Tic Tac Toe

This is the solution for the Milestone Project! A two player game made within a Jupyter Notebook. Feel free to download the notebook to understand how it works!

First some imports we'll need to use for displaying output and set the global variables

In [1]:
```
# Specifically for the iPython Notebook environment for clearing output.
from IPython.display import clear_output

# Global variables
board = [' '] * 10
game_state = True
announce = ''
```

Next make a function that will reset the board, in this case we'll store values as a list.

In [2]:
```
# Note: Game will ignore the 0 index
def reset_board():
    global board,game_state
    board = [' '] * 10
    game_state = True
```

Now create a function to display the board, I'll use the num pad as the board reference. Note: Should probably just make board and player classes later....

In [3]:
```
def display_board():
    ''' This function prints out the board so the numpad can be used as a refe
rence '''
    # Clear current cell output
    clear_output()
    # Print board
    print "  "+board[7]+" |"+board[8]+" | "+board[9]+" "
    print "------------"
    print "  "+board[4]+" |"+board[5]+" | "+board[6]+" "
    print "------------"
    print "  "+board[1]+" |"+board[2]+" | "+board[3]+" "
```

Define a function to check for a win by comparing inputs in the board list. Note: Maybe should just have a list of winning combos and cycle through them?

In [4]:
```
def win_check(board, player):
    ''' Check Horizontals,Verticals, and Diagonals for a win '''
    if (board[7]  ==  board[8] ==  board[9] == player) or \
        (board[4] ==  board[5] ==  board[6] == player) or \
        (board[1] ==  board[2] ==  board[3] == player) or \
        (board[7] ==  board[4] ==  board[1] == player) or \
        (board[8] ==  board[5] ==  board[2] == player) or \
        (board[9] ==  board[6] ==  board[3] == player) or \
        (board[1] ==  board[5] ==  board[9] == player) or \
        (board[3] ==  board[5] ==  board[7] == player):
        return True
    else:
        return False
```

Define function to check if the board is already full in case of a tie. (This is straightforward with our board stored as a list) Just remember index 0 is always empty.

In [5]:
```
def full_board_check(board):
    ''' Function to check if any remaining blanks are in the board '''
    if " " in board[1:]:
        return False
    else:
        return True
```

Now define a function to get player input and do various checks on it.

In [6]:
```
def ask_player(mark):
    ''' Asks player where to place X or O mark, checks validity '''
    global board
    req = 'Choose where to place your: ' + mark
    while True:
        try:
            choice = int(raw_input(req))
        except ValueError:
            print("Sorry, please input a number between 1-9.")
            continue

        if choice not in range(1,10):
            print("Sorry, please input a number between 1-9.")
            continue

        if board[choice] == " ":
            board[choice] = mark
            break
        else:
            print "That space isn't empty!"
            continue
```

Now have a function that takes in the player's choice (via the ask_player function) then returns the game_state.

In [7]:
```
def player_choice(mark):
    global board,game_state,announce
    #Set game blank game announcement
    announce = ''
    #Get Player Input
    mark = str(mark)
    # Validate input
    ask_player(mark)

    #Check for player win
    if win_check(board,mark):
        clear_output()
        display_board()
        announce = mark +" wins! Congratulations"
        game_state = False

    #Show board
    clear_output()
    display_board()

    #Check for a tie
    if full_board_check(board):
        announce = "Tie!"
        game_state = False

    return game_state,announce
```

Finally put it all together in a function to play the game.

In [8]:
```
def play_game():
    reset_board()
    global announce

    # Set marks
    X='X'
    O='O'
    while True:
        # Show board
        clear_output()
        display_board()

        # Player X turn
        game_state,announce = player_choice(X)
        print announce
        if game_state == False:
            break

        # Player O turn
        game_state,announce = player_choice(O)
        print announce
        if game_state == False:
            break

    # Ask player for a rematch
    rematch = raw_input('Would you like to play again? y/n')
    if rematch == 'y':
        play_game()
    else:
        print "Thanks for playing!"
```

Let's play!

In [9]:
```
play_game()
```

```
  O |X |
------------
  X |X | O
------------
  O |X |
X wins! Congratulations
Would you like to play again? y/nn
Thanks for playing!
```