Python Tic Tac Toe.

Create an implementation of Tic Tac Toe in Python which uses the command line as its canvas to display the current gameboard. I will have to randomly choose who would goes first and design an algorithm for the computer.

Randomly selecting who goes first.

This was very simple, when a new game starts, I prompt the user asking for heads or tails. I then randomly generate another heads or tails programmatically. If the both the players input and generated result are the same the player goes first, otherwise the computer goes first. This also determines who is what piece. Whoever goes first has the piece *X* meaning the opponent has *O*.

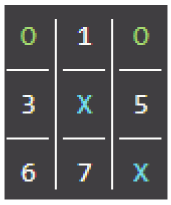
Managing and drawing the gameboard.

This, again, was quite simple. My gameboard state is an array (or a *list* in Python) which contains 9 strings. The strings are always one of the following: *X*, *O*, or any empty string. When the user moves, they enter a number which corresponds to an index in the gameboard array. If the space is empty and is a valid move, the players character will be written to that index. The computer does the same when it moves.

To even consider drawing the gameboard we must do two things. One, we must clear the terminal because the previous drawn gameboard will still visible. To do this, we run a system command that varies depending on the system (Windows uses *cls* whereas Unix like systems use *clear*). Secondly, we must process the gameboard so we can print it clearly. To do this, I iterate through the gameboard and every iteration I check if:

- The space is occupied by the player. If it is, the text is wrapped in special characters to make the text appear cyan.
- The space is occupied by the computer. If it is, the text is wrapped in special characters to make the text appear green.
- The space is unoccupied. If it is, the position of the board is displayed so the user knows what number corresponds to what place on the board.

Example of generated gameboard:



```
 O | 1 | O
---|---|---
 3 | X | 5
---|---|---
 6 | 7 | X
```

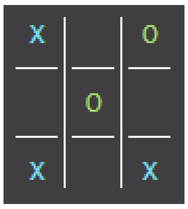
The resulting processed data is separated by characters to make a tidy board.

Checking for game ending states.

Tic Tac Toe is a very simple game, it only has 8 possible ways to win. So, to check if someone has one, I simply must check these 8 combinations. I check if any horizontal row has 3 of the same pieces, if it does, whoever piece that is has won. I then do the same thing vertically. Lastly, I check if 3 of the same pieces are in the diagonals. If so, whoever's piece has one. If all the spaces on the board are filled and nobody has won, the game ends with a draw. At the end of the game, I ask the user if they would like to play another game, otherwise I end the main event loop and quit.

Computers algorithm.

When it is the computers turn, the computer tries to find the most optimal move to make. The algorithm follows a hierarchy of rules with lowering importance. When making a move the computer checks if it can win in the next move. If it can, the computer makes this move – this is the most important rule because the computer wants to win. If the computer cannot win in the next move, the computer checks if the player could win next. If the player has a move that would let them win, the computer counteracts this and makes its move to block the player from winning. This move will often lead the computer drawing with its opponent. A draw is better than outright loosing, however. If nobody can win in the next turn, the computer randomly picks an available corner or the centre as these board positions are more powerful than regular spaces. If, however, there are no available corners and the centre is taken, the computer will randomly pick any remaining space without any strategy.



Example of the computer (green) being trapped by the player (cyan).

If you look at the example to the right, you will see that it is possible for the computer to fall into this trap, which means the player can beat it. The algorithm the computer uses has a fatal flaw – it doesn't weigh moves; it randomly picks spaces that are available if it cannot win or if it cannot block the player from winning. There are two potential solutions to this problem. Number one, get the computer to recognise the moves leading up to a trap like this and counteract them. This would be the easiest solution all-round. Number two, weight moves using machine learning. All moves that resulted in the opponent winning would punish the algorithm and all moves that leads to the computer winning would result in the algorithm being praised. Over time, this would create an algorithm that would look at the gameboard and find the best move which has the highest likelihood of the computer winning.

Source code.

The source code is available at: https://github.com/tywil04/tictactoe