***SNEHA MORE(Assignment: Connect four)***

*# Imports*from IPython.display

import display, HTML, clear\_output  
import random  
import time  
  
*# Game Constants*ROWS = 6  
COLUMNS = 7  
  
PIECE\_NONE = **' '**PIECE\_ONE = **'x'**PIECE\_TWO = **'o'**PIECE\_COLOR\_MAP = {  
 PIECE\_NONE : **'white'**,  
 PIECE\_ONE : **'black'**,  
 PIECE\_TWO : **'red'**,  
}  
  
DIRECTIONS = (  
 (-1, -1), (-1, 0), (-1, 1),  
 ( 0, -1), ( 0, 1),  
 ( 1, -1), ( 1, 0), ( 1, 1),  
)  
  
*# Board Functions*def create\_board(rows=ROWS, columns=COLUMNS):  
 *''' Creates empty Connect 4 board '''* board = []  
  
 for row in range(rows):  
 board\_row = []  
 for column in range(columns):  
 board\_row.append(PIECE\_NONE)  
 board.append(board\_row)  
  
 return board  
  
 *# Copy board* def copy\_board(board):  
 *''' Return a copy of the board '''* rows = len(board)  
 columns = len(board[0])  
 copied = create\_board(rows, columns)  
  
 for row in range(rows):  
 for column in range(columns):  
 copied[row][column] = board[row][column]  
 return copied  
  
def print\_board(board):  
 *''' Prints Connect 4 board '''* for row in board:  
 print **'|'** + **'|'**.join(row) + **'|'**def drop\_piece(board, column, piece):  
 *''' Attempts to drop specified piece into the board at the  
 specified column  
  
 If this succeeds, return True, otherwise return False.  
 '''* for row in reversed(board):  
 if row[column] == PIECE\_NONE:  
 row[column] = piece  
 return True  
  
 return False  
  
def find\_winner(board, length=4):  
 *''' Return whether or not the board has a winner '''* rows = len(board)  
 columns = len(board[0])  
  
 for row in range(rows):  
 for column in range(columns):  
 if board[row][column] == PIECE\_NONE:  
 continue  
  
 if check\_piece(board, row, column, length):  
 return board[row][column]  
  
 return None  
  
def check\_piece(board, row, column, length):  
 *''' Return whether or not there is a winning sequence starting from  
 this piece  
 '''* rows = len(board)  
 columns = len(board[0])  
  
 for dr, dc in DIRECTIONS:  
 found\_winner = True  
  
 for i in range(1, length):  
 r = row + dr\*i  
 c = column + dc\*i  
  
 if r not in range(rows) or c not in range(columns):  
 found\_winner = False  
 break  
  
 if board[r][c] != board[row][column]:  
 found\_winner = False  
 break  
  
 if found\_winner:  
 return True  
  
 return False  
  
*# HTML/SVG Functions*def display\_html(s):  
 *''' Display string as HTML '''* display(HTML(s))  
  
def create\_board\_svg(board, radius):  
 *''' Return SVG string containing graphical representation of board '''* rows = len(board)  
 columns = len(board[0])  
 diameter = 2\*radius  
  
 svg = **'<svg height="{}" width="{}">'**.format(rows\*diameter, columns\*diameter)  
 svg += **'<rect width="100%" height="100%" fill="blue"/>'** for row in range(rows):  
 for column in range(columns):  
 piece = board[row][column]  
 color = PIECE\_COLOR\_MAP[piece]  
 cx = column\*diameter + radius  
 cy = row\*diameter + radius  
 svg += **'<circle cx="{}" cy="{}" r="{}" fill="{}"/>'**.format(cx, cy, radius\*.75, color)  
  
 svg += **'</svg>'** return svg  
  
**Here are two initial players you can use to test your bot:**  
def HumanPlayer(board, history, players):  
 *''' Read move from human player '''* columns = len(board[0])  
 column = -1  
  
 while column not in range(0, columns):  
 column = input(**'Which column? '**)  
  
 return column  
  
def RandomPlayer(board, history, players):  
 *''' Randomly select a column '''* columns = len(board[0])  
 return random.randint(0, columns - 1)  
  
  
To simulate a game of Connect 4, you may use the following code:  
*# Globals*Players = (PIECE\_ONE, PIECE\_TWO)  
History = []  
Board = create\_board()  
Radius = 40  
Winner = None  
Tries = 0  
  
*# Game Loop*while not Winner:  
 turn = len(History)  
  
 if turn % 2 == 0:  
 move = HumanPlayer(Board, History, Players) *# Player One* else:  
 move = RandomPlayer(Board, History, Players) *# Player Two* if drop\_piece(Board, move, Players[turn % 2]):  
 Tries = 0  
 History.append(move)  
  
 if Tries > 3:  
 print **'Player {} is stuck!'**.format((turn % 2) + 1)  
 break  
  
 clear\_output()  
 display\_html(create\_board\_svg(Board, Radius))  
  
 time.sleep(1)  
  
 Winner = find\_winner(Board)  
  
print **'The Winner is {}'**.format(PIECE\_COLOR\_MAP[Winner])  
  
**Bot:**  
def BotPlayer(board, history, players):  
  *Board: This is the current board.  
 History: This is the history of the previous moves (columns).  
 Players: This is the list of players.  
  
 Your function must not modify any of these objects.  
  
 After analyzing the inputs, your bot should return the column that  
 represents the best possible move.*

***OUTPUT:***

