Problem 1 : Model Free Prediction

(a) True Value of each State :

State s_1 : 1 + 0 + 0.9(1+0) + 0.1(10+0) = 2.9
State s_2 : 2 + 0 + 0.9(1+0) + 0.1(10+0) = 3.9
State s_3 : 0 + 0.9(1+0) + 0.1(10+0) = 1.9
State s_4 : 1 + 0 = 1
State s_5 : 10 + 0 = 10
State s_6 : 0

(b) $V(s_1)$ and $V(s_2)$ using Monte Carlo Method:

$V(s_1) = \frac{(1+0+1+0)+(1+0+10+0)+(1+0+1+0)+(1+0+1+0)}{4} = 4.25$
$V(s_2) = \frac{2+0+10+0}{1} = 12$

(c) $V(s_1)$ and $V(s_2)$ using TD(0) Method:

First Sample: s_1 -> s_3 -> s_4 -> s_6
$V(s_1) = 0 + 1 * (1 + 0 - 0) = 1$
$V(s_3) = 0 + 1 * (0 + 0 - 0) = 0$
$V(s_4) = 0 + 1 * (1 + 0 - 0) = 1$

Second Sample: s_1 -> s_3 -> s_5 -> s_6
$V(s_1) = 1 + \frac{1}{2} * (1 + 0 - 1) = 1$
$V(s_3) = 0 + \frac{1}{2} * (0 + 0 - 0) = 0$
$V(s_5) = 0 + 1 * (10 + 0 - 0) = 10$

Third Sample: s_1 -> s_3 -> s_4 -> s_6
$V(s_1) = 1 + \frac{1}{3} * (1 + 0 - 1) = 1$
$V(s_3) = 0 + \frac{1}{3} * (0 + 1 - 0) = \frac{1}{3}$
$V(s_4) = 1 + \frac{1}{2} * (1 + 0 - 1) = 1$

Fourth Sample: s_1 -> s_3 -> s_4 -> s_6
$V(s_1) = 1 + \frac{1}{4} * (1 + \frac{1}{3} - 1) = \frac{13}{12}$
$V(s_3) = \frac{1}{3} + \frac{1}{4} * (0 + 1 - \frac{1}{3}) = \frac{1}{2}$
$V(s_4) = 1 + \frac{1}{3} * (1 + 0 - 1) = 1$

Fifth Sample: s_2 -> s_3 -> s_5 -> s_6
$V(s_2) = 0 + 1 * (2 + \frac{1}{2} - 0) = \frac{5}{2}$
$V(s_3) = \frac{1}{2} + \frac{1}{5} * (0 + 10 - \frac{1}{2}) = \frac{12}{5}$
$V(s_5) = 10 + \frac{1}{2} * (10 + 0 - 10) = 10$

Therefore, $V(s_1) = \frac{13}{12}$ and $V(s_2) = \frac{5}{2}$

(d) $V(s_2)$ using MLE:

Out of the 5 trajectories s_3 takes s_4 in 3 of them and s_5 in 2 of them. So, by MLE, P(s_4 | s_3) = 0.6 and P(s_5 | s_3) = 0.4

Hence, $V(s_3) = 0.6 * 1 + 0.4 * 10 = 4.6$
Therefore, $V(s_2) = 2 + V(s_3) = 6.6$

(e)

True Value of $V(s_2) = 3.9$
From the above values we can see that TD estimate is close to the true value and the Monte Carlo estimate

Problem 2 : On Learning Rates

$V(s) = V(s) + \alpha_t[r + \gamma V(s^{'}) - V(s)]$

Conditions for convergence:
$\sum_{t=1}^{t=\infty} \alpha_t = \infty$ and
$\sum_{t=1}^{t=\infty} \alpha_t^2 < \infty$

(1) $\alpha_t = \frac{1}{t}$ **:**

$\sum_{t=1}^{t=\infty} \alpha_t$ = $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}$ ....
But,

$\frac{1}{3} > \frac{1}{4}, \frac{1}{5} > \frac{1}{8}, \frac{1}{6} > \frac{1}{8}, \frac{1}{7} > \frac{1}{8}\dots$

Hence,

$\frac{1}{3} + \frac{1}{4} > 2 * \frac{1}{4} \;\; and \;\; \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} > 4 * \frac{1}{8}\dots$

$\implies \sum_{t=1}^{t=\infty} \alpha_t > 1 + \frac{1}{2} + 2 * \frac{1}{4} + 4 * \frac{1}{8} + 8 * \frac{1}{16}\dots$

$\sum_{t=1}^{t=\infty} \alpha_t > 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \frac{1}{2}\dots$

$\sum_{t=1}^{t=\infty} \alpha_t > 1 + \sum_{t=1}^{t=\infty} \frac{1}{2}$

$\sum_{t=1}^{t=\infty} \alpha_t > \infty$, which satisfies condition 1.

Consider $\sum_{t=1}^{t=\infty} \alpha_t^2 = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \frac{1}{7^2}\dots$

But,

$\frac{1}{3^2} < \frac{1}{2^2} \;\; and \;\; \frac{1}{5^2} < \frac{1}{4^2}, \frac{1}{6^2} < \frac{1}{4^2}, \frac{1}{7^2} < \frac{1}{4^2}\dots$

Hence,

$\frac{1}{2^2} + \frac{1}{3^2} < 2 * \frac{1}{2^2} \;\; and \;\; \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \frac{1}{7^2} < 4 * \frac{1}{4^2}$ and so on ....

$\implies \sum_{t=1}^{t=\infty} \alpha_t^2 < 1 + 2 * \frac{1}{2^2} + 4 * \frac{1}{4^2} + 8 * \frac{1}{8^2}\dots$

$\sum_{t=1}^{t=\infty} \alpha_t^2 < 1 + \frac{1}{2} + 4 * \frac{1}{4} + \frac{1}{8} + \frac{1}{16}\dots$

$\sum_{t=1}^{t=\infty} \alpha_t^2 < \sum_{t=0}^{t=\infty} \frac{1}{2^t}\dots$

$\sum_{t=1}^{t=\infty} \alpha_t^2 < 2 < \infty\dots$, which satisfies condition 2.

Therefore, for $\alpha_t = \frac{1}{t}$, V(s) converges

(2) $\alpha_t = \frac{1}{t^2}$ **:**

From the 1st problem above, $\sum_{t=1}^{t=\infty} \frac{1}{t^2} < \infty$ which fails condition 1.

Therefore, for $\alpha_t = \frac{1}{t^2}$ **, V(s) does not converge**

(3) $\alpha_t = \frac{1}{t^{\frac{2}{3}}}$ **:**

$\alpha_t = \frac{1}{t^{\frac{2}{3}}}$

$t \geq 1 \implies t^{\frac{2}{3}} < t \implies \frac{1}{t^{\frac{2}{3}}} > \frac{1}{t}$

$\implies \sum_{t=1}^{t=\infty} \frac{1}{t^{\frac{2}{3}}} > \sum_{t=1}^{t=\infty} \frac{1}{t}$

$\implies \sum_{t=1}^{t=\infty} \frac{1}{t^{\frac{2}{3}}} = \infty$

$\implies \sum_{t=1}^{t=\infty} \alpha_t = \infty$, which satisfies condition 1.

Consider $\sum_{t=1}^{t=\infty} \alpha_t^2 = \frac{1}{1^2}^{\frac{2}{3}} + \frac{1}{2^2}^{\frac{2}{3}} + \frac{1}{3^2}^{\frac{2}{3}} + \frac{1}{4^2}^{\frac{2}{3}} + \frac{1}{5^2}^{\frac{2}{3}} + \frac{1}{6^2}^{\frac{2}{3}} + \frac{1}{7^2}^{\frac{2}{3}} \ldots.$

But,

$\frac{1}{3^2}^{\frac{2}{3}} < \frac{1}{2^2}^{\frac{2}{3}} \ \ and \ \ \frac{1}{5^2}^{\frac{2}{3}} < \frac{1}{4^2}^{\frac{2}{3}}, \frac{1}{6^2}^{\frac{2}{3}} < \frac{1}{4^2}^{\frac{2}{3}}, \frac{1}{7^2}^{\frac{2}{3}} < \frac{1}{4^2}^{\frac{2}{3}} \ldots.$

Hence,

$\frac{1}{2^2}^{\frac{1}{3}} + \frac{1}{3^2}^{\frac{1}{3}} < 2 * \frac{1}{2^2}^{\frac{1}{3}} \ \ and \ \ \frac{1}{4^2}^{\frac{1}{3}} + \frac{1}{5^2}^{\frac{1}{3}} + \frac{1}{6^2}^{\frac{1}{3}} + \frac{1}{7^2}^{\frac{1}{3}} < 4 * \frac{1}{4^2}^{\frac{1}{3}}$ and so on ....

$\implies \sum_{t=1}^{t=\infty} \alpha_t^2 < 1 + \frac{1}{2^{\frac{1}{3}}} + \frac{1}{4^{\frac{1}{3}}} + \frac{1}{8^{\frac{1}{3}}} \ldots.$

$\sum_{t=1}^{t=\infty} \alpha_t^2 < 1 + \frac{1}{2^{\frac{1}{3}}} + \frac{1}{2^{\frac{2}{3}}} + \frac{1}{2^{\frac{3}{3}}} \ldots.$

$\sum_{t=1}^{t=\infty} \alpha_t^2 < \frac{1}{1 - \frac{1}{2^{\frac{1}{3}}}} < \infty$, which satisfies condition 2.

Therefore, for $\alpha_t = \frac{1}{t^{\frac{2}{3}}}$ **, V(s) converges**

(4) $\alpha_t = \frac{1}{t^{\frac{1}{2}}}$ **:**

$\sum_{t=1}^{t=\infty} \alpha_t^2 = \sum_{t=1}^{t=\infty} \frac{1}{t} = \infty$, which fails condition 2.

Therefore, for $\alpha_t = \frac{1}{t^{\frac{1}{2}}}$ **, V(s) does not converge**

Generalisation:

$For\ \alpha_t = \frac{1}{t^p}$

If p $\leq$ 1, $\sum_{t=1}^{t=\infty} \frac{1}{t^p} \geq \sum_{t=1}^{t=\infty} \frac{1}{t}$

$\implies \sum_{t=1}^{t=\infty} \alpha_t \geq \sum_{t=1}^{t=\infty} \frac{1}{t}$

$\implies \sum_{t=1}^{t=\infty} \alpha_t = \infty$

So if p <=1 $\sum_{t=1}^{t=\infty} \alpha_t = \infty$ and

If 2p > 1 $\sum_{t=1}^{t=\infty} \alpha_t^2 < \infty$

So for V(s) to converge, **p should belong to the range (1/2 , 1].**

Problem 3 : Policy Improvement

$Q_\pi(s, \pi^{'}(s)) = \sum_{a\epsilon A} \pi(a|s) Q_\pi(s,a)$

$= \frac{\epsilon}{m} \sum_{a\epsilon A} Q(s,a) + (1-\epsilon) max_{a\epsilon A} Q_\pi(s,a)$

But ,

$max_{a\epsilon A} Q_\pi(s,a) \geq \sum_{a\epsilon A} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1-\epsilon} Q_\pi(s,a)$

$\implies Q_\pi(s, \pi^{'}(s)) \geq \frac{\epsilon}{m} \sum_{a\epsilon A} Q(s,a) + (1-\epsilon) \sum_{a\epsilon A} \frac{\pi(a|s) - \frac{\epsilon}{m}}{1-\epsilon} Q_\pi(s,a)$

$Q_\pi(s, \pi^{'}(s)) \geq \frac{\epsilon}{m} \sum_{a\epsilon A} Q(s,a) + \sum_{a\epsilon A} \pi(a|s) Q_\pi(s,a) - \sum_{a\epsilon A} \frac{\epsilon}{m} Q_\pi(s,a)$

$Q_\pi(s, \pi^{'}(s)) \geq \sum_{a\epsilon A} \pi(a|s) Q(s,a)$

$V_{\pi'}(s) \geq V_\pi(s)$

Therefore, there is always a policy improvement.

Problem 4 : λ Return

$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^n$

$G_t^n = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n})$

Given,

$\lambda^{n-1} \leq \frac{1}{2}$

$\implies (n-1)log(\lambda) \geq log(\frac{1}{2}) \implies n \geq 1 - \frac{log(2)}{log\lambda}$

Therefore, $\eta(\lambda) = \left\lceil 1 - \frac{log(2)}{log\lambda} \right\rceil$

Given, $\eta(\lambda) = 3$

$\implies \left\lceil 1 - \frac{log(2)}{log\lambda} \right\rceil = 3$

$\implies 2 < 1 - \frac{log(2)}{log\lambda} \leq 3$

$\implies -2 \leq \frac{log(2)}{log\lambda} < -1$

$\implies -1 * log(2) < log(\lambda) \leq -\frac{1}{2} * log(2)$

$\implies 0.5 < \lambda \leq \frac{1}{\sqrt{2}}$

Problem 5 : Q-Learning

Initially all the values in the qtable are initialized to 0. Suppose in the first iteration of the training we started with state s_1 . with probability $\frac{1-2\epsilon}{3}$ we chose the argmax of the state, which is action a_1 in this case. Hence the update Q value corresponding to state s_1 and action a_1 is 0.7. In the second iteration if were to chose the action corresponding to epsilon greedy then we would have chosen action a_1, but the chosen action was a_2 which could probably have been random action.

So the first action, a_1, is a greedy action and the second action, a_2, is a random action.

Problem 6 : Game of Pac-Man

In [1]:

```
import numpy as np                    # For problem 6 and 7
```

In [2]:

```
class Environment_Pacman:
    def __init__(self,size,num_pellets):
        self.size = size
        self.num_pellets = num_pellets
        self.actions = [(0,1),(1,0),(0,-1),(-1,0),(0,0)]

    def initialize(self):
        self.pacman = (np.random.randint(1,self.size-1),np.random.randint(1,self
.size-1))
        self.new_ghost()
        self.new_pellets()
        self.reward = 0
        state = self.get_state()

        return state

    def new_ghost(self):
        (r,c) = self.pacman
        locations = []
        if c!=1:
            locations.append((r,1))
        if r!=1:
            locations.append((1,c))
        if c!=self.size-2:
            locations.append((r,self.size-2))
        if r!=self.size-2:
            locations.append((self.size-2,c))

        self.ghost = locations[np.random.randint(0,len(locations))]

    def new_pellets(self):
        locations = []
        for r in range(1,self.size-1):
            for c in range(1,self.size-1):
                if (r,c) != self.pacman:
                    locations.append((r,c))

        np.random.shuffle(locations)
        self.pellets = []
        for _ in range(self.num_pellets):
            self.pellets.append(locations.pop())

    def end(self):
        if self.reward == -100:
            return True
        return False

    def act(self,action):
        pacman = self.pacman
        ghost = self.ghost
        self.pacman = tuple(map(sum,zip(self.pacman,self.actions[action])))
        ghost_action = np.random.randint(0,4)
        self.ghost = tuple(map(sum,zip(self.ghost,self.actions[ghost_action])))

        (gr,gc) = self.ghost
        if gr == 0 or gr == self.size-1:
            self.new_ghost()
        elif gc == 0 or gc == self.size-1:
            self.new_ghost()
```

```
        (pr,pc) = self.pacman
        if self.pacman == self.ghost:
            self.reward = -100
        elif (pacman,ghost) == (self.ghost,self.pacman):
            self.reward = -100
        elif pr == 0 or pr == self.size-1:
            self.reward = -100
        elif pc == 0 or pc == self.size-1:
            self.reward = -100
        elif self.pacman in self.pellets:
            self.reward = 10
            self.pellets.remove(self.pacman)
        else:
            self.reward = 0

        if len(self.pellets) == 0:
            self.new_pellets()

        state = self.get_state()
        reward = self.reward
        done = self.end()

        return state,reward,done

    def get_state(self):
        state = str(self.pacman)+str(self.ghost)+str(sorted(self.pellets))
        return state

    def render(self):
        board_str = ''
        for r in range(self.size):
            for c in range(self.size):
                if (r,c) == self.pacman:
                    board_str += 'P'
                elif (r,c) == self.ghost:
                    board_str += 'G'
                elif (r,c) in self.pellets:
                    board_str += '.'
                elif r == 0 or r == self.size-1:
                    board_str += 'X'
                elif c == 0 or c == self.size-1:
                    board_str += 'X'
                else:
                    board_str += ' '
            board_str += '\n'
        board_str += '\n'
        print(board_str)
```

In [3]:

```
def update_qtable_pacman(qtable,state):
    if state not in qtable.keys():
        qtable[state] = np.zeros(5)
    return qtable
```

In [4]:

```
def chose_action_pacman(qtable,state,epsilon):
    if state in qtable.keys():
        prob = [1-((4*epsilon)/5)]
        actions = [np.argmax(qtable[state])]
        for i in range(5):
            if i != actions[0]:
                prob.append(epsilon/5)
                actions.append(i)

        action = np.random.choice(actions,p = prob)
        return qtable,action
    else:
        qtable[state] = np.zeros(5)
        return qtable,np.random.randint(5)
```

In [5]:

```
def qlearning_train_pacman(size,num_pellets,learning_rate = 0.1,gamma = 0.9,epsi
lon = 0.5,num_train_games = 10000):

    qtable = {}
    game = Environment_Pacman(size,num_pellets)

    print('-------------------- Training --------------------')
    for episode in range(num_train_games):
        if episode%1000 == 0:
            print('Episode : ',episode)

        state = game.initialize()
        done = False
        qtable = update_qtable_pacman(qtable,state)

        while not done:
            qtable,action = chose_action_pacman(qtable,state,epsilon)
            next_state,reward,done = game.act(action)

            qtable = update_qtable_pacman(qtable,next_state)
            qtable[state][action] = qtable[state][action] + learning_rate*(rewar
d + \
                                gamma*np.max(qtable[next_state]) - qtable[state]
[action])
            state = next_state

    print('-------------------- Finished Training --------------------')
    return qtable
```

In [6]:

```
def sarsa_train_pacman(size,num_pellets,learning_rate = 0.1,gamma = 0.9,epsilon
= 0.5,num_train_games = 10000):

    qtable = {}
    game = Environment_Pacman(size,num_pellets)

    print('-------------------- Training --------------------')
    for episode in range(num_train_games):
        if episode%1000 == 0:
            print('Episode : ',episode)

        state = game.initialize()
        done = False
        qtable = update_qtable_pacman(qtable,state)

        while not done:
            qtable,action = chose_action_pacman(qtable,state,epsilon)
            next_state,reward,done = game.act(action)

            qtable,max_idx = chose_action_pacman(qtable,next_state,epsilon)
            qtable[state][action] = qtable[state][action] + learning_rate*(rewar
d + \
                                    gamma*qtable[next_state][max_idx] - qtable[state
][action])
            state = next_state

    print('-------------------- Finished Training --------------------')
    return qtable
```

In [7]:

```
def test_pacman(size,num_pellets,qtable,num_test_games,display_flag):
    score = 0
    game = Environment_Pacman(size,num_pellets)

    for eps in range(num_test_games):
        state = game.initialize()
        done = False

        while not done:
            if display_flag == 1:
                game.render()
            _,action = chose_action_pacman(qtable,state,0)
            next_state,reward,done = game.act(action)
            if reward > 0:
                score += reward
            state = next_state
        if display_flag == 1:
                game.render()

    print('Average Score over ',num_test_games,' test is : ',score/num_test_game
s)
```

In [8]:

```
# Parameters. Please make sure to increase the number of train episodes accordin
gly when you increase the size of the grid world.
size = 5
num_pellets = 3
learning_rate = 0.1
gamma = 0.9
epsilon = 0.05
num_train_games = 10000
```

In [9]:

```
qtable_qlearning_pacman = qlearning_train_pacman(size,num_pellets,learning_rate,
gamma,epsilon,num_train_games)
```

```
-------------------- Training --------------------
Episode :  0
Episode :  1000
Episode :  2000
Episode :  3000
Episode :  4000
Episode :  5000
Episode :  6000
Episode :  7000
Episode :  8000
Episode :  9000
-------------------- Finished Training --------------------
```

In [10]:

```
# If you want to see the pacman in action, decrease the number of test games to
 1 and change the display_flag to 1
num_test_games = 100
display_flag = 0
```

In [11]:

```
test_pacman(size,num_pellets,qtable_qlearning_pacman,num_test_games,display_flag
)
```

```
Average Score over  100  test is :  303.3
```

```
XXXXX
X   X
XG  X
X. PX
XXXXX


XXXXX
X G X
X   X
X.P X
XXXXX


XXXXX
X. .X
X G X
XP  X
XXXXX


XXXXX
X.G.X
XP. X
X   X
XXXXX


XXXXX
X. .X
X P X
X   X
XXXXX


Average Score over  1  test is :  240.0
```

In [13]:

```
qtable_sarsa_pacman = sarsa_train_pacman(size,num_pellets,learning_rate,gamma,ep
silon,num_train_games)
```

```
-------------------- Training --------------------
Episode :  0
Episode :  1000
Episode :  2000
Episode :  3000
Episode :  4000
Episode :  5000
Episode :  6000
Episode :  7000
Episode :  8000
Episode :  9000
-------------------- Finished Training --------------------
```

In [14]:

```
test_pacman(size,num_pellets,qtable_sarsa_pacman,num_test_games,display_flag)
```

```
Average Score over  100  test is :  603.5
```

```
XXXXX
X. .X
X   X
XP.GX
XXXXX


XXXXX
X. .X
XP  X
X G X
XXXXX


XXXXX
X. .X
P   X
X .GX
XXXXX


Average Score over  1  test is :  210.0
```

Sarsa Agent did better as he scored more than Q learning Agent

Problem 7 : Game of Tic-Tac-Toe

(a,b)

In [16]:

```
class Environment_TTT:
    def __init__(self):
        self.state = np.zeros(9)
        self.empty_spots = np.arange(9)

    def init(self):
        self.state.fill(0)
        self.empty_spots = np.arange(9)
        player_side = np.random.choice([1,-1])
        opponent_side = -1*player_side
        return str(self.state),player_side,opponent_side

    def check_win(self):
        for i in range(3):
            if ((self.state[3*i]+self.state[3*i+1]+self.state[3*i+2] == 3) or \
                (self.state[3*i]+self.state[3*i+1]+self.state[3*i+2] == -3)):
                return True
            if ((self.state[i]+self.state[i+3]+self.state[i+6] == 3) or \
                (self.state[i]+self.state[i+3]+self.state[i+6] == -3)):
                return True
        if ((self.state[0] + self.state[4] + self.state[8] == 3) or \
            (self.state[0] + self.state[4] + self.state[8] == -3)):
            return True
        if ((self.state[2] + self.state[4] + self.state[6] == 3) or \
            (self.state[2] + self.state[4] + self.state[6] == -3)):
            return True

        return False

    def check_draw(self):
        if np.any(self.state == 0):
            return False
        return True

    def act(self,pos,side):
        self.state[pos] = side
        if self.check_win():
            return str(self.state),1,True
        if self.check_draw():
            return str(self.state),0.5,True
        return str(self.state),0.01,False

    def random_act(self,side):
        pos = np.random.choice(self.empty_spots)
        state,reward,done = self.act(pos,side)
        return state,reward,done,pos

    def safe_act(self,side):
        for pos in self.empty_spots:
            state,reward,done = self.act(pos,side)
            if reward == 1:
                return state,reward,done,pos
            else:
                self.state[pos] = 0

        for pos in self.empty_spots:
            state,reward,done = self.act(pos,-1*side)
            if reward == 1:
                state,reward,done = self.act(pos,side)
```

```
                return state,reward,done,pos
            else:
                self.state[pos] = 0

        state,reward,done,pos = self.random_act(side)
        return state,reward,done,pos

    def state_to_char(self,pos):
        if self.state[pos] == 0:
            return ' '
        if self.state[pos] == -1:
            return 'o'
        return 'x'

    def render(self):
        for i in range(3):
            board_str = self.state_to_char(i*3) + '|' +self.state_to_char(i*3+1)
\
            + '|' + self.state_to_char(i*3+2)

            print(board_str)
            if i != 2:
                print('-----')

        print("")
```

(c)

In [17]:

```
def update_qtable_ttt(qtable,state):
    if state not in qtable.keys():
        qtable[state] = np.zeros(9)
    return qtable
```

In [18]:

```
def chose_action_ttt(qtable,state,empty_spots,epsilon):
    if state in qtable.keys():
        if len(empty_spots) == 1:
            return qtable,empty_spots[0]
        prob = [1-epsilon+epsilon/len(empty_spots)]
        max_idx = np.argmax(qtable[state][empty_spots])
        positions = [empty_spots[max_idx]]
        for i in empty_spots:
            if i != positions[0]:
                prob.append(epsilon/len(empty_spots))
                positions.append(i)
        pos = np.random.choice(positions,p = prob)
        return qtable,pos
    else:
        qtable[state] = np.zeros(9)
        return qtable,np.random.choice(empty_spots)
```

In [19]:

```
def qlearning_train_ttt(opponent_choice,learning_rate = 0.7,gamma = 0.7,epsilon
= 0.05,num_train_games = 10000,num_val_games = 100):
    qtable = {}
    game = Environment_TTT()

    print('-------------------- Training --------------------')
    for episode in range(num_train_games):
        if episode%500 == 0:
            print('Episode : ',episode)

        state,player_side,opponent_side = game.init()
        done = False
        opponent = np.random.choice(opponent_choice)
        turn = 1

        if episode == 0:
            qtable = update_qtable_ttt(qtable,state)

        while not done:
            if turn == opponent_side:
                if opponent == 0:
                    next_state,reward,done,pos = game.random_act(opponent_side)
                else:
                    next_state,reward,done,pos = game.safe_act(opponent_side)
            else:
                qtable,pos = chose_action_ttt(qtable,state,game.empty_spots,epsi
lon)
                next_state,reward,done = game.act(pos,player_side)

            qtable = update_qtable_ttt(qtable,next_state)
            qtable[state][pos] = qtable[state][pos] + learning_rate*(reward - \
                                gamma*np.max(qtable[next_state]) - qtable[state]
[pos])

            turn *= -1
            game.empty_spots = game.empty_spots[game.empty_spots != pos]
            state = next_state

        if (episode+1)%200 == 0:
            print('-------------------- Validation --------------------')
            test_ttt(qtable,num_val_games,0,0)
            print('-------------------- Finished Validation --------------------
')

    print('-------------------- Finished Training --------------------')

    return qtable
```

In [20]:

```
def test_ttt(qtable,num_test_games,verbose,display_flag):
    wins_random = 0
    draws_random = 0
    loss_random = 0
    wins_safe = 0
    draws_safe = 0
    loss_safe = 0
    counter = 0

    game = Environment_TTT()

    for eps in range(num_test_games):
        state,player_side,opponent_side = game.init()
        done = False
        opponent = np.random.choice([0,1])
        turn = 1
        if display_flag == 1:
            if player_side == 1:
                if opponent == 0:
                    print('Our Player is X and Opponent is a Random Agent')
                else:
                    print('Our Player is X and Opponent is a Safe Agent')
            else:
                if opponent == 0:
                    print('Our Player is O and Opponent is a Random Agent')
                else:
                    print('Our Player is O and Opponent is a Safe Agent')

        if opponent == 0:
            counter += 1

        while not done:
            if turn == opponent_side:
                if opponent == 0:
                    next_state,reward,done,pos = game.random_act(opponent_side)
                else:
                    next_state,reward,done,pos = game.safe_act(opponent_side)
            else:
                _,pos = chose_action_ttt(qtable,state,game.empty_spots,0)
                next_state,reward,done = game.act(pos,player_side)

            if display_flag == 1:
                game.render()

            if done and reward == 1 and turn == player_side:
                if display_flag == 1:
                    print('Our Player has won!!')
                if opponent == 0:
                    wins_random += 1
                else:
                    wins_safe += 1
            if done and reward == 1 and turn == opponent_side:
                if display_flag == 1:
                    print('Opponent has won.')
                if opponent == 0:
                    loss_random += 1
                else:
                    loss_safe += 1
            elif done and reward == 0.5:
```

```
                if display_flag == 1:
                    print('It\'s a draw!')
                if opponent == 0:
                    draws_random += 1
                else:
                    draws_safe += 1

            turn *= -1
            game.empty_spots = game.empty_spots[game.empty_spots != pos]
            state = next_state

    if verbose == 0:
        print('Wins : ',wins_random+wins_safe,'\tDraws : ',draws_random+draws_sa
fe,'\tLost : ',loss_random+loss_safe)
    elif verbose == 1:
        print('Stats : ')
        print('Total Games Played : ',num_test_games)
        print('Total Games Won : ',wins_random+wins_safe)
        print('Total Games Drawn : ',draws_random+draws_safe)
        print('Total Games Lost : ',loss_random+loss_safe)
        print('Games Played against Random Agent : ',counter)
        print('\tGames Won against Random Agent : ',wins_random)
        print('\tGames Drawn against Random Agent : ',draws_random)
        print('\tGames Lost against Random Agent : ',loss_random)
        print('Games Played against Safe Agent : ',num_test_games-counter)
        print('\tGames Won against Safe Agent : ',wins_safe)
        print('\tGames Drawn against Safe Agent : ',draws_safe)
        print('\tGames Lost against Safe Agent : ',loss_safe)
```

In [21]:

```
# Opponent 0 for Random Agent and 1 for Safe Agent
# Verbose parameter in the test_ttt function if set to 1 gives a
# detailed description of the test results. Display_flag when set to 1 helps in
 seeing our agent in action.
```

(1) Training against only the Random Agent.

In [22]:

```
learning_rate = 0.7
gamma = 0.7
epsilon = 0.05
opponent = [0]
num_test_games = 1000
```

In [23]:

```
qtable_1 = qlearning_train_ttt(opponent,learning_rate,gamma,epsilon)
```

In [24]:

```
test_ttt(qtable_1,num_test_games,1,0)
```

```
Stats :
Total Games Played :  1000
Total Games Won :  563
Total Games Drawn :  222
Total Games Lost :  215
Games Played against Random Agent :  507
        Games Won against Random Agent :  388
        Games Drawn against Random Agent :  65
        Games Lost against Random Agent :  54
Games Played against Safe Agent :  493
        Games Won against Safe Agent :  175
        Games Drawn against Safe Agent :  157
        Games Lost against Safe Agent :  161
```

In [25]:

```
test_ttt(qtable_1,1,0,1)
```

```
Our Player is X and Opponent is a Safe Agent
 | |
-----
 | |
-----
 | |x

 | |
-----
 | |
-----
 |o|x

 | |x
-----
 | |
-----
 |o|x

 | |x
-----
 | |o
-----
 |o|x

x| |x
-----
 | |o
-----
 |o|x

x|o|x
-----
 | |o
-----
 |o|x

x|o|x
-----
 |x|o
-----
 |o|x

Our Player has won!!
Wins :  1      Draws :  0      Lost :  0
```

(2) Training against only the Safe Agent.

In [26]:

```
learning_rate = 0.7
gamma = 0.7
epsilon = 0.05
opponent = [1]
num_test_games = 1000
```

In [27]:

```
qtable_2 = qlearning_train_ttt(opponent,learning_rate,gamma,epsilon)
```

In [35]:

```
test_ttt(qtable_2,num_test_games,1,0)
```

```
Stats :
Total Games Played :  1000
Total Games Won :  692
Total Games Drawn :  285
Total Games Lost :  23
Games Played against Random Agent :  526
        Games Won against Random Agent :  446
        Games Drawn against Random Agent :  61
        Games Lost against Random Agent :  19
Games Played against Safe Agent :  474
        Games Won against Safe Agent :  246
        Games Drawn against Safe Agent :  224
        Games Lost against Safe Agent :  4
```

In [29]:

```
test_ttt(qtable_2,1,0,1)
```

```
Our Player is X and Opponent is a Random Agent
 | |x
-----
 | |
-----
 | |

 | |x
-----
 | |
-----
 | |o

 |x|x
-----
 | |
-----
 | |o

 |x|x
-----
 | |
-----
o| |o

x|x|x
-----
 | |
-----
o| |o

Our Player has won!!
Wins :  1       Draws :  0      Lost :  0
```

(3) Training against both Random and Safe Agent.

In [30]:

```
learning_rate = 0.7
gamma = 0.7
epsilon = 0.05
opponent = [0,1]
num_test_games = 1000
```

In [31]:

```
qtable_3 = qlearning_train_ttt(opponent,learning_rate,gamma,epsilon)
```

In [34]:

```
test_ttt(qtable_3,num_test_games,1,0)
```

```
Stats :
Total Games Played :  1000
Total Games Won :  647
Total Games Drawn :  274
Total Games Lost :  79
Games Played against Random Agent :  501
        Games Won against Random Agent :  421
        Games Drawn against Random Agent :  54
        Games Lost against Random Agent :  26
Games Played against Safe Agent :  499
        Games Won against Safe Agent :  226
        Games Drawn against Safe Agent :  220
        Games Lost against Safe Agent :  53
```

In [33]:

```
test_ttt(qtable_3,1,0,1)
```

```
Our Player is X and Opponent is a Random Agent
 | |x
-----
 | |
-----
 | |

 | |x
-----
o| |
-----
 | |

 | |x
-----
o| |
-----
 | |x

 | |x
-----
o| |
-----
 |o|x

 | |x
-----
o| |x
-----
 |o|x

Our Player has won!!
Wins :  1       Draws :  0      Lost :  0
```

(4) The 2nd Agent is the better one as we can see from the above results. It losses less games relatively and wins the same number of games as the other agents. It's quiet difficult to win games playing 2nd and this agent makes sure it doesn't lose those games.

(5) The Agent is definetely not unbeatable. Training it with a better agent or by training it using a different approach such as DQNs or DDQNs might increase the agents performance.