# Final Assignment 2315541

## Introduction

Used Kaggle Environment to play Connect X. Also known as Connect 4. Default grid size is 6X7 and 4 in a row to determine the winner. It has option  like multiple players can play.

## Environment Setup

In this assignment, we used 4X5 grid and 3 in a row to decide winner with two players can play option. Placing player's mark three time in a row either vertically, horizontally, or diagonally will decided as winner. Played against two default agents called 'random' - easy to play against this agent and 'negamax' - hard to play against this agent.

## Working style of Environment and Default Agent

The ConnectX Environment places the player's piece vertically but the indices are count row wise which makes this game quite difficult to play against player like 'negamax'. It has been developed by mini-max algorithm that is based on the principle that one player's gain is another player's loss.

## Possible Actions

Possible approaches to this game is to focus on placing pieces 3 in a row in either aspect rather than blocking the opponent's pieces especially if we are the first player.
Placing pieces in the same column if it not fulled. And then placing pieces near by columns to make more opportunity to win with less actions rather than filling new column.
Using random actions.
Using learned policies.

## Learning policies

Used three methods to update policy. Updating randomly generated actions as a policy, SARSA derived policy and Q-Learning derived policy.

Learning policies from opponent's(negamax) action for each observation is the best strategy rather than storing randomly generated action.

Because initially I used random actions to create a policy using SARSA and Q-Learning method. None of them worked effectively against ‘nagamax’. It worked against ‘random’ agent only.

But copying the opponent’s action isn’t an easy task. Track opponent’s mark, find his recent mark, map his mark to retrieve which action the agent has taken is long and difficult task. It requires more logical way of interpretation of observation board and effective policy update technique. I tried up to a particular step, after that I unable to retrieve the opponent’s action successfully. And I think it’s more time consuming one as because the environment allows 60 seconds to make action to each player.

## Methodology

**RandomAgent2**:

In the act function, I used random action to make move unless some condition satisfied (epsilon). After that I used learned policy.

In the learn function, I stored randomly generated action for each observation to update my policy.

Working moderately against ‘random’ but not against ‘negamax’.

**SARSA Agent**:

In the act function, I used random action to make move unless some condition satisfied (epsilon). After that I used learned policy.

In the learn function, I updated observation and respected action values using SARSA policy update method..

Working moderately against ‘random’ but not against ‘negamax’.

**Q-Learning Agent**:

In the act function, I used random action to make move unless some condition satisfied (epsilon). After that I used learned policy.

In the learn function, I updated observation and respected action values using Q-Learning policy update method.

Working moderately against ‘random’ but not against ‘negamax’.

## Chosen Algorithm:

I prefer all three algorithms, because it’s my initial stage of developing these three agents to play environment like ConnectX. Unlike the frozen lake, it’s environment quite different and difficult to observe. Also tracking the second players move is crucial to make decision. Here, I can’t use a set of actions alternatively, I can only use same action when don’t have any policy in hand and haven’t used the random action. Also I don’t know how to pass desired action in a sequence order. So far, I learned so much about the environment and have some ideas to implement but the time is not sufficient. If I make the desired changes to all three agents, they will play at least at intermediate level against ‘negamax’. Even though the reward increased over time when palyed against ‘random’ but the learning rate is not enough to beat ‘negamax’.