Tic Tac Toe Pyramid

Team Members: Ariadne Rincon, Daniel Cao, Andy Cao

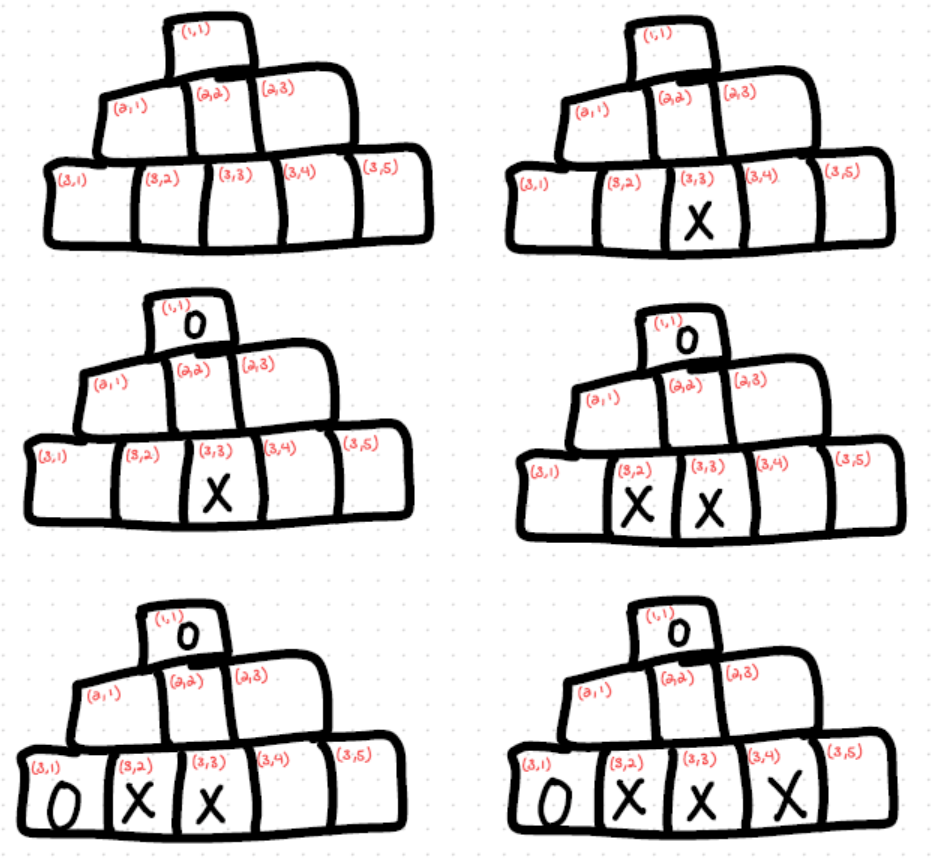
For this final project, our team worked on creating a variation of the very common tic tac toe game. In order to produce something interesting, we modified the look of the game and changed the standard 3x3 grid into a pyramid shape. The general approach with this program was to use the game of nim algorithm and then change the elements and gameplay to tic tac toe. The game of nim already produced that desired pyramid shape, but then coded the tic tac toe rules (what moves are valid/invalid, when the game ends, which player wins, etc.). Our end goal was also to create a more polished game with a visual representation of the game as it progresses. With tkinter we aim to create a GUI component for the game that will update after every move. Since we have all become familiar with python this semester we decided to program the game with this language. A bonus was that tkinter is a GUI built within the python library, simplifying the process of finding compatible options.

Here is the breakdown of the files the final project folder contains:

- README.me contains the project description, image representations, and how the game works for those who may not be familiar with the standard tic tac toe game
- tic_tac_toe_pyramid.py contains the main code where the initial state of the game is created, functions to update the game state, and display functions to output the moves and current states. Also contains the tkinter code that displays the game on a pop up screen.
- games.py (UNTOUCHED) contains the given textbook code and is necessary for the main program to run
- utils.py (UNTOUCHED) contains the given textbook code and is necessary for the main program to run

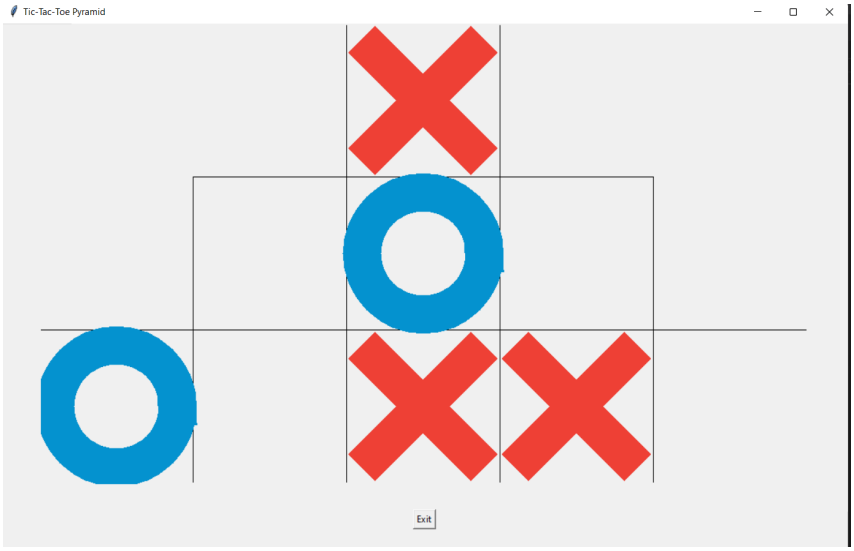
The way we initially evaluated the code was to run the tic_tac_toe_pyramid.py file through the terminal and printing out all the “moving parts” within each step to ensure everything is changing correctly. The “moving parts” included checking the next player’s turn (either X or O), all of the possible moves after a turn is taken, displaying which move was selected from the list, the current board, the game state object with all of its attributes, and finally when the game ends we display the winner and make sure that is accurate. Before creating the GUI portion we used the output to create a drawing after every step and verify that the program was working perfectly. The time that this game takes depends heavily on the user because the program will return a move for ‘X’ within milliseconds. It is very difficult for the user to win as the program has a heavy preference on making its first move at (3,3).

An attachment of a drawing following a run-through below:



Computer plays as ‘X’. User plays as ‘O’. ‘X’ wins.

Once the GUI portion has been added the evaluation process for the code was much simpler because instead of redrawing and imagining where the placement of each player’s marker was tkinter displayed it all visually. Example image shown below:



Moving forward with what we have created, an ideal enhancement to add is the ability to have the program also choose a random move for you if you choose. Let's say as the player you are having a difficult time selecting the best move for yourself, the program is designed to select the best move from the list already and that can also be used for the player's benefit as well. Having a "Help me Choose!" or "Random" button that triggers an auto-selection would be an ideal way to implement the addition. Overall, this program really taught us the importance to prioritize the user's experience with playing games. If we were to not create a GUI component the game would have been far too difficult for anyone to follow making the game less enjoyable. In general, this reinforces the importance of user interpretability in order to make successful programs and games.

References

- games.py
- utils.py
- tkinter