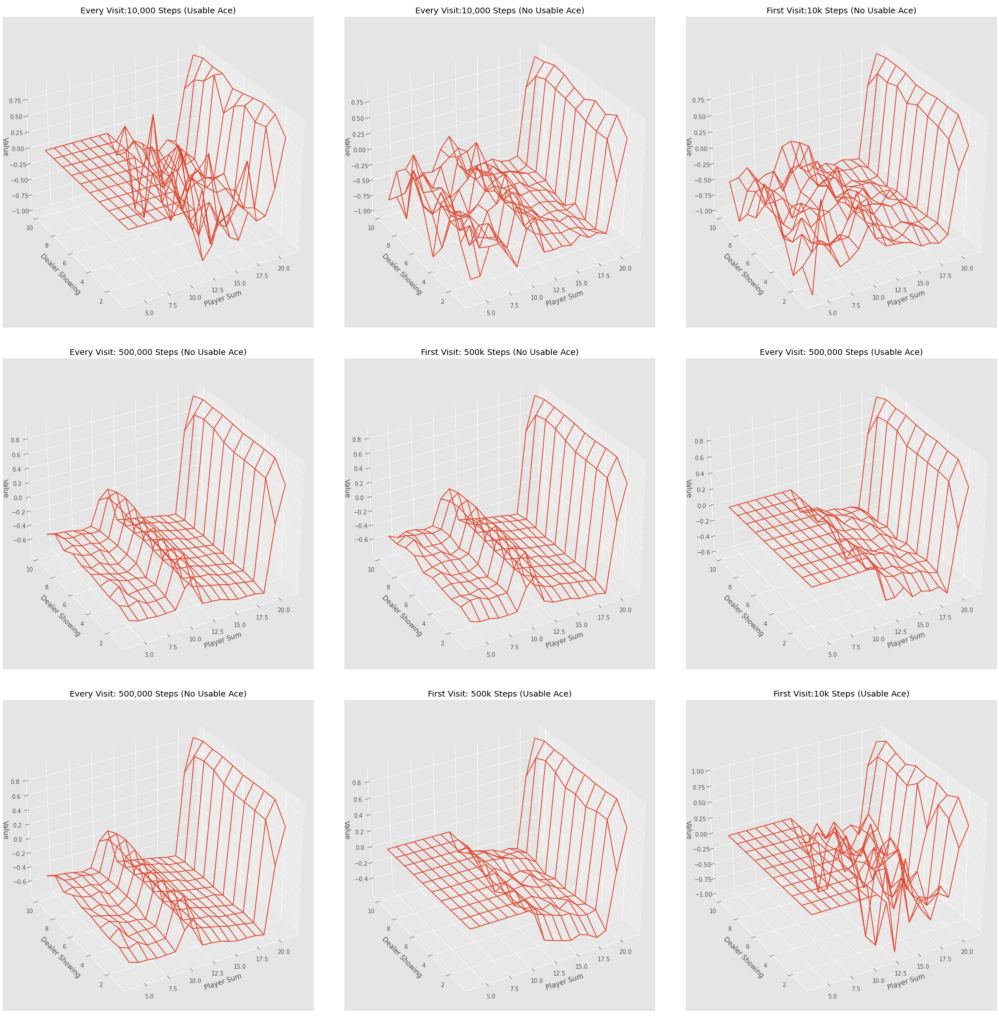
CS427:Reinforcement Learning **(Date: 28-Mar-2022)**

Assignment 4

Instructor: Dr. Prabuchandran K J *Rollno.:* 211022005, 211022001

1 Monte-Carlo Prediction and Control

Prediction: Considered Blackjack setting with a policy μ of "HIT" till player sum is 20 or more, otherwise "STICK". For the Blackjack setting Markov Decision Process (MDP) is formulated by state representing (player's sum, dealer's face card, ace or not) and actions are to "HIT" or "STICK". The prediction algorithm finds value V_μ of each state following a policy μ. The prediction results using first visit and every-visit are as follows.



Control: Using Monte-carlo control the estimated value function V^* is shown below for cases with usable and no Ace.

Mountain Car Problem In this problem the goal is to move a low power car up the hill by gaining momentum via back and forth motion. The state space of car is made of position and velocity. Position is bounded in range [-1.2,0.5] while velocity is bounded in [-0.07,0.07]. The position and velocity are updated by

$$v(t+1) = bound[v(t) + 0.001a(t) - 0.0025\cos 3x(t)]$$

$$x(t+1) = bound[x(t) + v(t+1)]$$

Actions are defined to be "accelerate left", "accelerate right" and "Don't accelerate". The reward for not being at flag is -1, while being at flag is not penalised. The state at which we reach the flag-post is the terminal state.
As the positions and velocity are continuous we have to discretize it which could be done by considering a window over which position and state would be fixed.

Acrobot Problem The acrobot system includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height.
Actions: $\pm$1torque or 0 torque. **States:** 6 states(4 refer to $sin()$ and $cos()$ of angle made at two joints and 2 for angular velocities at two angles.) **Rewards:** -1 for not reaching ceiling.

SARSA Envs: MountainCar, Cartpole, Acrobot In order to solve mountain car problem using SARSA the states were discretized by dividing play environment into windows of size 30. Following this in each window velocity, position were fixed to a constant value. The code snippet is as follows

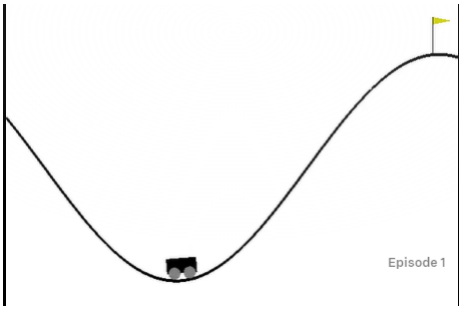
```
def discretization(state):
    resolution = (env.observation_space.high-env.observation_space.low)/30
    #calculate what bin a state should go after resolution
    discrete_state  = (state-env.observation_space.low)//resolution
    return tuple(discrete_state.astype(int))
```

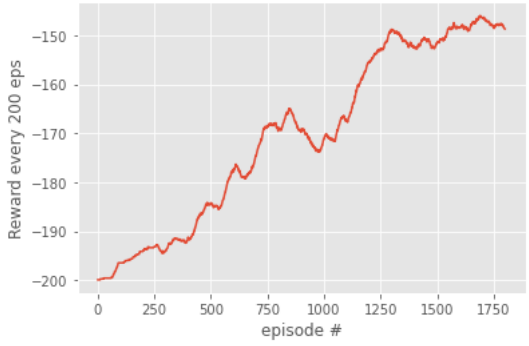
Q(s,a) is update according to SARSA-Onpolicy given by,

$$Q(s,a) = Q(s,a) + \alpha[R + \gamma Q(s',a') - Q(s,a)]$$

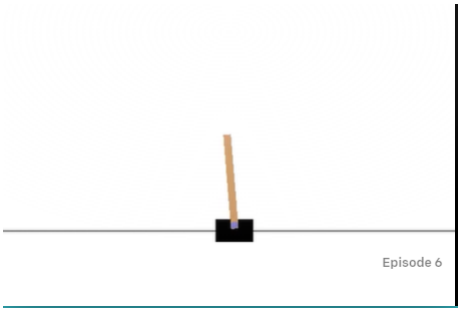
and action's are taken according to a $\epsilon-greedy$ policy. The parameters $\alpha = 0.5$, $\epsilon = 0.1$ and discount factor of unity. We saw algorithm learning within 1000 episodes to get to flag-post. **Observations** We found SARSA to be slighty faster than Montecarlo and q-learning on mountain car task. And SARSA learns from it's stochastic policy as opposed learning from best policy in Q- Learning.

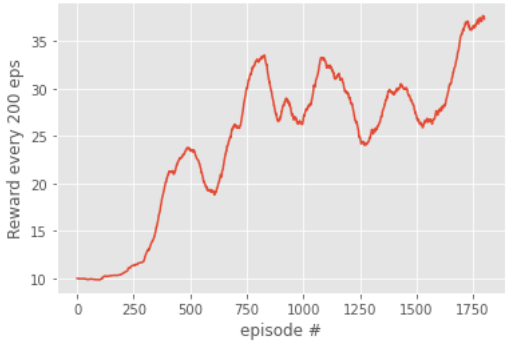
SARSA on Mountain Car



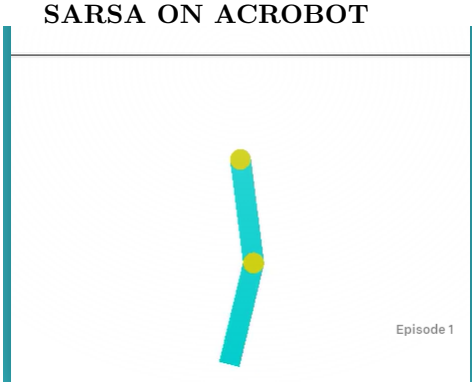


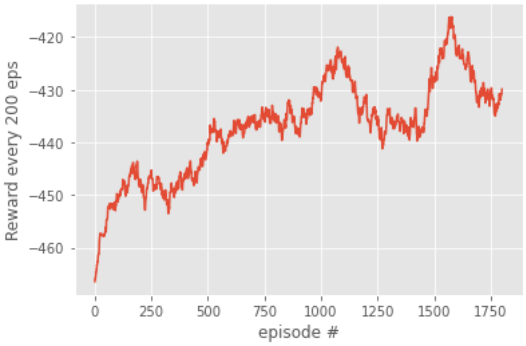
SARSA ON CARTPOLE





SARSA ON ACROBOT



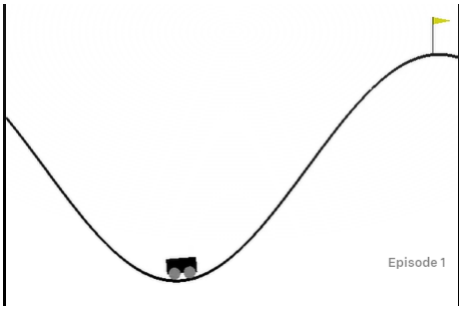


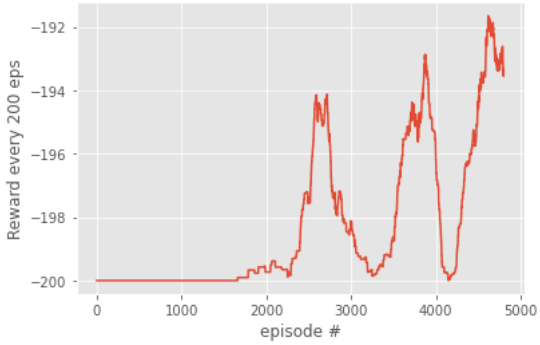
Q Learning We observed that Q-learning took more episodes to learn the mountain car problem than SARSA. The update rule of Q learning is as follows

$$Q(s,a) = Q(s,a) + \alpha[R + \gamma * max_a Q(s',a) - Q(s,a)]$$

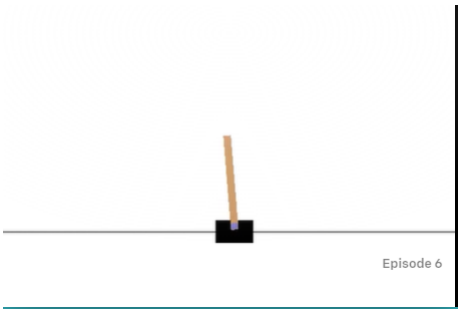
The parameters $\alpha = 0.5$, $\epsilon = 0.1$ and $\gamma = 1$. **Environments: Custom Blob, Mountain Car, Cartpole** In Custom Blob Environment, there can be minimum 3 blobs (food, enemy and player). Player has to move to the food without crashing with enemy. We also took a case where all 3 blobs are moving and saw that most of the times player moved to the food without crashing with enemy.

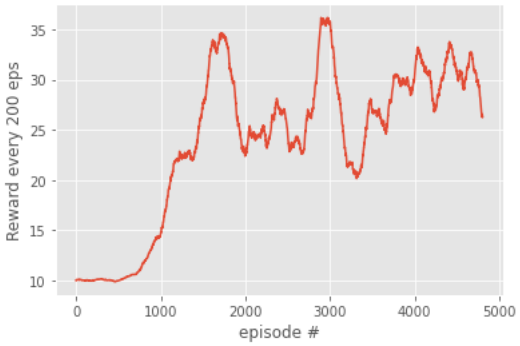
MOUNTAIN CAR Q Learning



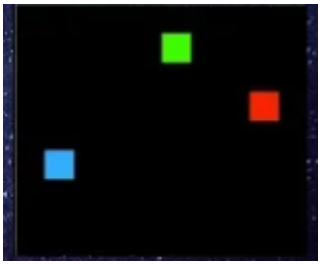


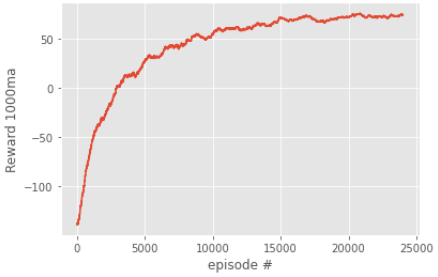
CARTPOLE Q LEARN





Blob Environment Q Learning





This problem is based on
`https://pythonprogramming.net/own-environment-q-learning-reinforcement-learning-python-tutorial/`.
Parameters in Blob Environment Player is "BLUE", Food is "GREEN" while Enemy is "RED"

- Food Reward=100
- Move Penalty=-1
- Enemy Penalty=-300
- $\epsilon = 0.9$, decayed by 0.998 per episode
- Learning rate=0.1
- Discount=0.95

We found it useful if we want to create our own environment using image processing toolbox opencv in python.
Observations
For this custom blob environment we tried to put more number of blobs of each class but with

Q-Learning memory requirements grow exponentially as state space grows. This could be extended to DQN which saves memory by using Function Approximation.

Monte Carlo On-policy: Environments: BlackJack, Mountain Car We observed that montecarlo was not performing well for Mountain Car unless a positive reward was given when car moves uphill. This however is not a complete solution and further exploration is required.