15-110: Principles of Computing, Spring 2018

Programming Assignment 10

Due: Tuesday, April 17 by 9PM

IMPORTANT ANNOUNCEMENT
You cannot drop this assignment even if it is your lowest PA score.
Failure to submit this assignment on time will result in a 0 which will be included in your PA total.
We only drop the lowest PA score from PA1-PA8. Please submit whatever you can before the deadline, even if it is not completely done.

Note: You are **responsible for protecting your solutions** to the following problems from being seen by other students both physically (e.g., by looking over your shoulder or verbal discussion) and electronically. In particular, since the lab machines use the Andrew File System (AFS) to share files worldwide, you need to be careful that you do not put files in a place that is publicly accessible.

If you are doing the assignment on the Gates-Hillman Cluster machines we use in the lab or on `unix.andrew.cmu.edu`, please remember to have your solutions inside a `private` folder (which is under your home directory). Our recommendation is that you create a `pa10` folder under `~/private/15110` for this assignment. That is, the new directory `pa10` is inside the directory named `15110`, which is inside the `private` directory.

Setup

For this assignment, you will create a Python file for each of the problems below. You should save all of these files in a folder named `pa10`. Once you have every file, you should zip up the `pa10` folder and submit the zipped file on Autolab.

This assignment will help you put the principles you learned in class to creating a new simulation.
NOTE: If you use your own laptop and have Python 3 installed, you should have the tkinter module available. See a TA if you wish to install Python 3 on your laptop. If you ssh into the Andrew servers, tkinter is now available to import, but you need to use the –X option when connecting.

Overview

In this assignment, you will create a Python program to allow two players to play a game of Connect Four, a popular family game.

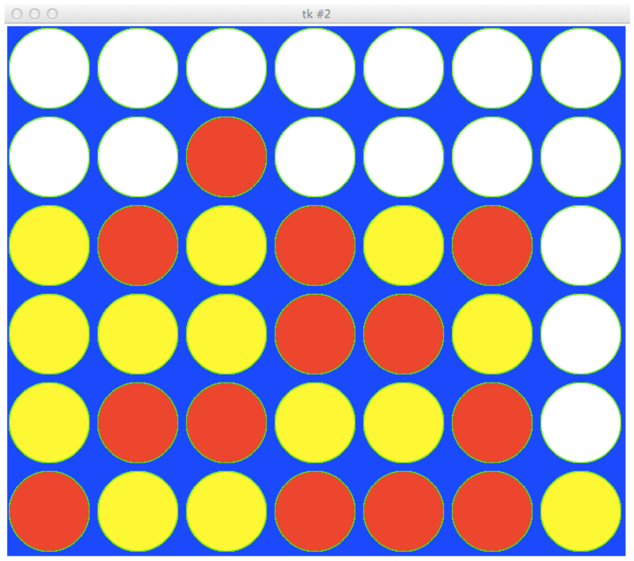
The vertical game board consists of 7 columns that can hold up to 6 discs. Two players (1 and 2) alternate dropping a colored disc (red and yellow, respectively) into one of the columns, with each disc falling as far as it can go. The first player to get four of the same color in a row (vertically, horizontally or diagonally) wins the game. If neither player can do this after all disc locations are filled, the game ends in a tie.

In this game, the players are represented with the numbers 1 (Red) and 2 (Yellow). The game board is represented internally as a matrix (i.e. list of lists) with 6 rows and 7 columns. In each cell of this matrix, we store a 0 if the corresponding board position has no disc, a 1 if there is a red disc in that position, or a 2 if there is a yellow disc in that position. Initially, the matrix contains only 0's since no discs have been played yet. Rows are numbers 0 through 5 from top to bottom, and columns are numbered 0 through 6 from left to right.

To display the board, we will use a window with a Canvas with a width of 700 pixels and a height of 600 pixels. The window is broken up into 42 square regions of size 100 X 100, forming 6 rows and 7 columns. In each square, we draw a circle centered in the square region with a diameter of 90 pixels, filled in with either white (empty), red or yellow, depending on the value stored in the equivalent position in the matrix used to represent the game board. Each circle should have an explicit <u>green</u> outline.

For example, here is the matrix for the game board shown at right.

```
[[0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0],
 [2, 1, 2, 1, 2, 1, 0],
 [2, 2, 2, 1, 1, 2, 0],
 [2, 1, 1, 2, 2, 1, 0],
 [1, 2, 2, 1, 1, 1, 2]]
```



NOTE: If you have any issues with colorblindness or sight concerns, please contact your instructor for accommodations.

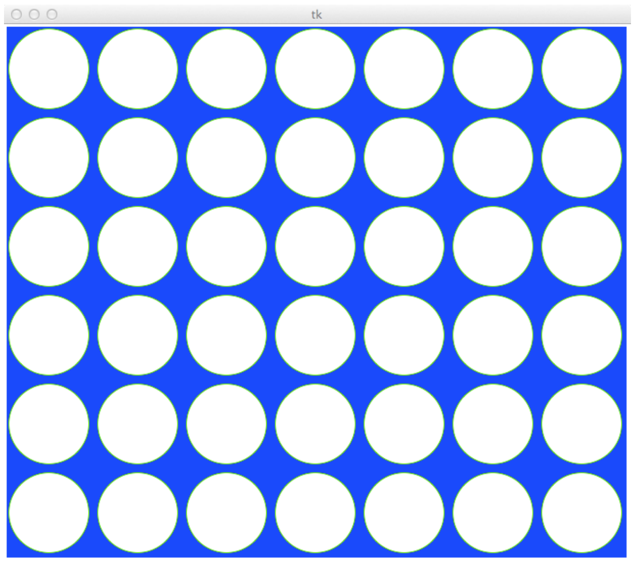
Problems

Start by downloading a copy of the file `connect4.py` into your `pa10` folder from the course website. Read through the first two functions we give you to see how they work. You are responsible for understanding all of the code we've given you. The main function that you run to play the game is called `play()`.

You will see that the file we give you has all of the functions represented as "stubs". This means that each function is written so that it returns something so that the entire program will run without crashing if you haven't completed all of the functions. As you complete each function, replace the code we give you for the function in question with your own code, but leave the subsequent stubs in place until you have tested your current function(s) completely.

1. (1 point) Complete the function `new_board()` (in the file `connect4.py`). This function should create and return a matrix, i.e. list of lists, with 6 rows and 7 columns with all values set to 0. This will represent a new game board at the start of a game (i.e. no players have played yet).

2. (1 point) Complete the function `display_board(c, board)` (in the file `connect4.py`). This function has two parameters: `c` represents the canvas that is created in the `play` function that displays the game and `board` represents the 6 X 7 matrix storing what is in each position of the game board (0, 1, or 2). This function should draw the visual representation of the current game board on the canvas. Remember that each circle should have a diameter of 90 pixels, not 100, and be centered in each row and column. There should be some blue background around each circle. The circles should not touch. They should be separated by 10 pixels.

   TESTING YOUR WORK: Once you finish this function, you should be able to run the program by calling the function `play()` in the `python3` interpreter. You should see an empty board with 42 white circles. You can try to enter moves, but none will be accepted.

   ```
   > python3 -i connect4.py
   >>> play()
   ```

   Once you have this completed, copy your current solution into `connect4_v1.0.py` in the `pa10` folder at the command line prompt after leaving the `python3` interpreter. Do not change this copy.

   ```
   cp connect4.py connect4_v1.0.py
   ```

3. (3 points) Complete the function `add_piece(board, player, column)` (in the file `connect4.py`). This function has three parameters: `board` represents the 6 X 7 matrix storing what is in each position of the game board (0, 1 or 2), `player` represents the number of the current player who is dropping a disc (1 or 2), and `column` represents the column where the disc is being dropped (an integer between 0 and 6, inclusive). This function should update the board to the state it would be in after the disc has been dropped down the given column (if possible).

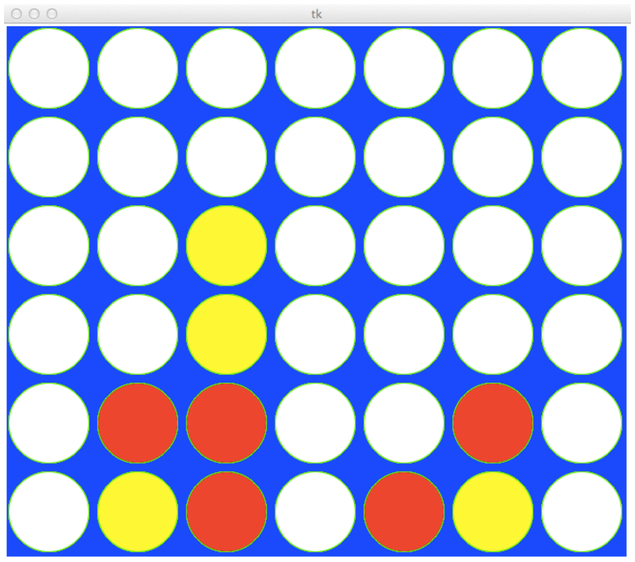
   The basic idea is to scan the given column from the bottom to the top for the first empty (0) cell. You will then "drop" the disc into that cell by storing the player's number in that location. If the entire column is full, then you will return `None` and leave the board as is.

   General algorithm:

```
I. For each row from 5 down to 0, in the given column, do the following:
   A. If board[row][column] is 0 (i.e. empty), then do the following:
      1. Store player in board[row][column].
      2. Return the row where the disc stopped in that column.
II. If every row is tested and none work, return None.
```

   TESTING YOUR WORK: Once you finish this function, you should be able to run the program by calling the function `play()` in the `python3` interpreter and play the game. You should be able to drop discs (alternating red and yellow), but the game will not test to see if anyone wins yet.

```
> python3 -i connect4.py
>>> play()
Player 1: Which column (0-6, quit)? 2
Player 2: Which column (0-6, quit)? 1
Player 1: Which column (0-6, quit)? 2
Player 2: Which column (0-6, quit)? 2
Player 1: Which column (0-6, quit)? 1
Player 2: Which column (0-6, quit)? 2
Player 1: Which column (0-6, quit)? 4
Player 2: Which column (0-6, quit)? 5
Player 1: Which column (0-6, quit)? 5
Player 2: Which column (0-6, quit)? quit
[[0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0], [0, 0, 2, 0, 0, 0, 0], [0,
0, 2, 0, 0, 0, 0], [0, 1, 1, 0, 0, 1,
0], [0, 2, 1, 0, 1, 2, 0]]
```



   Once you have this completed, copy your current solution into `connect4_v2.0.py` in the `pa10` folder at the command line prompt after leaving the `python3` interpreter. Do not change this copy.

```
cp connect4.py connect4_v2.0.py
```

4. (1 point) Complete the function `check_win_vertical(board, row, column)` (in the file `connect4.py`). This function has three parameters: `board` represents the 6 X 7 matrix storing what is in each position of the game board (0, 1, or 2), and `row` and `column` represents where the most recent disc was placed. This function should return `True` if the player who placed this disc has won vertically, or `False` otherwise.

   To test if the player wins vertically, you only have to count the number of discs of the same player below the newly placed disc until you either find a different player or you reach the bottom of the game board. None of the locations below the newly placed disc can be empty. If your count gets to 4, that player has won.

   General algorithm:

   ```
   I. Set $player$ equal to $board[row][column]$.
   II. Set $count$ equal to 1. (Count the newly dropped disc.)
   III. Set $r$ equal to $row$+1. (Start in the next row, same column.)
   IV. While $r$ is less than or equal to 5 and $board[r][column]$ is equal to $player$, do the following:
       A. Add 1 to $count$.
       B. Add 1 to $r$ to move to next row.
   V. If $count$ is equal to 4, return True. Otherwise return False.
   ```

   TESTING YOUR WORK: Once you finish this function, you should be able to run the program by calling the function `play()` in the `python3` interpreter and play the game. You should be able to drop discs (alternating red and yellow) and detect a player who wins vertically.

   Once you have this completed, copy your current solution into `connect4_v3.0.py` in the `pa10` folder at the command line prompt after leaving the `python3` interpreter. Do not change this copy.

   ```
   cp connect4.py connect4_v3.0.py
   ```

5. (2 points) Complete the function `check_win_horizontal(board, row, column)` (in the file `connect4.py`). This function has three parameters: `board` represents the 6 X 7 matrix storing what is in each position of the game board (0, 1, or 2), and `row` and `column` represents where the most recent disc was placed. This function should return `True` if the player who placed this disc has won horizontally, or `False` otherwise.

   This function works similarly to the previous function, except that you have to count the number of adjacent discs of the same color both left and right of the most recently placed disc to see if the complete total is <u>at least</u> 4. Make sure you understand how the algorithm works.

   General algorithm:

   ```
   I. Set $player$ equal to $board[row][column]$.
   II. Set $count$ equal to 1.
   ```

continued on next page

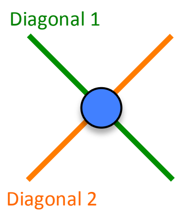
```
III. Set c equal to column + 1.
IV. While c is less than or equal to 6 and board[row][c] is equal to player, do the following:
    A. Add 1 to count.
    B. Add 1 to c.
V. Set c equal to column - 1.
VI. While c is greater than or equal to 0 and board[row][c] is equal to player, do the following:
    A. Add 1 to count.
    B. SUBTRACT 1 FROM c.   ← CORRECTION
VII. If count is at least 4, return True. Otherwise return False.
```

TESTING YOUR WORK: Once you finish this function, you should be able to run the program by calling the function `play()` in the `python3` interpreter and play the game. You should be able to drop discs (alternating red and yellow) and detect a player who wins vertically or horizontally.

Once you have this completed, copy your current solution into `connect4_v4.0.py` in the `pa10` folder at the command line prompt after leaving the `python3` interpreter. Do not change this copy.

```
cp connect4.py connect4_v4.0.py
```

6. (2 points) Complete the functions `check_win_diagonal1(board, row, column)` and `check_win_diagonal2(board, row, column)` (in the file `connect4.py`). Each function should return `True` if the player who placed the disc has won diagonally, or `False` otherwise. There are two functions since there are two diagonal directions. Each function should test one diagonal direction. You will need to figure out how to write these functions, but they are similar to the functions you wrote above.



TESTING YOUR WORK: Once you finish this function, you should be able to run the program by calling the function `play()` in the `python3` interpreter and play the entire game.

Once you have this completed, copy your current solution into `connect4_v5.0.py` in the `pa10` folder at the command line prompt after leaving the `python3` interpreter. Do not change this copy.

```
cp connect4.py connect4_v5.0.py
```

Submission

You should now have the `pa10` folder that contains (up to 5) completed versions of the game:

`connect4_v1.0.py` `connect4_v2.0.py` `connect4_v3.0` *etc.*

We will grade the highest version number for each submission you make. You may have additional files but these will not be graded. Zip up the folder and submit the zipped file named as `pa10.zip` on Autolab. You may submit as you get each step done to get partial credit. Don't wait until the end to submit everything. Be sure to check your submission to see that you submitted the correct code each time.