Game of Connect 4

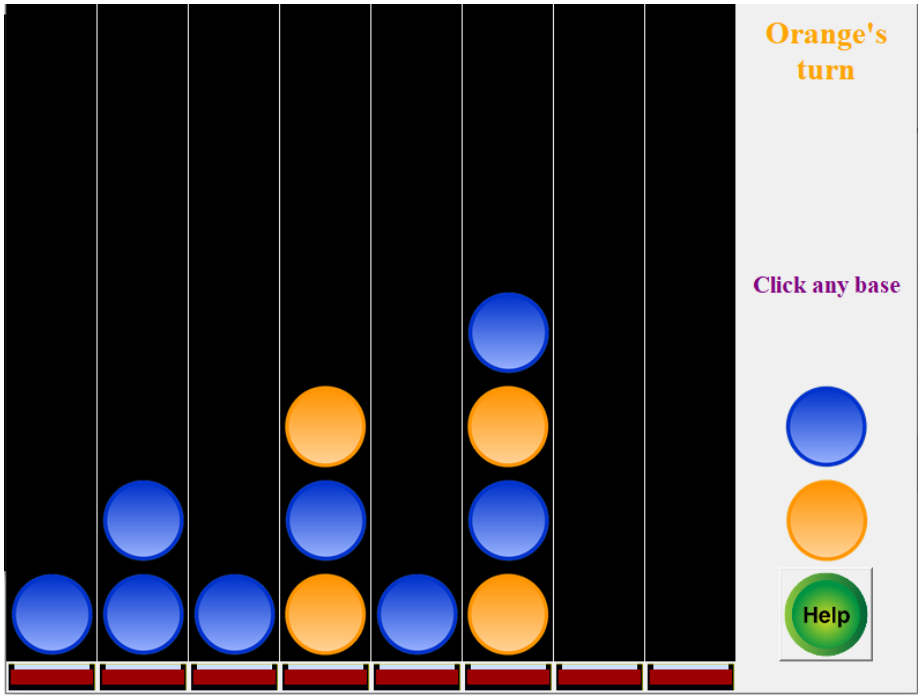
Connect Four is a 2-player game which consists of two sets of colored coins and a standing grid of rows and columns. Each player takes one set of coins and then by turn drops coins down any of the vertical columns (we will call them “tubes”). See the picture below.

The goal of the game is to get 4 coins of the same color to arrange themselves along a row, column, or diagonal. The first player to do this wins the game.

If you want to play with my final program to get a feel for this game, click the link given at the end of the article. Try not to peek at the scripts yet, since we want to design them ourselves below. Click here to download all program files.

How to run the program:

1. Run the program file. Enter "h" to view help.
2. You (the human player) begin with the "Orange" coin and the computer plays with "Blue" coins. The variable “Turn” shows whose turn it is.
3. The computer gets to drop 2 coins at every turn.
4. Click the base of the tube in which you want to drop your coin.



Python and CS Concepts Used

When we design this program, we will make use of the following Python and CS concepts. Learn these concepts if you don't know them before proceeding further.

Main concepts:

- Algorithms
  - Abstraction
  - Designing new algorithms
  - Pseudo-code
- Arithmetic
  - Expressions
  - Basic operators (+, -, *, /)
- Concurrency
  - Multi-threading for timed callback
- Conditional statements:
  - Conditions: YES/NO questions
  - Relational operators (==, <, >, etc.)
  - Conditionals (IF)
  - Conditionals (If-Else)
  - Conditionals (nested IF)
  - Boolean operators (and, or, not)
- Data structures – list
  - List operations

    - Using list as 2-D array
    - List traversal
    - List comprehension
- Data types – strings
    - String operations
- Looping (iteration)
    - Looping – simple, with counter (for)
    - Looping - conditional (while)
- Procedures
        - User defined (custom)
        - Procedures with parameters and return value
- Program output
    - Text
    - GUI (Tkinter)
- Random numbers
- Sequence
- User interface elements
        - Label
        - Button
        - Grid
- User input
    - Click labels and buttons
- Variables
    - Simple
    - Local/global scope

High Level Design:

This is where we take a step back from the computer (literally!), analyze the problem in our mind (and on a piece of paper if necessary), and break it down into multiple smaller ideas which can be programmed separately.

Console (text) Version: High Level Design

In this version, we will use a simple text-based user interface. As usual, we will have the front-end that interacts with the user, and the back-end that performs all game functions.

Front-end components:

- 7x8 connect4-board:
    - Shows the current layout
    - User input to select a tube
- 2 coin pieces (blue, orange)
    - The board above will show these coins as the game progresses

Back-end components:

We will need some data structure that can internally represent the "7 rows x 8 columns" matrix of the Connect 4 board.

We could use a list of lists to represent a 2 dimensional matrix. For example, the first item in this list would represent the 1st row, the next item would represent the 2nd row, and so on. (Note: We could have made each item represent a tube also, but just to conform to the convention of "rows of columns" we give priority to the rows.)

Data structure:

A list variable called "grid" consisting of 7 sub-lists with each sub-list in turn containing 8 items: this grid represents the 56 cells of Connect 4 board; initially all 0.
Rows will be counted bottom-up (i.e. 1st sub-list is the *bottom* row of Connect 4), and columns will be counted left to right as usual.

Every time a coin is dropped in, we will record its presence in this grid: we would know its "column" simply from which tube was clicked, and its "row" from the "balls in tube" list. We will record the coin by using its color: "orange" or "blue".

Global variables:

List "Board": list of 7 sub-lists each containing 8 items
String "Turn": specifies whose turn it is to play (Orange or Blue)
List "coins in tubes": 8 integers, each indicates # of coins in each tube
We will add more global variables if required as we add more features below.
All globals will be defined in a separate config file.

Initial Set of Features:

Clearly it makes sense to start with the most basic apparatus – which is 2 sets of coins and the grid. Next, we will write scripts for dropping coins in the tubes. This involves the following features:

- Choosing a tube
- Choosing the right coin to drop
- Dropping a coin down the selected tube
- When a tube becomes full, don't allow coins to drop in it.

That will be our starting version of the game. We will discuss what to do next after building this version.

Let us get rolling with these various ideas one by one.

Feature Idea # 1: Choosing tube and coin

Implement a way for the players to choose a tube, and have a way for the players to take turns.

Design:

Selecting a tube is straightforward. Players can simply enter which tube (1 thru 8) they want to choose.

To ensure players play by turn, we can have a variable called "Turn" which will indicate whose turn it is. If it says "orange" an orange coin will be dropped and if it says "Blue" a blue coin will be dropped.

Feature Idea # 2: Drop the coin

Write scripts to drop the selected coin into the selected tube.

Step 1: Take the user-specified tube and drop a coin into it.

Design:

The variable "Turn" tells us which coin is to be dropped. The player will specify the selected tube.

Making the coin drop into the tube is straightforward.

If we knew the row and column in the board where the coin is expected to go, we are all set. We already know the column (tube number). The row number would be 1 more than the number of coins already present in that tube.

We will maintain an 8-item list called "balls in tubes" in which each item will tell us how many coins there are in that tube. For example, item 1 will tell # of coins in the 1st tube, item 2 will tell # of coins in the 2nd tube, and so on.

But, who will create and maintain this list? Every time a tube (i.e. its base) is selected we will increment its corresponding item in this list.

Step 2: When a tube becomes full, don't allow coins to drop in it.

Design:

There are different ways to implement this feature. For example, you could keep a count of the number of coins inside each tube in a list variable, and check that count every time a coin is dropped.

If you remember, we already have a list called "balls in tubes" that tells us how many coins each tube has any time. We can just refer to this list each time before dropping a coin.

We will name this script "drop coin" – which would become, as you will notice below, an important entry point for most of the backend logic.

Save as Program Version 1

Before continuing to the next set of ideas, we will save our project. This way, we have a backup of our project that we can go back to if required for any reason.

Next Set of Features/ideas:

We need to implement one important feature of the game: determining the winner. We will also add a few more features to make the program more tidy, robust, and user-friendly. Here are the things we will consider in this version:

- Detect winner.
- Detect stalemate (i.e. board becoming full).
- Create a script to play the game (basically tie all the features together).
- Make the computer one of the players.
- Add help.

Let us get cracking with these ideas and features one by one.

Feature Idea # 3: Determine the winner

The program should detect and declare winner when 4 coins of either color line up contiguously vertically, horizontally, or diagonally.

Design:

This is the most challenging feature thus far! We need to find a way for the program to see and analyze the entire board of 8 tubes that can each hold 7 coins max. We already have a data structure (the "board" 2-D list) that represents the Connect 4 board as the game proceeds.

Let us now design the algorithms to detect a winner. This is the approach we will use: every time a coin is dropped, it will occupy a cell C in "grid". We will then scan cells adjacent to C horizontally, vertically, and diagonally to check if there are 4 contiguous cells with the same color as C. Since we will run this algorithm every time a coin is dropped, we don't need to scan the entire grid, but just the area adjacent to this coin.

Do you get the idea? The following example might help:

0	Blue	Blue	Orange	Blue	Orange	0	0
0	0	0	Orange	0	Blue	0	0
0	0	0	Orange	0	0	0	0
0	0	0	Orange	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

In this image, we see a snapshot of the Connect 4 program. The grid is shown in table format. The last coin dropped was the orange coin in tube 4. The algorithm will scan cells in its vicinity and discover that there are indeed 4 orange coins lined up contiguously in column 4, and hence it will go ahead and return "True" (i.e. a winner was found).

Here is the algorithm:

```
Algorithm "Is There a Winner"
(Called when a coin is dropped)
Input: row, column, color
For the row:
   -  count left and right from current position, if 4 contiguous
      coins of 'color' found return True
For the column:
   -  count up and down from current position, if 4 contiguous coins
      of 'color' found return True
For diagonals:
   -  count NW and SE from current position, if 4 contiguous coins of
      'color' found return True
   -  count NE and SW from current position, if 4 contiguous coins of
      'color' found return True
No contiguous pattern found, so return False
```

Note that this algorithm will be called every time a coin is dropped (from the "drop coin" scripts of feature #2).

Now, here are details of each step of scanning. Note that since the input cell (whose row, col are passed) is counted twice, our final condition is "count > 4"

```
Algorithm Check Winner in Rows( row, col, color )
count = 0
lrow = row
lcol = col
Repeat until lcol > 8 or item(lrow, lcol) != color
      count += 1
      lcol += 1
lcol = col
Repeat until lcol < 1 or item(lrow, lcol) != color
      count += 1
      lcol -= 1
If (count > 4) return True
```

```
Algorithm Check Winner in Columns( row, col, color )
count = 0
lrow = row
lcol = col
Repeat until lrow > 7 or item(lrow, lcol) != color
      count += 1
      lrow += 1
lrow = row
Repeat until lrow < 1 or item(lrow, lcol) != color
      count += 1
      lrow -= 1
If (count > 4): return True
```

```
Algorithm Check Winner in NE-SW diagonal ( row, col, color )
count = 0
lrow = row
lcol = col
Repeat until lrow > 7 or lcol > 8 or item(lrow, lcol) != color
      count += 1
      lrow += 1
      lcol += 1
lrow = row
lcol = col
Repeat until lrow < 1 or lcol < 1 or item(lrow, lcol) != color
      count += 1
      lrow -= 1
      lcol -= 1
If (count > 4): return True
```

```
Algorithm Check Winner in NW-SE diagonal ( row, col, color )
count = 0
lrow = row
lcol = col
Repeat until lrow < 1 or lcol > 8 or item(lrow, lcol) != color
      count += 1
      lrow -= 1
      lcol += 1
lrow = row
lcol = col
Repeat until lrow > 7 or lcol < 1 or item(lrow, lcol) != color
      count += 1
      lrow += 1
      lcol -= 1
If (count > 4): return True
Return False
```

Feature Idea # 4: Detect stalemate

The program should detect when the entire board becomes full without anyone winning.

Design:

Now that we have the "board" data structure that continuously shows the state of the Connect 4 board, we can easily detect when the grid becomes full of coins. Basically, we will need to scan the entire grid and look for empty cells, i.e. cells containing 0. See the algorithm below:

```
Algorithm "Is Grid Full"
For every row of grid
      For every column of grid
            If cell at (row, column) contains 0
                  Return False
            End if
      End for
End for
```

We will call this algorithm after checking winner.

Feature Idea # 5: Script to play

Now that we have all the important pieces in place, write a script that will allow us to play.

Design:

We will basically go into a loop asking the user which tube he/she wants to select.

Here is a suggested script:

```
Forever:
      Display the board and whose turn it is.
      Ask user to enter tube number (or q to quit and h for help).
      Drop the respective coin in this tube (call "drop coin" script).
      If game over, reset everything for a new game.
      Continue.
End forever
```

Feature Idea # 6: Help

Add a help feature.

Design:

This should be a straightforward task. We will arrange the code such that the help text appears when user enters the 'h' key.

Feature Idea # 7: Play with the computer

Make the program play as one of the players.

Step 1: Use the monkey algorithm for the computer player.

Design:

We will always start the game with the human player making the first move with the "Orange" coin. We will then have the program make the next move with the "Blue" coin. And so on.

"Making the move" for the computer would simply mean dropping the blue coin into one of the tubes. We already have the scripts that do all the work. We just need to "mimic" what happens when a user clicks on one of the tube bases, which is shown below:

```
When tube is selected:
If tube not full:
      Ask my coin to fall (call "drop coin")
```

As mentioned above, we will use the "monkey" approach – which means the program will pick a tube at random. Once the tube has been decided, rest is a matter of duplicating the above actions.

Okay, so far so good. But how will this monkey script be invoked?

As mentioned above, this script should be invoked immediately after the human player has played. So, we could invoke the monkey script where we switch the "turn" variable. In order not to block Tkinter refresh engine, we will invoke the monkey script in a new thread.

Here is the outline of the monkey script:

```
Find a good tube, i.e. tube that is not full
Call the "coin fall" script
```

Step 2: Make the computer a "worthy" opponent.

Design:

You will notice (after playing a bit with the monkey algorithm) that the monkey approach is quite disappointing because the human player can win almost every time with ease. How can we make the game a little more challenging?

Here is one idea. What if we allowed the computer to drop multiple coins at every turn? As it turns out, this can make the program a little more interesting because even if the coins are dropped randomly, with multiple blue coins there is less opportunity for the human player (who only gets to drop one coin) to win easily.

All we need to do then is use a "repeat" loop in the monkey script, and let the user set "how many" turns the computer should get. By default, we will set this variable to 2 and display it as a slider for the user to change.

There are a couple of small glitches that we need to fix. Currently the "Turn" variable determines whose turn it is, and this variable is switched after every coin drop. So the repeat loop in the "monkey script" will cause blue and orange coins to be dropped alternately!

This can be fixed easily by explicitly setting the "turn" variable to "Blue" in the monkey script.

The other glitch relates to the event when the game is over: either when someone wins or when the board becomes full. In these events, the monkey should stop playing because the game is over.

One easy fix is to check the return code of "drop coin" script and check if the game is over.

Here is the final monkey script:

```
N = # of coins the computer gets to play every turn (2 by default)
Repeat N times:
      Find a good tube, i.e. tube that is not full
      Set turn = Blue
      Call the "drop coin" script
            If return code is True (i.e. game is over): stop script
End repeat
```

Save as Program Console Version

Congratulations! You have completed all features of the game for the Console (text) version. Before continuing to the next version (GUI), we will save our project. This way, we have a backup of our project that we can go back to if required for any reason.

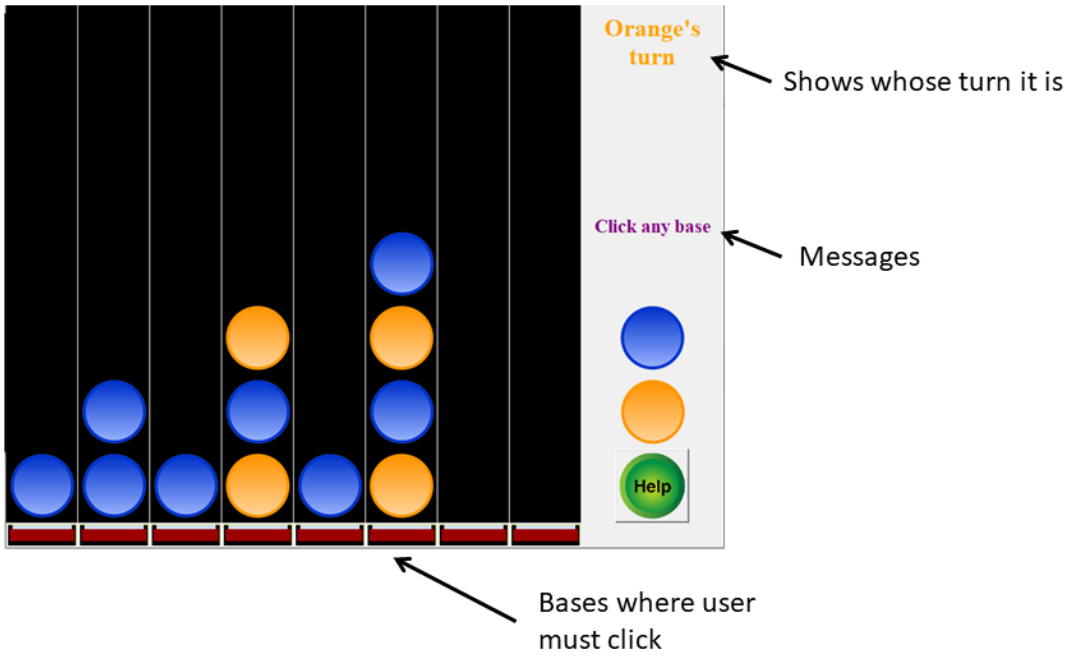
File: connect4-console.py

How to run the program:

1. Run the program file.
2. Enter the base # of the tube in which you want to drop your coin. Enter "h" to view help, or "q" to quit.
3. You (the human player) begin with the "Orange" coin and the computer plays with "Blue" coins. The variable “Turn” shows whose turn it is. The computer gets to drop 2 coins at every turn.

GUI Version

In this version, we will use a graphical user interface as we have shown in the very beginning. Let us see it once again.



Since we already have most of the game logic in place, we only need to figure out how to replace the text interface with an interface where the board is visible as above and user input comes via clicks.

Let's consider how this new type of front-end can replace the current text-based interface.

Front-end components:

- 7x8 board: The frontend will display the board in a 7x8 grid format and pass user input (which base was clicked) to the backend. The frontend will always show the current state of the board. We will use a grid of Tkinter image labels to display the grid. We can borrow the design from the tic-tac-toe program we wrote earlier and modify it.
- An additional row of label widgets for the bases of each tube: this is where user will click to pick a tube.
- An additional column of widgets for the following:
    - A click-button "Help" to display instructions.
    - A message area where messages such as "who is the winner" can be displayed.
    - A text label to show whose turn it is to play.
- In all, we will need an 8x9 Tkinter grid (8 rows and 9 columns) to accommodate all the above components.

Feature Idea # 8: Set up the GUI

Draw the graphical interface with the grid of vertical tubes and other pieces.

Design:

For this, we can simply borrow our older Python program "Tic-tac-toe" which provides the functionality of drawing a grid of Tkinter label widgets with grid manager. We will modify it to suit our design.

Step 1: The connect4 board

The board itself does not need to be sensitive to clicks. We just need the ability to change the images in each label. Initially, when the board is empty, all labels will be blank. When a coin is placed, we just change the "image" property.

Since we need subsequent access to the grid, we will save all label IDs in a 2-D list.

For the bottom row (columns 1 thru 8), each cell (i.e. label widget) will possess its column number so that the backend can associate each base with a tube on the game board. When a base is clicked, the clickLabel callback function will send this column number to the backend.

The grid, as you can see, has a series of vertical lines which define the tubes. These vertical lines between tubes can be achieved by making the image label slightly smaller in width than the width of the base. (Conversely, to prevent similar lines appearing horizontally between rows, make the image label slightly larger than the coin image.)

Step 2: The board state

We already have the list “Board” to represent the board. Each item in this list shows the content of each cell.

The board state changes after a coin is dropped, at which time we will update the image label of the cell where the coin ends up (using its row and column numbers).

Step 3: The additional column of widgets.

As discussed earlier, we will use the last column of the Tkinter grid to place these additional widgets. “Help” is a button widget, which will call the “show help” function when clicked. The other 2 widgets are simple text labels of which we just need to modify the "text" property once in a while.

Feature Idea # 9: Drop the coin

Write scripts to drop the selected coin into the selected tube.

Design:

The variable “Turn” tells us which coin is to be dropped. The player will click on the base of the selected tube. Making the coin drop into the tube is straightforward. We already know which row and column in the grid the coin ends up at. We just need to update the image label of the corresponding cell.

Save as the GUI Program Version

Congratulations! You have completed the program with all the GUI features we had planned. Save your program as “Connect4-final.py”.

Compare your program with my program below.

File: connect4-gui.py

How to run the program:

1. Run the program file. Enter "h" to view help.
2. You (the human player) begin with the "Orange" coin and the computer plays with "Blue" coins. The variable “Turn” shows whose turn it is.
3. The computer gets to drop 2 coins at every turn.
4. Click the base of the tube in which you want to drop your coin.

Additional Challenge:

When a winner is detected, highlight the 4 balls that are lined up contiguously – either by changing their color or by making them blink.

Author: Abhay B. Joshi (abjoshi@yahoo.com)
Last updated: 8 July 2020