Pacman Environment

In my implementation the key idea is to keep the state space small and use the minimal crucial information required for decision making as the representation of a state. Pacman may just need to know about status of immediate next cells i.e. Left, Right, Up, Down of the pacman cell. Assuming a state to be the status of cells around the pacman i.e. what is on the left, right, up, down of pacman.

Status of a cell can be either Danger(0), Safe(1) or Food(2). Danger represents the ghost cell or boundary cell, Safe represents an empty cell and Food represents a cell with food. A state is a tuple of 4 values and each value can be 0,1 or 2. Hence **state space** is of size 3^4 = 81 (indexing from 0 to 80).

Possible actions for ghost and pacman are moving to left, right , upward or downward from the current cell. Hence **action space** has 4 actions L,R,U,D or 0,1,2,3.

Default **ghost policy** used:

- Randomly initialise the ghost position if it hits the boundary
- If the ghost is placed at left or the row of pacman then always move right until it hits the boundary.
- If placed on right of row always move left
- If placed at top of the column always move down
- If placed at bottom of column always move upward

Given a state and an action, **reward** on taking the action is:

- If next state is Danger, reward = -10
- If next state is Safe, reward = 0
- If next state is Food, reward = 2

Symbols in render method: “g” (Ghost), “p” (Pacman), “*” (Food cell), “-” (Empty cell)

Pacman Agent

- learning.py contains implementation of Tabular SARSA and Q Learning Algorithm
- Initializing the Q table of size 81 X 4 with random numbers.
- getPolicy() and getEpsilonGreedyPolicy() define a policy using Q Table.

Training and Evaluation

train_episodes = 5000 # Number of episodes for training
test_episodes = 100 # Number of episodes for testing
gridsize = 5 # Grid size of Pacman game
num_food = 10 # Number of food objects in Pacman game
lr = 0.4 # Learning rate(alpha)
gamma = 0.99 # Discount factor

epsilon = 0.9 # **Here epsilon is prob with which greedy action is selected** thats why its value is high
max_steps = 2500 # max steps in an episode

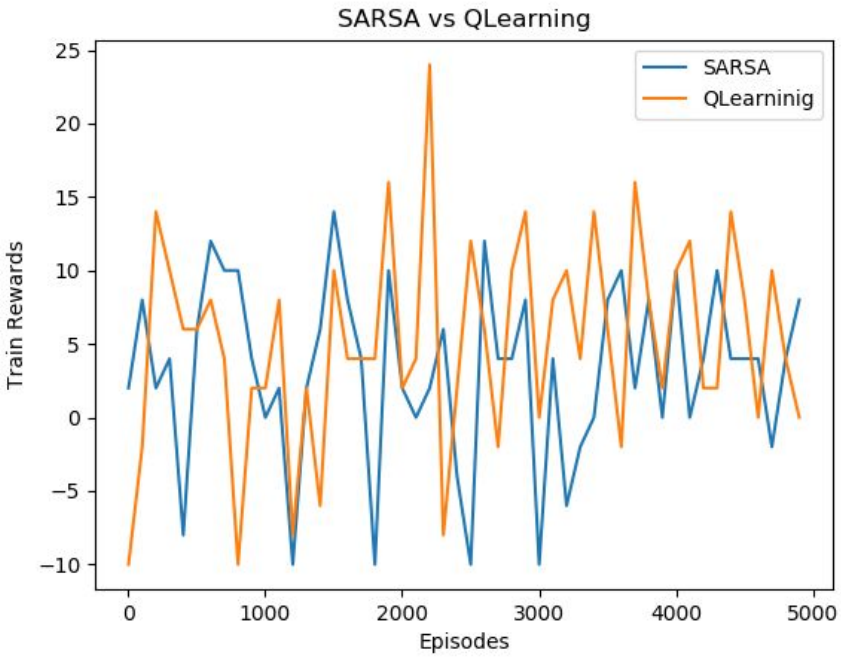
SARSA Agent Vs QLearning Agent

Average Test Rewards(SARSA Agent):7.74

Average Train Rewards(SARSA Agent):4.3288

Average Test Rewards(QLearning Agent):11.2

Average Train Rewards(QLearning Agent):5.0168



Observation

Average test reward is higher for Q Learning Agent. Q Learning agent performed better in 100 test episodes compared to SARSA Agent.

From the plot it can be seen that QLearning starts performing better and earlier than SARSA which means it converges faster and hence higher average train reward.

In pacman_output.txt one can see the game scores and pacman game rendering during the Testing. Observe that game scores of Q Learning agent are better than SARSA agent.