# Implementation of Q-Learning for Dots and Boxes

## IMPLEMENTATION OF Q-LEARNING TECHNIQUE – 2 X 2 GRID

Learning rate : 0.6

Discount factor: 0.7

Epsilon : 0.6 – for epsilon greedy policy

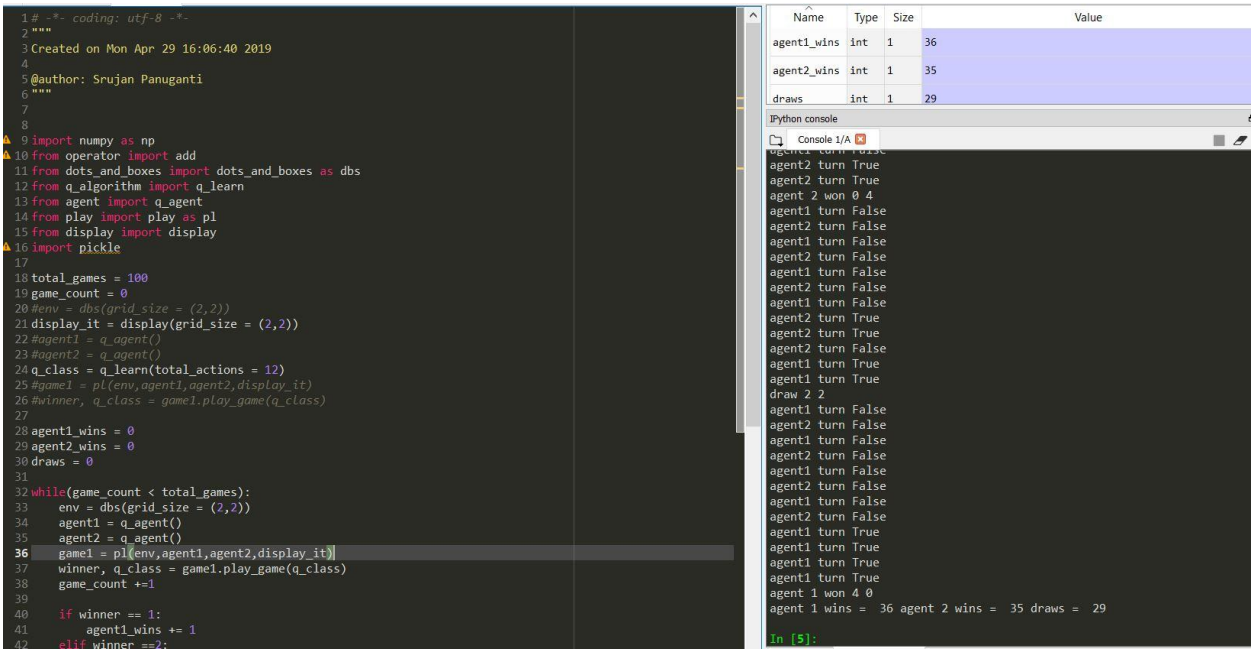
**For 100 games – self Play:**

Time consumed : 1 second

Agent 1 wins : 36

Agent 2 wins : 35

Draws : 29



```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 29 16:06:40 2019

@author: Srujan Panuganti
"""


import numpy as np
from operator import add
from dots_and_boxes import dots_and_boxes as dbs
from q_algorithm import q_learn
from agent import q_agent
from play import play as pl
from display import display
import pickle

total_games = 100
game_count = 0
#env = dbs(grid_size = (2,2))
display_it = display(grid_size = (2,2))
#agent1 = q_agent()
#agent2 = q_agent()
q_class = q_learn(total_actions = 12)
#game1 = pl(env,agent1,agent2,display_it)
#winner, q_class = game1.play_game(q_class)

agent1_wins = 0
agent2_wins = 0
draws = 0

while(game_count < total_games):
    env = dbs(grid_size = (2,2))
    agent1 = q_agent()
    agent2 = q_agent()
    game1 = pl(env,agent1,agent2,display_it)
    winner, q_class = game1.play_game(q_class)
    game_count +=1

    if winner == 1:
        agent1_wins += 1
    elif winner ==2:
```

| Name | Type | Size | Value |
|---|---|---|---|
| agent1_wins | int | 1 | 36 |
| agent2_wins | int | 1 | 35 |
| draws | int | 1 | 29 |

IPython console

Console 1/A

```
agent2 turn True
agent2 turn True
agent 2 won 0 4
agent1 turn False
agent2 turn False
agent1 turn False
agent2 turn False
agent1 turn False
agent2 turn False
agent1 turn False
agent2 turn True
agent2 turn True
agent2 turn False
agent1 turn True
agent1 turn True
draw 2 2
agent1 turn False
agent2 turn False
agent1 turn False
agent2 turn False
agent1 turn False
agent2 turn False
agent1 turn False
agent2 turn False
agent1 turn True
agent1 turn True
agent1 turn True
agent1 turn True
agent 1 won 4 0
agent 1 wins =  36 agent 2 wins =  35 draws =  29

In [5]:
```

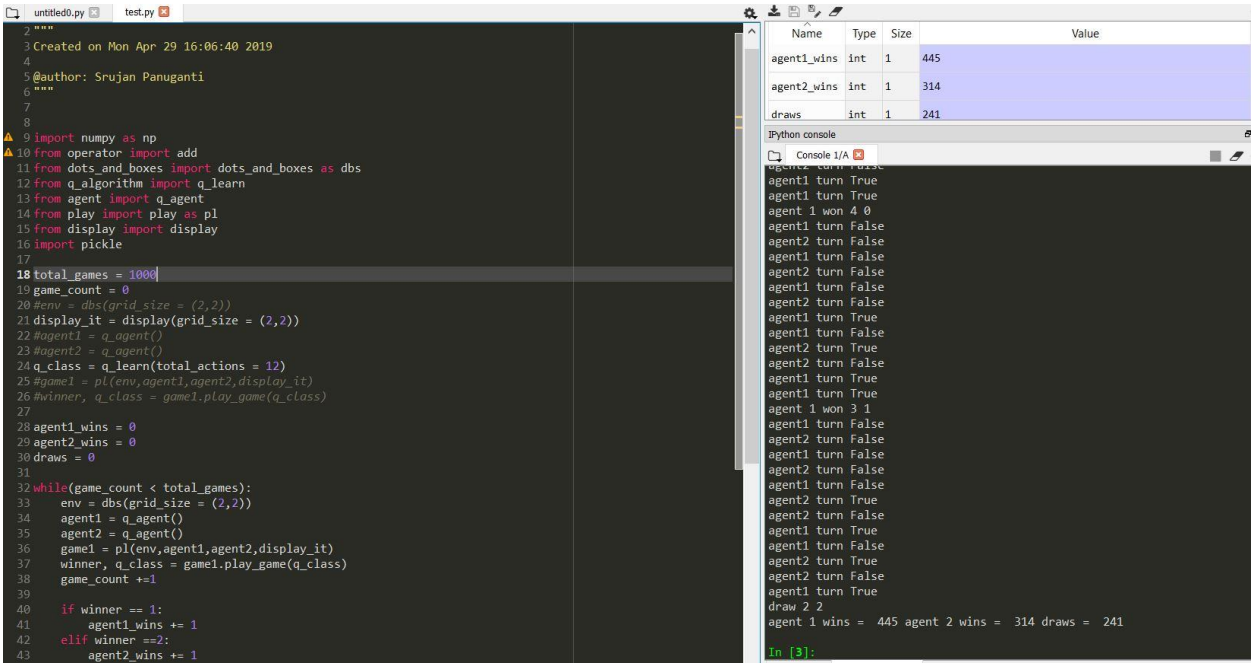
**For 1000 games – Self Play:**

Time consumed: 13 seconds

Agent 1 wins :445

Agent 2 wins: 314

Draws : 241



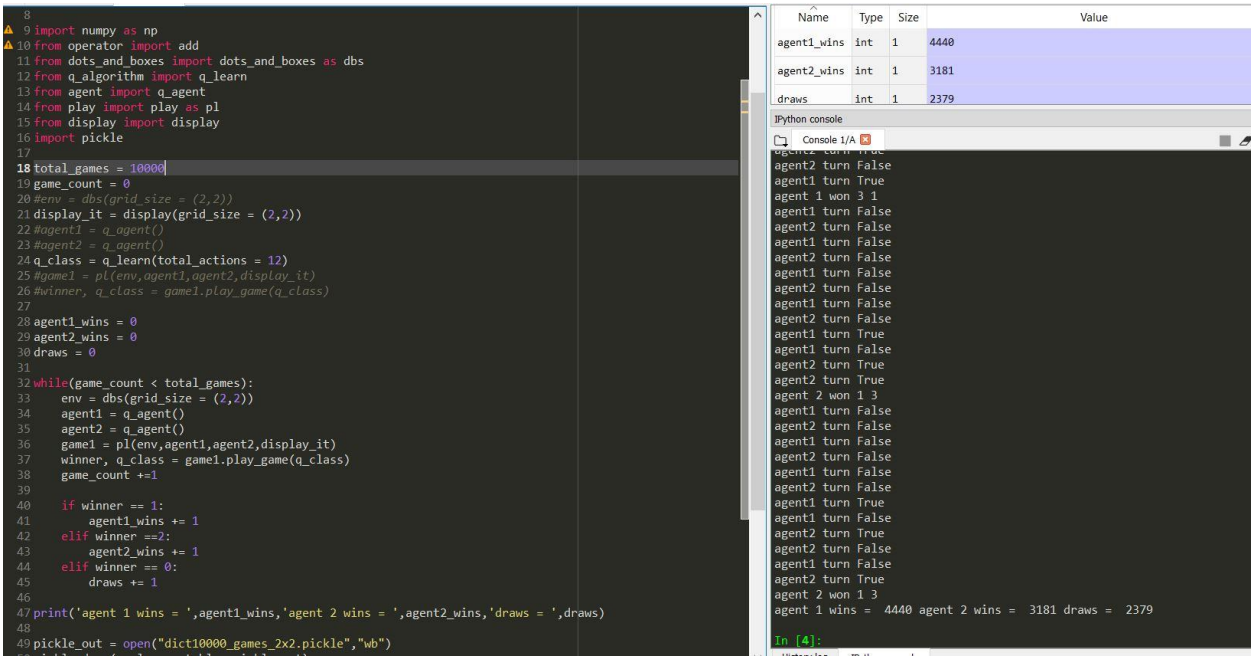

**For 10000 games – Self Play:**

Time consumed 1 minute 45 seconds

Agent 1 wins : 4440

Agent 2 wins : 3181

Draws : 2379

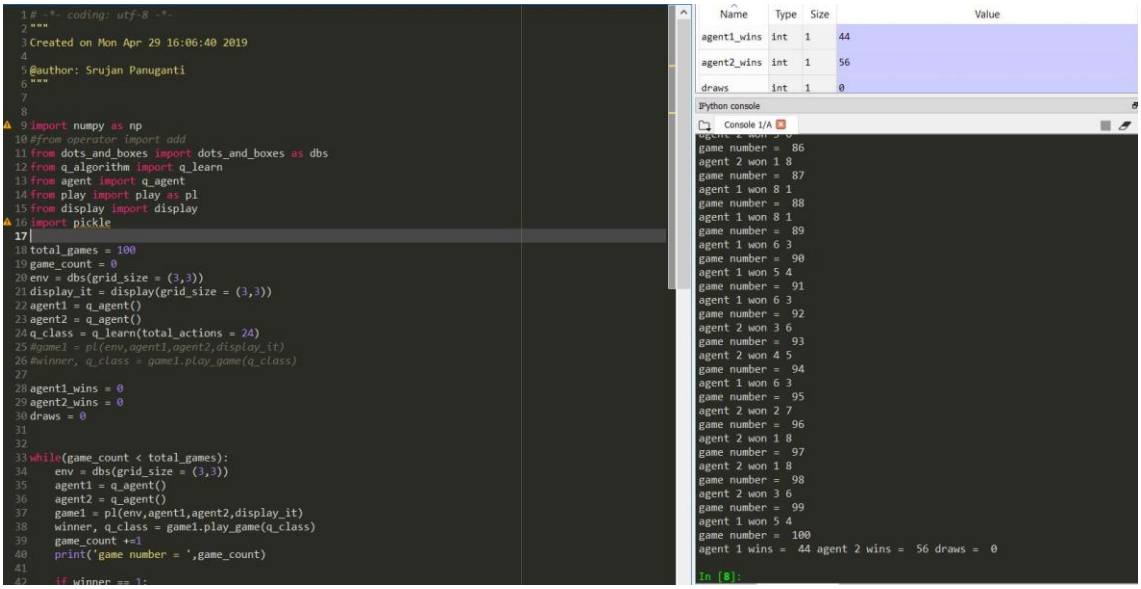


# IMPLEMENTATION OF Q-LEARNING TECHNIQUE – 3 X 3 GRID

**For 100 games – self -Play**

Time consumed: 1 minute 30 seconds

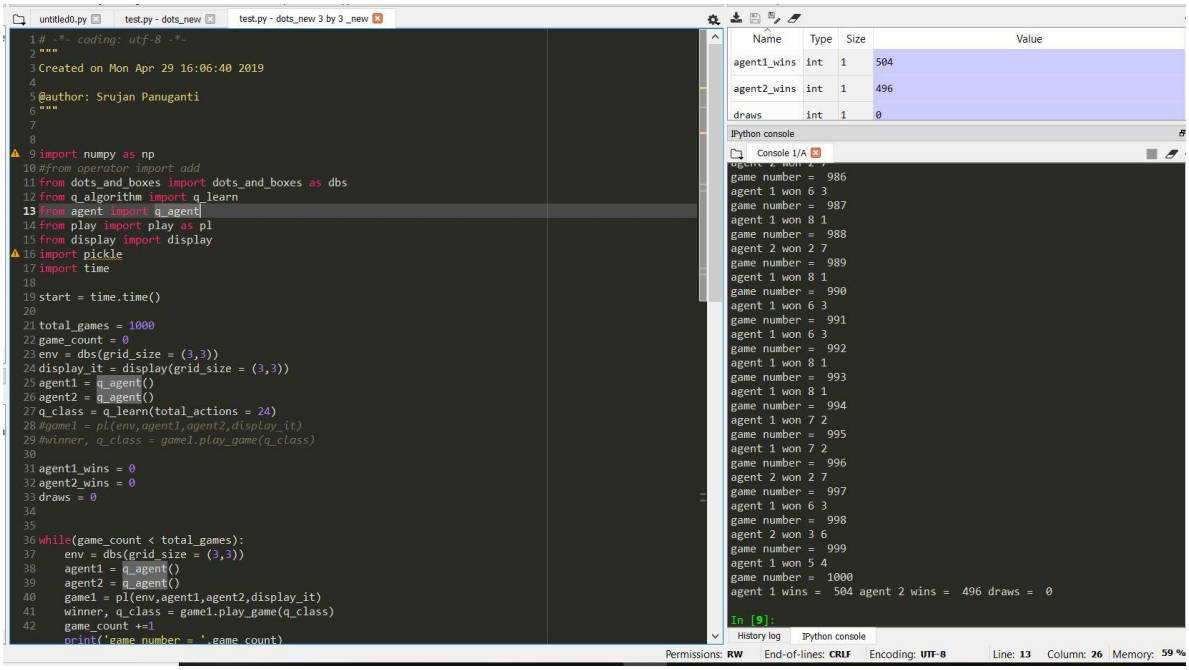
Agent 1 wins : 44 , Agent 2 wins : 56




**For 1000 games – Self Play:**

Time consumed : 14 minutes 10 seconds

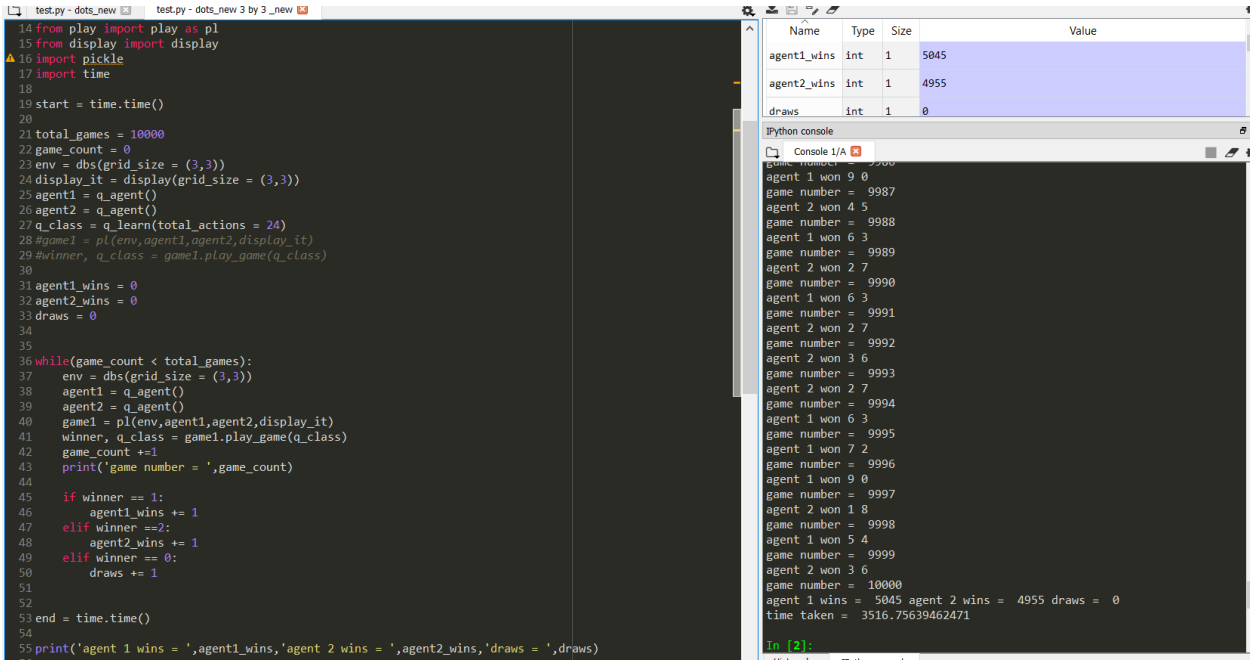
Agent 1 wins = 504 , Agent 2 wins = 496



For 10000 games:

Time consumed = 3516.756 seconds (59.6 minutes)

Agent 1 wins = 5045 , Agent 2 wins  = 4955




## PLAYING WITH RANDOM AGENT 2X2 GRID:

Agent 1 is initialized with q table (based on 100 games experience), where as agent 2 is a random agent

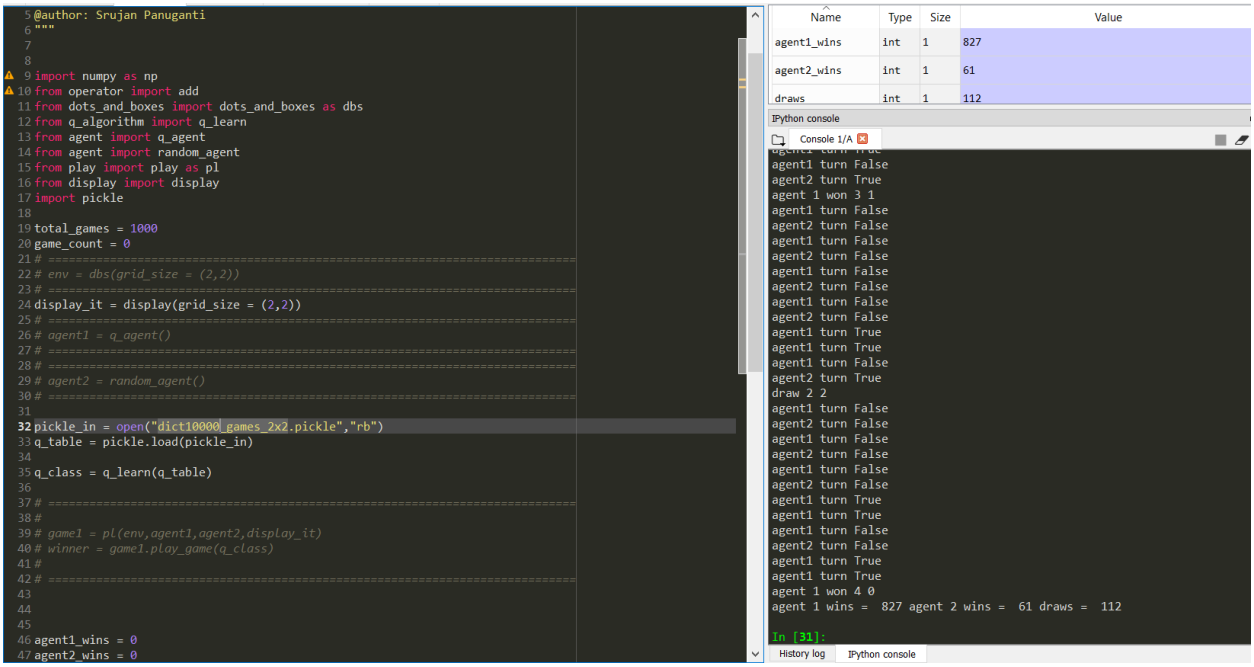
Agent 1 won = 787, agent 2 wins = 77, draws = 136

Agent 1 is initialized with q table (based on 1000 games experience), where as agent 2 is a random agent

Agent 1 won = 804, agent 2 wins = 65, draws = 131

Agent 1 is initialized with q table (based on 10000 games experience), where as agent 2 is a random agent

Agent 1 won = 827, agent 2 wins = 61, draws = 112




## PLAYING WITH RANDOM AGENT 3X3 GRID:

Agent 1 is initialized with q table (based on 100 games experience), where as agent 2 is a random agent

For 100 games, Agent 1 won = 94, agent 2 wins = 6, draws = 0

Time taken = 49 seconds

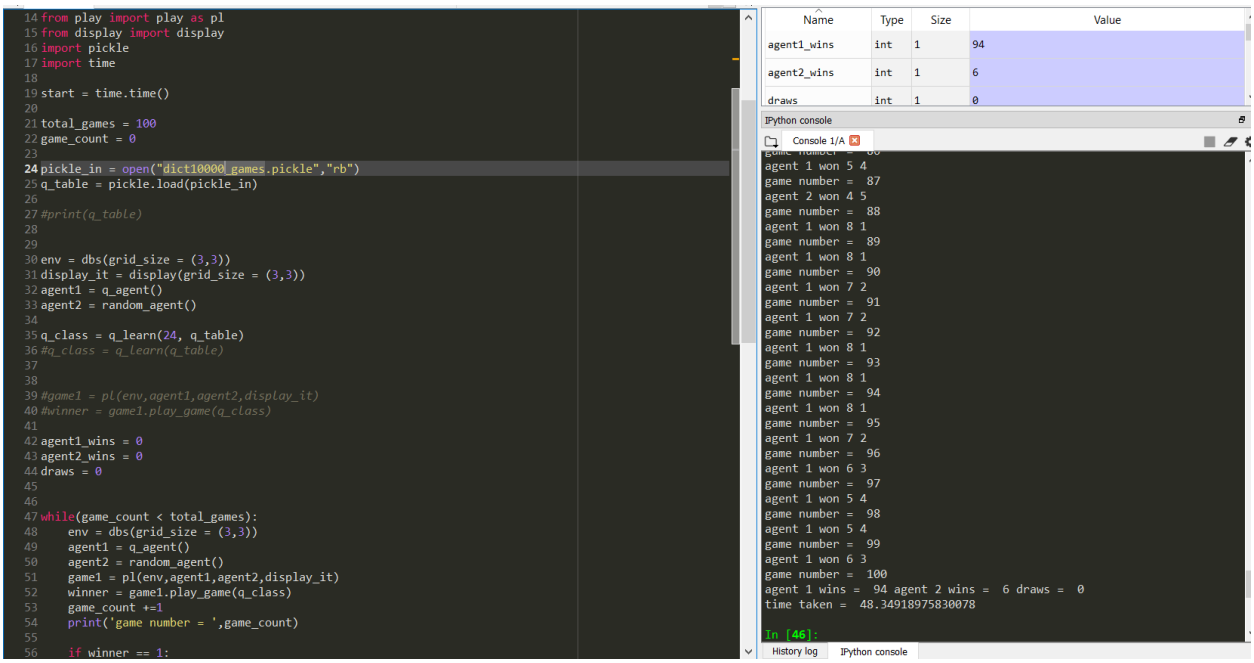
Agent 1 is initialized with q table (based on 1000 games experience), where as agent 2 is a random agent

Agent 1 won = 95, agent 2 wins = 5, draws = 0

Time taken = 50 seconds

Agent 1 is initialized with q table (based on 10000 games experience), where as agent 2 is a random agent

Agent 1 won = 94, agent 2 wins = 6, draws = 0



```
from play import play as pl
from display import display
import pickle
import time

start = time.time()

total_games = 100
game_count = 0

pickle_in = open("dict10000_games.pickle","rb")
q_table = pickle.load(pickle_in)

#print(q_table)


env = dbs(grid_size = (3,3))
display_it = display(grid_size = (3,3))
agent1 = q_agent()
agent2 = random_agent()

q_class = q_learn(24, q_table)
#q_class = q_learn(q_table)


#game1 = pl(env,agent1,agent2,display_it)
#winner = game1.play_game(q_class)

agent1_wins = 0
agent2_wins = 0
draws = 0


while(game_count < total_games):
    env = dbs(grid_size = (3,3))
    agent1 = q_agent()
    agent2 = random_agent()
    game1 = pl(env,agent1,agent2,display_it)
    winner = game1.play_game(q_class)
    game_count +=1
    print('game number = ',game_count)

    if winner == 1:
```

| Name | Type | Size | Value |
|---|---|---|---|
| agent1_wins | int | 1 | 94 |
| agent2_wins | int | 1 | 6 |
| draws | int | 1 | 0 |

IPython console

Console 1/A

```
agent 1 won 5 4
game number =  87
agent 2 won 4 5
game number =  88
agent 1 won 8 1
game number =  89
agent 1 won 8 1
game number =  90
agent 1 won 7 2
game number =  91
agent 1 won 7 2
game number =  92
agent 1 won 8 1
game number =  93
agent 1 won 8 1
game number =  94
agent 1 won 8 1
game number =  95
agent 1 won 7 2
game number =  96
agent 1 won 6 3
game number =  97
agent 1 won 5 4
game number =  98
agent 1 won 5 4
game number =  99
agent 1 won 6 3
game number =  100
agent 1 wins =  94 agent 2 wins =  6 draws =  0
time taken =  48.34918975830078

In [46]:
```

History log | IPython console

## FUNCTION APPROXIMATION FOR 3X3 GRID:

Attempts have made to implement a neural network with 1 input layer, 2 hidden layer and an output layer is implemented to train the game.

Randomly few states are selected from the q-table whose q-values are not equal to zero as the features to train the network. But promising results are not obtained