Lab 2: Tic Tac Toe

Due: Wednesday, March 2, 2022, 11:59 EST

Everyone enjoys a sporting game of tic tac toe. Tic Tac Toe is a 2-player game where you attempt to get 3 in a row before your opponent in a 3x3 board (https://playtictactoe.org/).

Please read the instructions carefully and completely before doing any work, having a solid understanding of the expectations of the lab (and all the hints in the instructions) will make the assignment much more manageable.

In this lab, we will develop:

- A working tic tac toe simulation
- A minimal interface
- A simple AI

Outcomes:

We will approach creating this simulation methodically by thinking of the requirements to make a working simulation, building a skeleton, and modularly creating each feature.

Take specific notice of our focus on modularity, we want to be able to debug more effectively and easily make our code work in similar games. This is an important skill and the autograder will test each component method individually.

Writing almost any application uses the same design thinking.
You will be exposed to creating a working and modularized project using object-oriented thinking.

First, before coding anything, we want to set up the design of our system.
We model the game of tic tac toe like this:

$[[0,0,0],$
$[0,0,0],$
$[0,0,0]]$

When a player takes a square, we can change its value to "1" or "2" depending on which player moved there. You must use 0, 1, and 2 for the autograder to successfully test your code.

We can create a clean project by using functions to do each of the repeated tasks.
For the bare minimum of the game, we need to do each of the following:

- Make a move
- Check if a move is valid
- Check if someone has won
- Get available squares
- Ask the player to make a move

We will also want to have state variables that can store information needed for the game to function.

- Whose turn is it
- The Board

Part 0: Making a Plan

The following parts will guide you through the steps to write the program, but before you dive in, you have to have a game plan.

Draw on paper how your program should work, you can use any shapes, lines, and words that make sense to you. You should include the step-by-step logical flow of your program starting with something like this: Initialize game -> Take Turn -> Show Board -> Ask Player to make move -> etc.

Part 1: Simulating the Game

1. Download the file tic-tac-toe.py
2. Examine the skeleton code, you may not include any modules other than random
3. Write the init method. The $\_\_init\_\_$ method must include the following variables:
    - self.board
        - It must be a list consisted of 3 lists of 3 zeros.
        - After finishing this method, you can check your work by **printing** sim.board, you should see this: [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
    - self.AI
        - self.AI must be a boolean type
        - True or False of whether or not playing against an AI
    - self.turn
        - self.turn must be a integer type of strictly the numbers 1 and 2
        - By default unless the player is playing against an AI, self.turn $= 1$
        - This variable is used to determine the state of the game (i.e. which player is making the move)
    - self.AI_turn
        - self.AI_turn is only used when you are playing against an AI

  - Like self.turn, must be either 1 or 2 and should be able to help you determine who goes first or second when playing against an AI
  - Depending on this input set, self.AI_turn determines whether the AI goes first (=1) or second (=2)
  - Note: This variable should not change

4. Write the play_game() method:
   - For this method, you will be simulating the entire game. For now, just have an infinite while loop that will call print_board(), take_turn(), and change_turn(). We will write the logic for these methods later.
5. Write the change_turn() method
6. You should be able to initialize the board using: sim = TicTacToeSim()

Part 2: Showing the State of the Game and Debugging

- Write a **print_board** method
  - Go through the board list and print a grid
- You should now be able to show the board using sim.print_board()
- It should just show an empty 3x3 board
- This method does not change the actual board, it just prints a readable version that changes the 0s, 1s, and 2s to empty spaces, Xs, and Os.

Part 3: Getting Player Input

- Ask the player for a row and column
- Return a tuple (row, col)
- If you test with sim.get_move()

Part 4: Taking Turns and Validating Inputs

- Complete three methods that accomplish the following:
  - To take a turn (take_turn())
    - First ask the player for input
    - Then check whether the move is valid
    - While invalid (ask again)
    - If valid, make the move
  - To validate input (get_available_squares())
    - Go through every row and column of the board (nested for loop)

        * Check if the pair is taken
        * Return a list of tuples of available squares
    - Making a move (make_move())
        * Take as input a move
        * Unpack the tuple to get the row and column
        * Mark that square on the board by the current player

- You can now use the methods you've written so far in the play_game() method to simulate the game. To check your work make sure to call sim.play_game()

Part 5: Winning the Game

- Checking for winning is the hardest part of the assignment
- Check if any rows, columns, or diagonals are all occupied by the same player
- Return the player if they won and None if nobody has won
- Now replace the while loop in the play_game() method to check if the game has been won and if it has been won, to print the winner and stop the loop
- After this part, you should be able to play against yourself with a working tic tac toe game

Part 6: Random Moves "AI"!

- First, we need to add an AI in the init and set to True for testing.
- When making a move, if player = self.AI_turn and self.AI is True, then use random_move() to make the move instead of asking for input
- Let's examine the random_move() method
    - Here, we return a random value from the available squares
    - This has been provided for you in the starter code.
- Now when you play the game, player 2 should make random moves

Part 7: Can Win and Can Lose

- Let's improve the AI by checking if it can win this turn or lose next turn using the method smart_move() that calls two different methods.
- **If** it can win, then win (using the winning_move() method)
    - This is a hard method and similar to check_winner(), but needs to use the available moves and check if it will create a 3 in a row
    - Return the winning move if applicable.
- **If** it can lose, then block a move (using the threat_to_lose() method)

  - You should attempt to reuse the winning_move() method by calling it with the opponent's turn.
  - Return the blocking move if applicable.

- **Else**, Return a random move
- Now the AI should be able to win if it can and stop you from winning (Thinking one move ahead)

Part 8: Settings

For this section, you are asking the player for the settings of the game. The settings that you will be asking for is the following (and in this exact order):

- If they would like to play against an AI (Input has to be 'True' or 'False')
- If they are playing against the AI, ask them whether or not they would like to be player 1 or 2 (Input has to be '1' or '2')
- Modify the get_settings method and use it at the beginning of the game.
- get_settings is a void method and should not return a value, but it does modify values in the class.
- You must set both the AI and AI_turn fields

Part 9 (Extra Credit)

- Get started early (+2 points)
  - If you submit a serious attempt (get at least 50 points from the autograder) completing the first 5 parts on **Wednesday, February 23, 2022 11:59PM EST**, you will receive 2 points extra credit. (This is referring to the current active submission)
- Make Tic Tac Toe on a nxn board (+10 points)
  - For advanced students
  - Create a completely new file called tic-tac-toe-n.py
  - The winner is whoever gets a row, column, or diagonal of length n
  - The constructor should include a parameter n to determine the size
  - You should only modify a few methods and when 3 is passed for n your code should still be able to pass all test cases

Submission Instructions

Submit 2 (or 3) files:
tic-tac-toe.py
tic-tac-toe-n.py if you completed the extra credit.
README.pdf with any notes on the lab and your explanation from Part 0

Submit all parts on Gradescope.

Make sure to comment out or remove all of your own testing. It will mess with the autograder because of the input() functions.

At the top of the README and tic-tac-toe-n, include the following information:

- Your name
- The name of any classmates you discussed the assignment with, or the words "no collaborators"
- A list of sources you used (textbooks, wikipedia, research papers, etc.) to solve the assignment, or the words "no sources"
- Whether or not you're using the extension option

Grading Methodology

Each part of the lab will be graded by an automatic grading problem. It will use the method signatures specified in this lab and will use multiple test cases. Each phase of the lab will be graded independently (8*10 each) and together in its entirety (10).

The remaining 10 points are allocated for proper comments and styling:

- Descriptive variable names
- Comments for what functions do
- Comments for complex parts of code
- Proper naming conventions for any variables, functions, and classes