Python Programming Sections 1832 & 1833

Deadline October, 27 2020 by 23:59 PM PST

Programming Assignment 3 (100 Points): Tic-Tac-Toe (updated October 16, 2020)

Assignment Description:

In assignment 3 you will create a Python tic-tac-toe game where a single player competes against the computer. For this assignment you will need to write several Python functions, use a game-board (a Python list of lists) to store the game state and use random to control the computer's behavior. The computer always moves first and is **O's**.

Program Description, Interfaces, and Functionality

To create a Python tic-tac-toe game, use Python's **randint** function from the random module to control the computer's moves. To determine if the game has been won or is a tie/draw, write at least the functions: `def move()`, `def check_cols()`, `def check_diag1()`, `def check_diag2()`, `def check_rows()`, and `def print_board()`. Four of the functions will check if a winning state exist. To win tic-tac-toe, a user needs to have three of the same game pieces, either an **X** or an **O**, in a row, column, or diagonal. When either the player or computer wins, the game will announce a winner. Use the provided functions `def check_ifwin()` to determine if the board is in a winning state. Refer to the example game play in this handout.

Create an **A03.py** file to submit to Canvas with the following:

1. Comments at the top of the **.py** file as shown below
2. The `def move()` function will make a computer move using the random Python module. Refer to the skeleton code for how to move the computer.
3. The functions: `def check_cols()`, `def check_diag1()`, `def check_diag2()`, `def check_rows()`, will traverse the board to check for winning states. Refer to the skeleton code for more details.
4. The `def print\_board()` function will print out the current state of the board. Refer to the example game play for the look-and-feel of the game.
5. The game of tic-tac-toe will continuously play until either the computer or player wins or the game is a draw. A player can then decide to play another round or exit the game.

Where to do the assignment

You can do this assignment on your own computer, or from the SMC Virtual labs via Citrix. In either case, ensure the code runs on Windows and in IDLE. Submit one **.py** file named **A03.py**. Do not use any other name or else points will be deducted.

Submitting the Assignment

Include your name, your student id, the assignment number, the submission date, and a program description in comments at the top of your files. Submit the assignment on Canvas (`https://online.smc.edu`) by **uploading your .py file** to the Assignment 3 entry as an attachment. Do not cut-and-paste your script into a text window. Do not hand in a screenshot of your program's output. Do not hand in a text file containing the output of your program. Do not save and turn in the interpreter session (e.g. the one with >>>).

Saving your work

Save your work often on a flash-drive or to the cloud (e.g., GoogleDrive, Microsoft OneDrive, Canvas, etc.). Always save a personal copy of your files (e.g. .py, etc.). Do not store files on the virtual lab computers.

Example tic_tac_toe() usage:

```
>python A03.py
---
O--
---
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 11
---
OX-
O--
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 00
X--
OX-
OO-
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 12
We have a winner!
X--
OXX  #example computer wins
OOO
Do you want to try again? enter yes or no: yes
---
---
--O
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 11
---
OX-
--O
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 20
---
OX-
XOO
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 02
We have a winner!
--X
OX-  #example player wins
XOO
Do you want to try again? enter yes or no: yes
O--
---
---
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 11
O--
-X-
O--
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 22
O--
-XO
O-X
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 02
OOX
-XO
O-X
Enter in cell for your move as a coordinate pair (e.g. 00, 10, etc): 10
Better luck to both of you next round
OOX
XXO  #example tie/draw
OOX
```

Skeleton code

```
from random import randint

#void function to print the board state
def print_board(board):
    #prints the current board
    print("the board as rows and columns")

# Boolean function to check rows for a win
def check_rows(board):
    # X X X    O O O <-check each row for a win
    #
    # if a win, return true
    # else return false
    #
    return True # or False

# Boolean function to check cols for a win
def check_cols(board):
    # X      O  <-check each column for a win
    # X  or  O
    # X      O

    # if a win, return true
    # else return false
    return True # or False

# Boolean function to check diagonal 1 for a win
def check_diag1(board):
    # X        O     <-check diagonal 1 for a win
    #   X   or   O
    #     X        O

    # if a win, return true
    # else return false
    return True # or False

# Boolean function to check diagonal 2 for a win
def check_diag2(board):
    #     X        O <-check diagonal 2 for a win
    #   X   or   O
    # X        O

    # if a win, return true
    # else return false
    return True # or False
```

Skeleton code continued:

```
# Void function to control the computer's moves uses randint for next move
def move(board):
    # Tip: a 2D coordinate can be computed from
    # a number n using integer division and
    # modulo arithmetic
    #
    # For example, the coordinate pairs
    # (xy) 00,01,10,11 for a 2x2 grid with
    # four cell numbers (c) 0,1,2,3 can be
    # computed from the cell number c as:
    #
    # c = randint(0,3)
    # x = c // 2; y = c % 2
    print("move")

# Function to check if there is  a winning state on the board
def check_ifwin(board):
    if (check_rows(board)):
        return True

    if (check_cols(board)):
        return True

    if (check_diag1(board)):
        return True

    if (check_diag2(board)):
        return True

    return False
```

Skeleton code continued

```
# Game function of tic-tac-toe
def tic_tac_toe():
    while True:
        # Create the game-board, list of lists
        b= [[0,0,0],[0,0,0],[0,0,0]]

        # Todo A3:
        # initialize the board to an empty game piece. For example, '-'
        #  - - -
        #  - - -
        #  - - -

        # keep track of moves made, computer can move max 5 times
        total_moves = 0

        # Play the game
        while (not (check_ifwin(b))):

            # Computer moves first/last
            move(b)
            print_board(b)

            # Todo A3: check if computer won or reached max moves
            # if (     TODO  A3     )
                #break

            #
            # Human move
            # A3 optional todo: check if cell occupied by computer
            # before making the move, not implemented in skeleton code
            #
            r = input("Enter in cell for your move as a coordinate pair (e.g.00,10, etc):
            x = int(r[0]); y = int(r[1]) b[x][y] = 'X';

            # total number of moves made
            total_moves+=1

        # Check for winning state
        if (not check_ifwin(b)):
            print("Better luck to both of you next round")
        else:
            print("We have a winner!")

        # Print the game-board
        print_board(b);

        r = input("Do you want to try again? enter yes or no\n")
        if(r == "no"):
            break

    input("Enter any key to exit")

#play the game
tic_tac_toe()
```