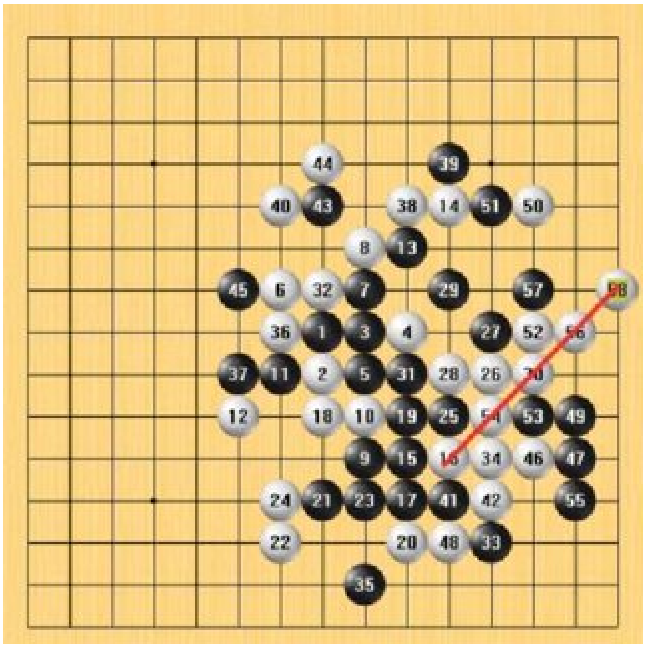
Project - Python Gomoku

Background: A variation from the ancient Chinese board game of GO that became popular when first played in Japan (~270BC) is the game of GoMoku. Specifically, “5 in a row”. It uses the standard 19x19 GO board with white and black pieces alternatingly making moves. Either white or black is declared winner if one of them connects 5 in a row, horizontally, vertically or diagonally. (See example below. Ignore the numbers on the pieces - those are moves in the order they are made)



(A gomoku online with GUI, showcasing the diagonal win condition for white)

In this project, you will make a command-line, simplified version of the GoMoku game in Python.

(The GNU and the Linux Penguin playing a game of GO)

Instruction:

The GO Game Board

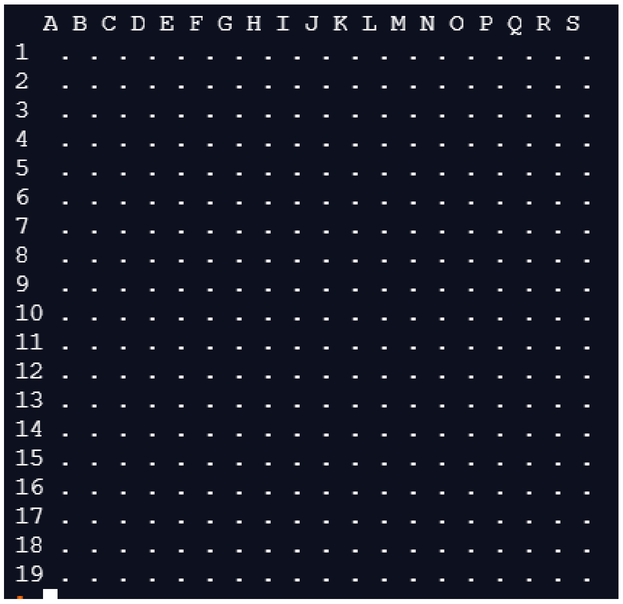
- A 19x19 command-line game board has been given to you in the ***gameboard.py*** file that I’ve provided. Study these functions to understand how the gameboard is created and printed to the console. This uses a **2-dimensional list**. (Lists inside a list)

- Notice that the rows are labelled 1 to 19, and columns labelled A to S. Your user will play the Gomoku game by giving their moves in these coordinates.

1. **Welcome screen** - You should aim to make a decent looking ASCII art welcome screen that demonstrates some aspects of the game the user will be playing. This is totally up to your creativity. You are NOT required to draw the ASCII art yourself, you can simply find one on the internet and put it inside a print() statement. If you are feeling very uncreative, a simple banner with ===== and a welcome message is fine too.

2. **Game Instructions** - After the welcome screen is loaded, your user should have an option to go into game instructions (such as a menu that prompts them to press 1 to start the game, press 2 to read instructions). Here you will list the instructions step by step of how to play.

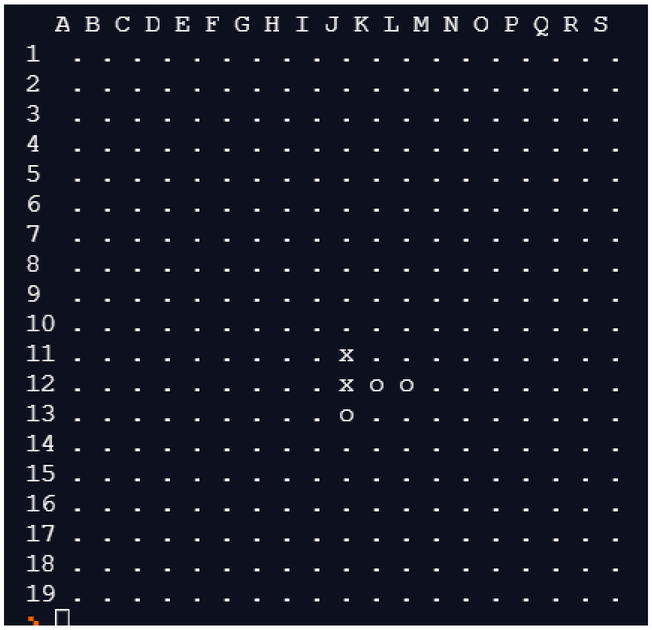
3. **Game Play** -



```
  A B C D E F G H I J K L M N O P Q R S
1  . . . . . . . . . . . . . . . . . . .
2  . . . . . . . . . . . . . . . . . . .
3  . . . . . . . . . . . . . . . . . . .
4  . . . . . . . . . . . . . . . . . . .
5  . . . . . . . . . . . . . . . . . . .
6  . . . . . . . . . . . . . . . . . . .
7  . . . . . . . . . . . . . . . . . . .
8  . . . . . . . . . . . . . . . . . . .
9  . . . . . . . . . . . . . . . . . . .
10 . . . . . . . . . . . . . . . . . . .
11 . . . . . . . . . . . . . . . . . . .
12 . . . . . . . . . . . . . . . . . . .
13 . . . . . . . . . . . . . . . . . . .
14 . . . . . . . . . . . . . . . . . . .
15 . . . . . . . . . . . . . . . . . . .
16 . . . . . . . . . . . . . . . . . . .
17 . . . . . . . . . . . . . . . . . . .
18 . . . . . . . . . . . . . . . . . . .
19 . . . . . . . . . . . . . . . . . . .
```

- Your game is NOT required to provide an AI (that would be way too hard for this project).
- Your game will support 2 player mode. That is, rotating between player 1 “o” and player 2 “x”. You must prompt the user with “It is now player [n]’s turn to move”.
- Player 1 “o” will always start the first move.
- Your game will terminate and report “Player [n] has won” if the ***winning condition*** (see below) has been reached by either player “o” or player “x”.
- Ask them if they wish to play again (Y/N). A no will thank them for playing and display the exit screen, exit program.

4. **Make Move Command**


```
  A B C D E F G H I J K L M N O P Q R S
1  . . . . . . . . . . . . . . . . . . .
2  . . . . . . . . . . . . . . . . . . .
3  . . . . . . . . . . . . . . . . . . .
4  . . . . . . . . . . . . . . . . . . .
5  . . . . . . . . . . . . . . . . . . .
6  . . . . . . . . . . . . . . . . . . .
7  . . . . . . . . . . . . . . . . . . .
8  . . . . . . . . . . . . . . . . . . .
9  . . . . . . . . . . . . . . . . . . .
10 . . . . . . . . . . . . . . . . . . .
11 . . . . . . . . . x . . . . . . . . .
12 . . . . . . . . . x o o . . . . . . .
13 . . . . . . . . . o . . . . . . . . .
14 . . . . . . . . . . . . . . . . . . .
15 . . . . . . . . . . . . . . . . . . .
16 . . . . . . . . . . . . . . . . . . .
17 . . . . . . . . . . . . . . . . . . .
18 . . . . . . . . . . . . . . . . . . .
19 . . . . . . . . . . . . . . . . . . .
```

- The user will make a move by giving inputs like **12,G**. You can tell the user what kind of input format you take from them. The most common is **row,column** separated by a comma. But likewise you can also separate by space or dash etc. It's whatever you are able to parse into the correct coordinates on the board.
- Valid: If the user's move is valid (i.e. no pieces on top of the coordinate already, it's inside the board), replace the display at that coordinate with the user's piece "o" or "x". Print the new board.
- Invalid: If it's invalid, prompt the user that it's invalid and get the user to make a different move.
- Hint: The column is given by an alphabet. The way to turn this into a value between 1 & 19 is to make a list of alphabets "A" to "S", and then try to find the position of the user's input alphabet in the list. That position is its column position on the board (the 2D-list).
- Prompt the other player to make a move and repeat this alternating move sequence until a user has won.

5. **Winning Condition**
   - This is perhaps the hardest part of the game to implement.
   - As you are doing a simplified Gomoku, you only have to check if the user has a **Horizontal** win condition or **Vertical** win condition, you do not need to check the diagonal win condition (it will be a bonus if you do).
   - Implementation will be as follows:
      i. Everytime a user makes a move, you will check if they currently have a win condition.
      ii. You can do this by finding the coordinate with the piece the user just played. You will check if that piece ***hasHorizontalWin(...)*** or ***hasVerticalWin(...)***.

        iii. If either condition is true for that piece, it means they've reached a winning condition and they have won.
        iv. As mentioned in Game Play, your game will terminate and report who has won.
    - You are ALLOWED a different kind of implementation if you have your own ideas, as long as they produce the same correct outcome. You don't have to follow the steps I gave above to the T.

6. **Some functions you will need to implement** (you decide what parameters you want to use for them):
    - **move_x(...)**: changes the symbol at the user's move coordinate to an "x" in the 2D-list if it's a valid move.
    - **move_o(...)**: changes the symbol at the user's move coordinate to an "x" in the 2D-list if it's a valid move.
    - **hasHorizontalWin(...):** checks if given any piece (its coordinate), if the piece has a 5-connected horizontal of the same symbol. If it does, return True. Otherwise return False. Note that this one will have to take into consideration details of the coordinate of the piece you are checking for. If it's far left or far right, you might only have a few pieces to its left or right before reaching the end of the board. That is, there are quite a bit of "edge" cases you have to think about.
    - **hasVerticalWin(...):** same as hasHorizontalWin, but does the vertical checking.
    - **BONUS: hasDiagonalWin(..):** same as horizontal and vertical, but checks the cross "X" through a piece to see if it has a diagonal win.
    - **ANY OTHER CUSTOM FUNCTIONS** you want to have (I encourage you to make as many as necessary to make well-organized code and reuse functions) to make your game flow. For example, you might want a print_instructions() function that just prints the instructions. You might want a display_welcom() function that displays the welcome message, etc. Get creative here :)

I'm looking forward to playing you (& defeating you) in your game >:D