|  |
| --- |
| # import modules |
|  | import random |
|  | import sys |
|  | import copy |
|  |  |
|  |  |
|  | class Game: |
|  | "Tic-Tac-Toe class. This class holds the user interaction, and game logic" |
|  | def \_\_init\_\_(self): |
|  | self.board = [' '] \* 9 |
|  | self.player\_name = '' |
|  | self.player\_marker = '' |
|  | self.bot\_name = 'TBot' |
|  | self.bot\_marker = '' |
|  | self.winning\_combos = ( |
|  | [6, 7, 8], [3, 4, 5], [0, 1, 2], [0, 3, 6], [1, 4, 7], [2, 5, 8], |
|  | [0, 4, 8], [2, 4, 6], |
|  | ) |
|  | self.corners = [0,2,6,8] |
|  | self.sides = [1,3,5,7] |
|  | self.middle = 4 |
|  |  |
|  | self.form = ''' |
|  | \t| %s | %s | %s | |
|  | \t------------- |
|  | \t| %s | %s | %s | |
|  | \t------------- |
|  | \t| %s | %s | %s | |
|  | ''' |
|  |  |
|  | def print\_board(self,board = None): |
|  | "Display board on screen" |
|  | if board is None: |
|  | print self.form % tuple(self.board[6:9] + self.board[3:6] + self.board[0:3]) |
|  | else: |
|  | # when the game starts, display numbers on all the grids |
|  | print self.form % tuple(board[6:9] + board[3:6] + board[0:3]) |
|  |  |
|  | def get\_marker(self): |
|  | marker = raw\_input("Would you like your marker to be X or Y?: ").upper() |
|  | while marker not in ["X","Y"]: |
|  | marker = raw\_input("Would you like your marker to be X or Y? :").upper() |
|  | if marker == "X": |
|  | return ('X', 'Y') |
|  | else: |
|  | return ('Y','X') |
|  |  |
|  |  |
|  | def help(self): |
|  | print ''' |
|  | \n\t The game board has 9 sqaures(3X3). |
|  | \n\t Two players take turns in marking the spots/grids on the board. |
|  | \n\t The first player to have 3 pieces in a horizontal, vertical or diagonal row wins the game. |
|  | \n\t To place your mark in the desired square, simply type the number corresponding with the square on the grid |
|  |  |
|  | \n\t Press Ctrl + C to quit |
|  | ''' |
|  |  |
|  | def quit\_game(self): |
|  | "exits game" |
|  | self.print\_board |
|  | print "\n\t Thanks for playing :-) \n\t Come play again soon!\n" |
|  | sys.exit() |
|  |  |
|  | def is\_winner(self, board, marker): |
|  | "check if this marker will win the game" |
|  | # order of checks: |
|  | # 1. across the horizontal top |
|  | # 2. across the horizontal middle |
|  | # 3. across the horizontal bottom |
|  | # 4. across the vertical left |
|  | # 5. across the vertical middle |
|  | # 6. across the vertical right |
|  | # 7. across first diagonal |
|  | # 8. across second diagonal |
|  | for combo in self.winning\_combos: |
|  | if (board[combo[0]] == board[combo[1]] == board[combo[2]] == marker): |
|  | return True |
|  | return False |
|  |  |
|  | def get\_bot\_move(self): |
|  | ''' |
|  | find the best space on the board for the bot. Objective |
|  | is to find a winning move, a blocking move or an equalizer move. |
|  | Bot must always win |
|  | ''' |
|  | # check if bot can win in the next move |
|  | for i in range(0,len(self.board)): |
|  | board\_copy = copy.deepcopy(self.board) |
|  | if self.is\_space\_free(board\_copy, i): |
|  | self.make\_move(board\_copy,i,self.bot\_marker) |
|  | if self.is\_winner(board\_copy, self.bot\_marker): |
|  | return i |
|  |  |
|  | # check if player could win on his next move |
|  | for i in range(0,len(self.board)): |
|  | board\_copy = copy.deepcopy(self.board) |
|  | if self.is\_space\_free(board\_copy, i): |
|  | self.make\_move(board\_copy,i,self.player\_marker) |
|  | if self.is\_winner(board\_copy, self.player\_marker): |
|  | return i |
|  |  |
|  | # check for space in the corners, and take it |
|  | move = self.choose\_random\_move(self.corners) |
|  | if move != None: |
|  | return move |
|  |  |
|  | # If the middle is free, take it |
|  | if self.is\_space\_free(self.board,self.middle): |
|  | return self.middle |
|  |  |
|  |  |
|  | # else, take one free space on the sides |
|  | return self.choose\_random\_move(self.sides) |
|  |  |
|  | def is\_space\_free(self, board, index): |
|  | "checks for free space of the board" |
|  | # print "SPACE %s is taken" % index |
|  | return board[index] == ' ' |
|  |  |
|  | def is\_board\_full(self): |
|  | "checks if the board is full" |
|  | for i in range(1,9): |
|  | if self.is\_space\_free(self.board, i): |
|  | return False |
|  | return True |
|  |  |
|  | def make\_move(self,board,index,move): |
|  | board[index] = move |
|  |  |
|  | def choose\_random\_move(self, move\_list): |
|  | possible\_winning\_moves = [] |
|  | for index in move\_list: |
|  | if self.is\_space\_free(self.board, index): |
|  | possible\_winning\_moves.append(index) |
|  | if len(possible\_winning\_moves) != 0: |
|  | return random.choice(possible\_winning\_moves) |
|  | else: |
|  | return None |
|  |  |
|  | def start\_game(self): |
|  | "welcomes user, prints help message and hands over to the main game loop" |
|  | # welcome user |
|  | print '''\n\t----------------------------------- |
|  | \n\t TIC-TAC-TOE by Mawuli Adzaku |
|  | \n\t------------------------------------ |
|  | ''' |
|  | self.print\_board(range(1,10)) |
|  | self.help() |
|  | self.player\_name = self.get\_player\_name() |
|  |  |
|  | # get user's preferred marker |
|  | self.player\_marker, self.bot\_marker = self.get\_marker() |
|  | print "Your marker is " + self.player\_marker |
|  |  |
|  | # randomly decide who can play first |
|  | if random.randint(0,1) == 0: |
|  | print "I will go first" |
|  | # self.make\_move(self.board,random.choice(self.corners), self.bot\_marker) |
|  | #self.print\_board() |
|  | self.enter\_game\_loop('b') |
|  | else: |
|  | print "You will go first" |
|  | # now, enter the main game loop |
|  | self.enter\_game\_loop('h') |
|  |  |
|  |  |
|  | def get\_player\_move(self): |
|  | move = int(input("Pick a spot to move: (1-9) ")) |
|  | while move not in [1,2,3,4,5,6,7,8,9] or not self.is\_space\_free(self.board,move-1) : |
|  | move = int(input("Invalid move. Please try again: (1-9) ")) |
|  | return move - 1 |
|  |  |
|  | def get\_player\_name(self): |
|  | return raw\_input("Hi, i am %s" % self.bot\_name + ". What is your name? ") |
|  |  |
|  |  |
|  | def enter\_game\_loop(self,turn): |
|  | "starts the main game loop" |
|  | is\_running = True |
|  | player = turn #h for human, b for bot |
|  | while is\_running: |
|  | if player == 'h': |
|  | user\_input = self.get\_player\_move() |
|  | self.make\_move(self.board,user\_input, self.player\_marker) |
|  | if(self.is\_winner(self.board, self.player\_marker)): |
|  | self.print\_board() |
|  | print "\n\tCONGRATULATIONS %s, YOU HAVE WON THE GAME!!! \\tn" % self.player\_name |
|  | #self.incr\_score(self.player\_name) |
|  | is\_running = False |
|  | #break |
|  | else: |
|  | if self.is\_board\_full(): |
|  | self.print\_board() |
|  | print "\n\t-- Match Draw --\t\n" |
|  | is\_running = False |
|  | #break |
|  | else: |
|  | self.print\_board() |
|  | player = 'b' |
|  | # bot's turn to play |
|  | else: |
|  | bot\_move = self.get\_bot\_move() |
|  | self.make\_move(self.board, bot\_move, self.bot\_marker) |
|  | if (self.is\_winner(self.board, self.bot\_marker)): |
|  | self.print\_board() |
|  | print "\n\t%s HAS WON!!!!\t\n" % self.bot\_name |
|  | #self.incr\_score(self.bot\_name) |
|  | is\_running = False |
|  | Break |
|  | else: |
|  | if self.is\_board\_full(): |
|  | self.print\_board() |
|  | print "\n\t -- Match Draw -- \n\t" |
|  | is\_running = False |
|  | #break |
|  | else: |
|  | self.print\_board() |
|  | player = 'h' |
|  |  |
|  | # when you break out of the loop, end the game |
|  | self.end\_game() |
|  |  |
|  | def end\_game(self): |
|  | play\_again = raw\_input("Would you like to play again? (y/n): ").lower() |
|  | if play\_again == 'y': |
|  | self.\_\_init\_\_() # necessary for re-initialization of the board etc |
|  | self.start\_game() |
|  | else: |
|  | print "\n\t-- GAME OVER!!!--\n\t" |
|  | self.quit\_game() |
|  |  |
|  |  |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | TicTacToe = Game() |
|  | TicTacToe.start\_game() |