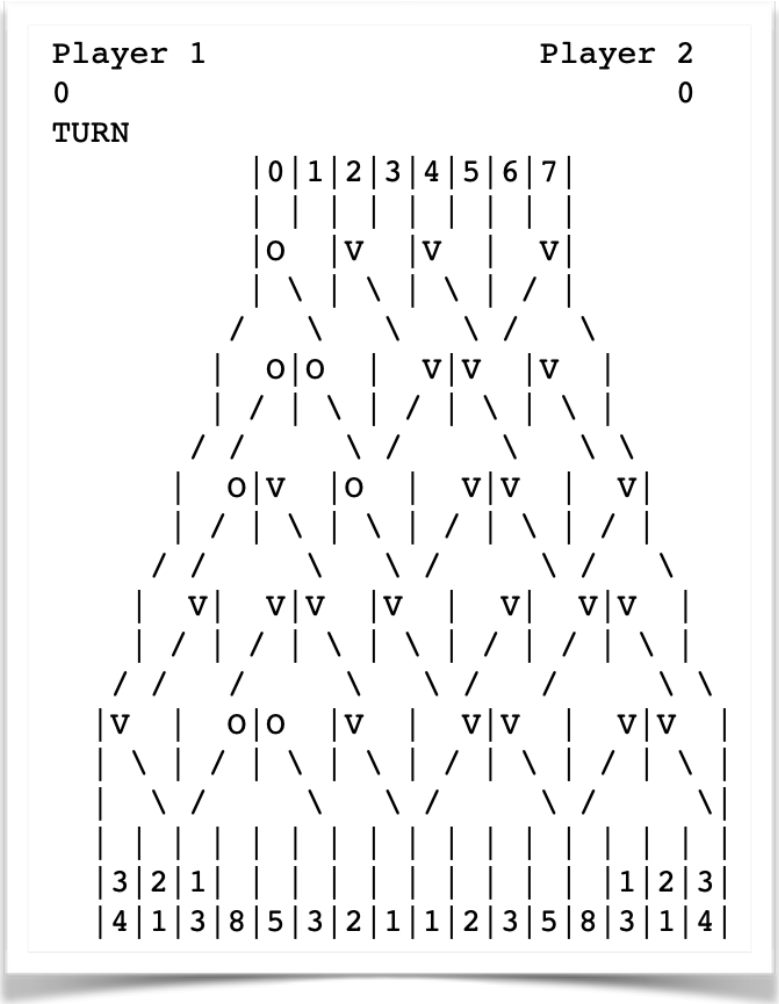
Reinforcement Learning Course Project Summary Report

of MSDS-603, taught by Brian Spiering

Ömer Sakarya - June 27, 2019

Framework & Gameplay

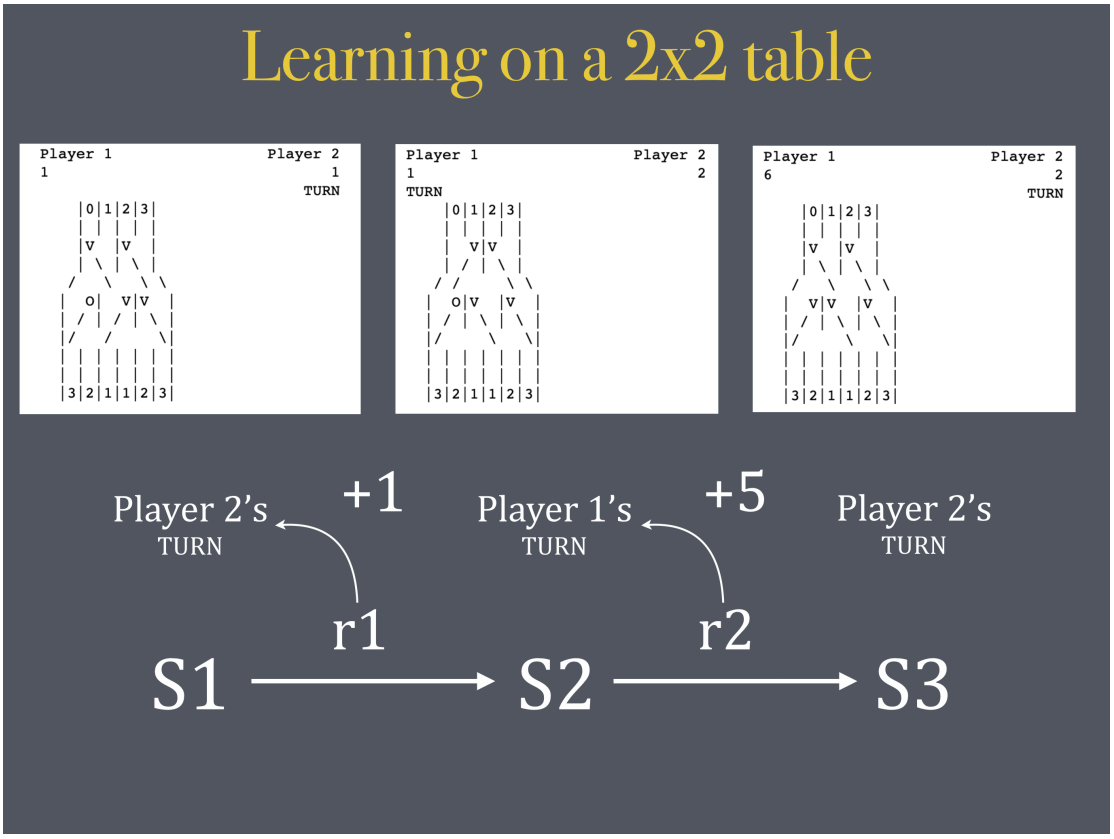
This summary report of my attempt is on teaching a computer program to play a game called Treasure Drop using reinforcement learning methods. There is a screenshot at the cover page. I have also put a link to a sample gameplay video in the appendix. The game consists of four rounds. In each round, scores at the very bottom change. Each round ends when one of the players hit a certain score. The game ends when the fourth round ends. In the cover, there is a version with holes. However, I have simplified this game for the agent and removed the holes. Also, I have made the game consisting of only one round, with score buckets consisting of fibonacci scores. Below, is a screenshot of the output of the program. More specifically, printState() method of TreasureDrop class.



Each lever can have four states. They can be to the left or to the right. They can have a coin or they might not. Therefore, number of states the game can be in is four to the power of number of levers. In this case, number of levers is thirty. Therefore, number of states this game can be in is 4^{30}. Players take action by dropping coins from one of the slots above. In this set-up, there are 8 possible holes for coins. Players score by dropping coins all the way to the bottom. Score gained from a drop is equal to the number associated with a bucket.

Deep Q Learning

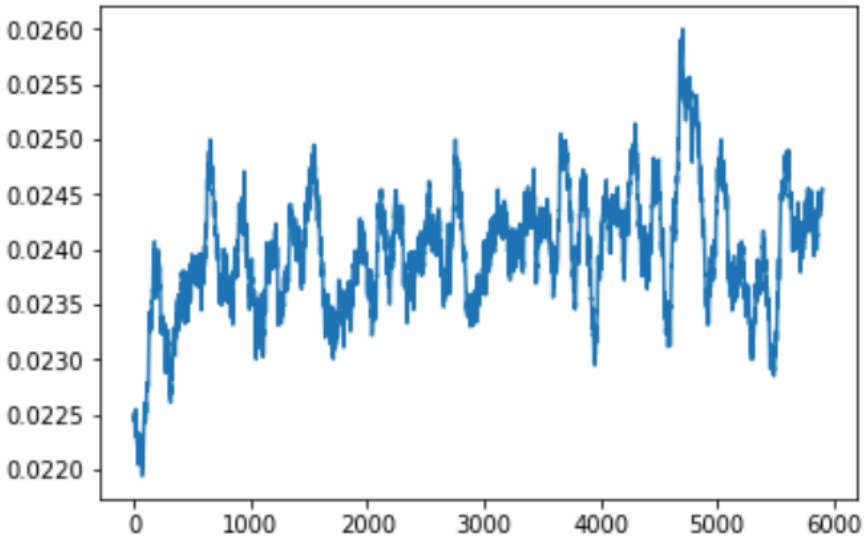
For simplicity and the sake of starting small, I have started playing with a 2x2 table. Before starting to code, one thing about this game caught my attention. Consider the case below. Player 2 makes a move, gains 1 points. Player 1 makes a move and gains 5 points. And then it's Player 2's turn again. In this case, S2 in the figure below might look like an attractive state to be in since it has potential to bring five points. However, if the agent thinks that S2 is a desirable state and goes there, it's the other player's turn and the other player gets the point. Because of that, S3 is really the next state for Player 2 because it is the next state when Player 2 gets hold of the game.



In order to model that effect, I have came up with the formula below. It's a slightly modified version of Q learning formula. Notice the term $-r_2$ right after r_1. The idea here is that, whatever point the opponent makes is my loss since this a two-player, zero sum game. Therefore, if I gain r_1 and lose $-r_2$ -, what I really gain is $r_1 - r_2$. Also, if s_3 is the next state I am being in, the discounted sum of future rewards should be with s_3. Hence, the formula below.

$$Q(s_1, a_1) \leftarrow Q(s_1, a_1) + \alpha(r_1 - r_2 + \gamma max_a Q(s_3, a) - Q(s_1, a_1))$$

4^{30} is a big number for a tabular Q learning method. Therefore, I have attempted to use deep q-learning for this task. I have implemented experience replay and had the training loop sample (s1, a1, r1, s2, a2, r2, s3) pairs from the experience replay object and train a neural network using the formula above. I will not beat around the bush and jump directly to the sad part. It did not converge.




```
plt.plot(moving_aves)
```

However, I have observed a strange behavior. Every once in a while during training, I have faced the agent off with a random agent. Random agent is just agent's opponent dropping coins in a random manner. I have noticed that agent was consistently losing against the random agent. In the figure below, I have captured the smart and random agent's average scores and win ratios. I was not able to figure out why this was happening due to shortage of time.

```
Epoch: 0 Model's win ratio: 0.32 Model's avg score: 82.59 Random Agent's avg score: 95.29
Epoch: 100 Model's win ratio: 0.63 Model's avg score: 95.4 Random Agent's avg score: 83.77
Epoch: 200 Model's win ratio: 0.91 Model's avg score: 101.29 Random Agent's avg score: 72.29
Epoch: 300 Model's win ratio: 0.43 Model's avg score: 88.92 Random Agent's avg score: 92.85
Epoch: 400 Model's win ratio: 0.05 Model's avg score: 59.41 Random Agent's avg score: 101.84
Epoch: 500 Model's win ratio: 0.17 Model's avg score: 73.94 Random Agent's avg score: 99.09
Epoch: 600 Model's win ratio: 0.07 Model's avg score: 65.76 Random Agent's avg score: 101.24
Epoch: 700 Model's win ratio: 0.05 Model's avg score: 68.63 Random Agent's avg score: 101.29
Epoch: 800 Model's win ratio: 0.28 Model's avg score: 82.08 Random Agent's avg score: 96.39
Epoch: 900 Model's win ratio: 0.05 Model's avg score: 63.85 Random Agent's avg score: 101.2
Epoch: 1000 Model's win ratio: 0.09 Model's avg score: 68.43 Random Agent's avg score: 100.95
Epoch: 1100 Model's win ratio: 0.21 Model's avg score: 76.62 Random Agent's avg score: 98.41
Epoch: 1200 Model's win ratio: 0.03 Model's avg score: 63.03 Random Agent's avg score: 101.76
Epoch: 1300 Model's win ratio: 0.02 Model's avg score: 62.67 Random Agent's avg score: 101.57
Epoch: 1400 Model's win ratio: 0.03 Model's avg score: 64.7 Random Agent's avg score: 101.91
Epoch: 1500 Model's win ratio: 0.2 Model's avg score: 79.3 Random Agent's avg score: 98.76
Epoch: 1600 Model's win ratio: 0.17 Model's avg score: 81.03 Random Agent's avg score: 98.67
Epoch: 1700 Model's win ratio: 0.02 Model's avg score: 64.0 Random Agent's avg score: 101.94
Epoch: 1800 Model's win ratio: 0.16 Model's avg score: 79.05 Random Agent's avg score: 98.43
Epoch: 1900 Model's win ratio: 0.05 Model's avg score: 65.85 Random Agent's avg score: 100.87
Epoch: 2000 Model's win ratio: 0.07 Model's avg score: 68.32 Random Agent's avg score: 101.05
Epoch: 2100 Model's win ratio: 0.19 Model's avg score: 78.64 Random Agent's avg score: 99.45
Epoch: 2200 Model's win ratio: 0.17 Model's avg score: 78.09 Random Agent's avg score: 99.78
Epoch: 2300 Model's win ratio: 0.21 Model's avg score: 79.05 Random Agent's avg score: 98.37
Epoch: 2400 Model's win ratio: 0.23 Model's avg score: 79.34 Random Agent's avg score: 97.87
Epoch: 2500 Model's win ratio: 0.05 Model's avg score: 61.49 Random Agent's avg score: 101.36
Epoch: 2600 Model's win ratio: 0.03 Model's avg score: 64.71 Random Agent's avg score: 101.25
Epoch: 2700 Model's win ratio: 0.02 Model's avg score: 63.58 Random Agent's avg score: 101.85
Epoch: 2800 Model's win ratio: 0.19 Model's avg score: 79.46 Random Agent's avg score: 97.98
Epoch: 2900 Model's win ratio: 0.05 Model's avg score: 67.49 Random Agent's avg score: 101.52
Epoch: 3000 Model's win ratio: 0.01 Model's avg score: 63.66 Random Agent's avg score: 102.04
Epoch: 3100 Model's win ratio: 0.26 Model's avg score: 79.95 Random Agent's avg score: 96.37
Epoch: 3200 Model's win ratio: 0.04 Model's avg score: 67.2 Random Agent's avg score: 101.5
Epoch: 3300 Model's win ratio: 0.06 Model's avg score: 65.14 Random Agent's avg score: 100.84
Epoch: 3400 Model's win ratio: 0.22 Model's avg score: 77.34 Random Agent's avg score: 99.6
Epoch: 3500 Model's win ratio: 0.04 Model's avg score: 61.65 Random Agent's avg score: 101.29
Epoch: 3600 Model's win ratio: 0.22 Model's avg score: 75.98 Random Agent's avg score: 98.31
Epoch: 3700 Model's win ratio: 0.06 Model's avg score: 66.35 Random Agent's avg score: 101.14
Epoch: 3800 Model's win ratio: 0.07 Model's avg score: 66.32 Random Agent's avg score: 100.43
Epoch: 3900 Model's win ratio: 0.28 Model's avg score: 83.09 Random Agent's avg score: 97.5
Epoch: 4000 Model's win ratio: 0.04 Model's avg score: 64.48 Random Agent's avg score: 101.47
Epoch: 4100 Model's win ratio: 0.02 Model's avg score: 60.1 Random Agent's avg score: 101.84
Epoch: 4200 Model's win ratio: 0.15 Model's avg score: 77.1 Random Agent's avg score: 99.23
Epoch: 4300 Model's win ratio: 0.25 Model's avg score: 79.64 Random Agent's avg score: 96.97
Epoch: 4400 Model's win ratio: 0.03 Model's avg score: 64.58 Random Agent's avg score: 101.36
Epoch: 4500 Model's win ratio: 0.14 Model's avg score: 78.49 Random Agent's avg score: 99.42
Epoch: 4600 Model's win ratio: 0.2 Model's avg score: 77.61 Random Agent's avg score: 99.23
Epoch: 4700 Model's win ratio: 0.03 Model's avg score: 61.57 Random Agent's avg score: 101.95
Epoch: 4800 Model's win ratio: 0.04 Model's avg score: 66.68 Random Agent's avg score: 101.44
Epoch: 4900 Model's win ratio: 0.03 Model's avg score: 61.44 Random Agent's avg score: 101.61
```

Lessons Learned

Deep q learning has reputation of not converging since it's a quite new field. I was not able to figure out what exactly was going on. More knowledge of neural networks would be really helpful in my case.

Appendix

A video of sample play: https://www.youtube.com/watch?v=IhHBkuKkYJs