K. The y dimension shows the average number of rebound for most recent 1000 games. The average is in the range of 9 to 10.



MDP Modification

The game board is divided into 12 parts in x dimension and 12 parts in y dimension. Each part has length of 0.084 and velocity_x is larger than 0.03. Therefore, the time for passing one part is at most 2.8 seconds. If the ball has same y value with the middle of paddle. In this case, the paddle does not need to move unless the ball will go beyond the range of half of the paddle height in 2.8 seconds. Therefore, the velocity_y should smaller than 0.036. In this part, I changed the MDP states. The velocity_y is 1 when velocity_y is larger than 0.03, -1 when velocity_y is smaller than -0.03, and 0 in between. As the graph shown below, the performance has similar average bounce but it is more stable.



Part 1.2: Modification

The program in this part has one more agent in the left side.

Modification for MDP

1. State (ball_x, ball_y, velocity_x, velocity_y, paddle1_y, paddel2_y)
2. Rewards:
    a. Lose = -1
    b. Win = 1
    c. Hit = 0.1 (the expectation of rebound is 9 to 10 times with wall)

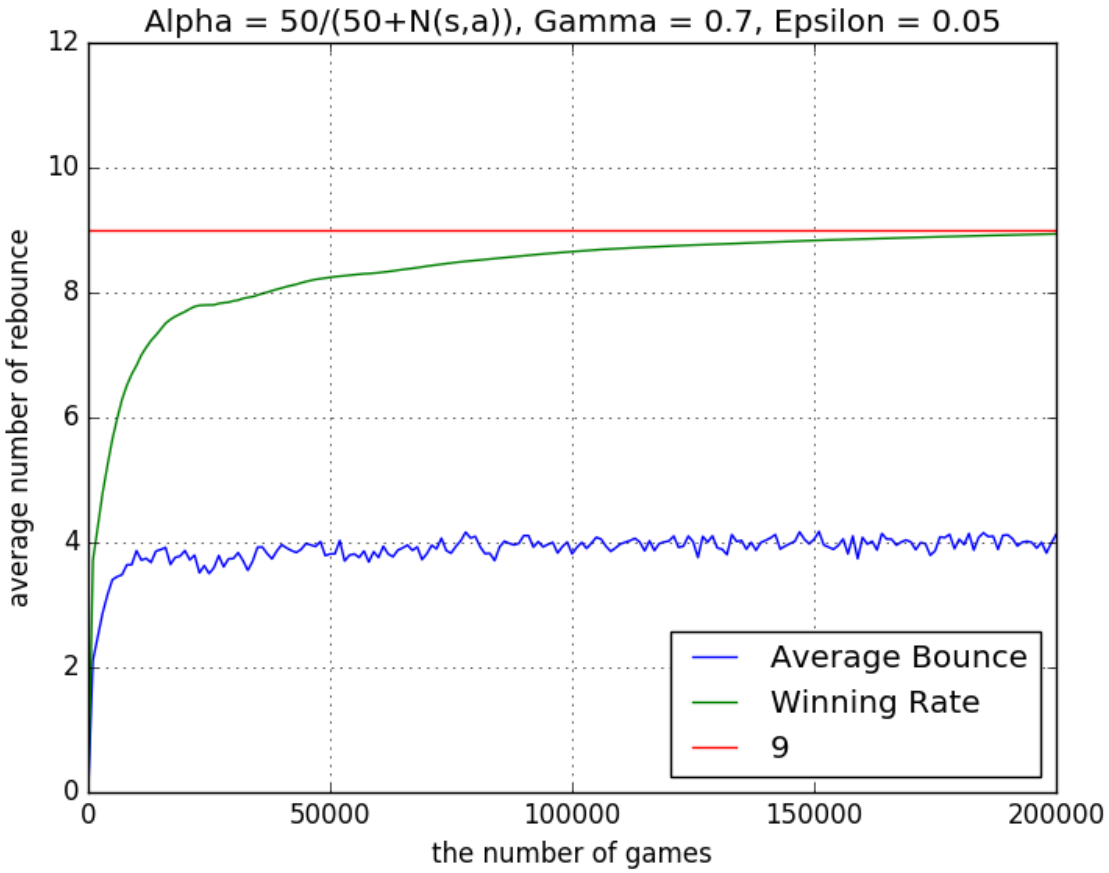
Add one more parameter in state so the side effect is the increase of training times. Modify the reward function so that the win has highest rewards and 10 hits have the same reward.

Modification for Code

Add one more paddle. Add the check function, rebound function, and moving function for this paddle. Add the drawing function for animation.

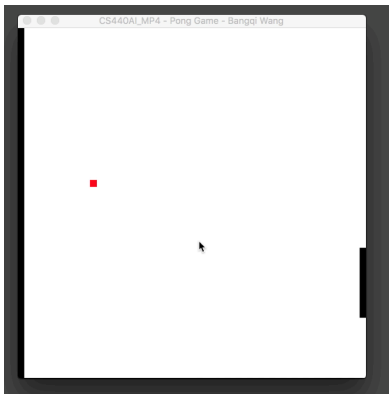
Result

The graph below shows result of this part. The green line represents the number of win out of 10 games. The blue line represents the average number of rebound. The winning rate is near 90%

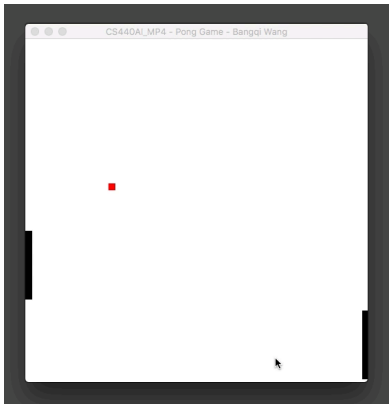


Extra Credit

Video for part1 and part2. Please see the screen recording in the video folder.



part1



part2