assignment1

April 24, 2020

In [6]:
```
# Import libraries
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [7]:
```
# Helpers
def move_still_possible(S):
    return not (S[S==0].size == 0)


def move_o(S):
    xs, ys = np.where(S==0)

    i = np.random.permutation(np.arange(xs.size))[0]

    S[xs[i],ys[i]] = -1

    return S

def move_x(S, vs, count):
    xs, ys = np.where(S==0)
    valid = False

    # first 10 games, play by selection
    if count < 10:
        while(not valid):
            print("Make a move: ", end="")
            i = int(input()) - 1
            x = i % 3
            y = i // 3

            if S[y, x] == 0:
                S[y, x] = 1
                valid = True
            else:
                print("Invalid move")
```

```
        # play by AI
        else:
            # AI choice
            if np.random.random_sample((0,1)) > 0.1:
                vs_values = vs[np.array_str(S)]
                valid_poses = [ys[i] * 3 + xs[i] for i in range(xs.size)]

                max_v = -np.inf
                i = -1
                for k in range(vs_values.size):
                    if (vs_values[k] > max_v and k in valid_poses):
                        max_v = vs_values[k]
                        i = k

                S[xs[i],ys[i]] = 1

            # Rand choice
            else:
                i = np.random.permutation(np.arange(xs.size))[0]
                S[xs[i],ys[i]] = 1


        return S, i


    def move_was_winning_move(S, p):
        if np.max((np.sum(S, axis=0)) * p) == 3:
            return True

        if np.max((np.sum(S, axis=1)) * p) == 3:
            return True

        if (np.sum(np.diag(S)) * p) == 3:
            return True

        if (np.sum(np.diag(np.rot90(S))) * p) == 3:
            return True

        return False

    # print game state matrix using characters
    def print_game_state(S):
        B = np.copy(S).astype(object)
        for n in [-1, 0, 1]:
            B[B==n] = symbols[n]
        print (B)
```

```
        def updateVs(vs, tracker, lastMove, last_vs):
            # Tracker length
            n_0 = len(tracker) - 1

            # Traverse tracker in reverse order
            for x in range(n_0, -1, -1):
                # Update terminal state V(s)
                if (x == n_0):
                    vs[np.array_str(tracker[x])][lastMove] = last_vs
                    continue

                # Update previous states V(s)
                vs[np.array_str(tracker[x])] = (vs[np.array_str(tracker[x])] + 0.2 * (vs[np.ar


        def showStatistics(wins, count):
            freq = []
            for x in range(0, len(wins), 100):
                freq.append(wins[x-100:x].count(1) / 100)

            plt.plot(freq)
            plt.ylabel('Win frequency')
            plt.show()

In [8]: # Symbols
        # python dictionary to map integers (1, -1, 0) to characters ('x', 'o', ' ')
        symbols = { 1:'x',
                   -1:'o',
                    0:' '}

In [ ]: # First game state initialization
        gameState = np.zeros((3,3), dtype=int)
        # V(s) hash-table
        vs = {np.array_str(gameState): np.ones(9) * 0.1}

        # Game variable
        countGame = 0
        wins = []

        while(countGame < 1010):
            # initialize an empty tic tac toe board
            gameState = np.zeros((3,3), dtype=int)
            # Last player before terminal state
            lastMove = None
            last_vs = 0

            # initialize the player who moves first (either +1 or -1)
            player = 1
```

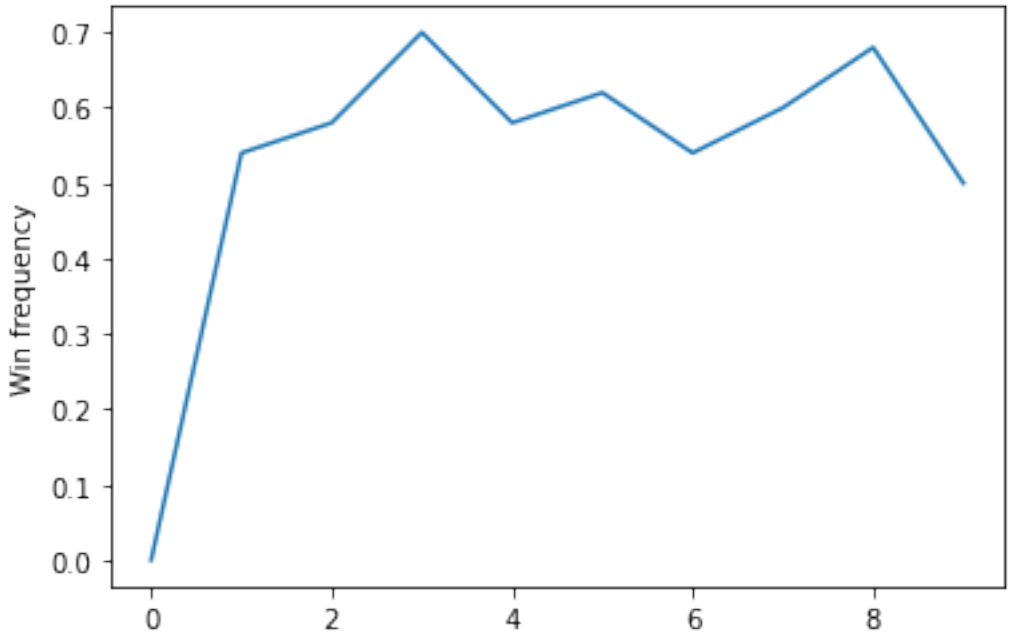
```
# initialize a move counter
mvcntr = 1

# initialize a flag that indicates whetehr or not game has ended
noWinnerYet = True

# State tracker
tracker = []

# Initialize vs with state0 V(s)
tracker.append(gameState)
while move_still_possible(gameState) and noWinnerYet:
    # turn current player number into player symbol
    name = symbols[player]
    print ('%s moves' % name)

    # let current player move at random
    if player == 1:
        gameState, lastMove = move_x(gameState, vs, countGame)
    else:
        gameState = move_o(gameState)

    # Check if the game state doesn't exists
    if np.array_str(gameState) not in vs.keys():
        vs[np.array_str(gameState)] = np.ones(9) * 0.1

    # Add successor state to V(s)
    tracker.append(gameState)

    # print current game state
    print_game_state(gameState)

    # evaluate current game state
    if move_was_winning_move(gameState, player):
        print ('player %s wins after %d moves' % (name, mvcntr))
        noWinnerYet = False
        last_vs = 1 if player == 1 else 0

    # switch current player and increase move counter
    player *= -1
    mvcntr +=  1

# Update Vs
updateVs(vs, tracker, lastMove, last_vs)

#ăUpdate wins when GameAI plays
if (countGame > 9):
```

```
                wins.append(last_vs)

            # Increase game counter
            countGame += 1

        if noWinnerYet:
            print ('game ended in a draw' )
```

In [12]:
```
# Number of states
print("Number of states played: " + str(len(vs.keys())))
```

```
Number of states played: 3197
```

In [13]:
```
# Show learning curve
showStatistics(wins, countGame)
```



In [ ]: