# CSC 202 Programming #1 Assignment

## *Implementing the Minimax Recursive Algorithm to play Tic-Tac-Toe*

**Due 5/3 (Thursday) at lab time.** ***Polylearn code submission due by start of lab.***

You will need to demo your program in lab (no exceptions), *and submit your git repository* on PolyLearn (in a zip archive). Be sure to follow the guidelines for repositories presented in the course syllabus.

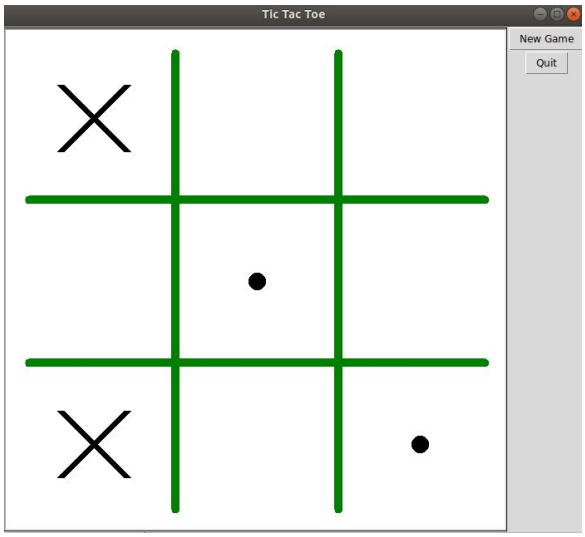
Read sections 4.8 and 4.9 in your textbook for background on minimax. I will also provide descriptions in lecture / lab.

Your starter code is: http://knuth.luther.edu/~leekent/CS2Plus/chap4/chap4.html#tic-tac-toe

***You will need to replace the X and O class definitions in the starter code with the definitions below. The specific changes from starter code are highlighted in orange.***

```
class X(RawTurtle):
    def __init__(self, canvas = None):
        if canvas != None:
            super().__init__(canvas)
            self.ht()
            self.getscreen().register_shape("X",((-40,-36),(-40,-44),(0,-4),(40,-44),(40,-36),(4,0), \
                                (40,36),(40,44),(0,4),(-40,44),(-40,36),(-4,0),(-40,-36)))
            self.shape("X")
            self.penup()
            self.speed(5)
            self.goto(-100,-100)

    def eval(self):
        return Computer

    def __repr__(self):
        return "X()"

    def __str__(self):
        return "X()"

class O(RawTurtle):
    def __init__(self, canvas = None):
        if canvas != None:
            super().__init__(canvas)
            self.ht()
            self.shape("circle")
            self.penup()
            self.speed(5)
            self.goto(-100,-100)

    def eval(self):
        return Human

    def __repr__(self):
        return "O()"

    def __str__(self):
        return "O()"
```

The final running program should look something like this:




***IMPORTANT - You must name your tic-tac-toe program exactly: tictactoe.py. Other names will cause 'import' problems for our grading script.***

Grading Rubric:
5 percent - git repository
5 percent - providing your own unit tests (must contain at least two unit tests for minimax, board equality, eval, full board functions)
5 percent - proper naming of source files (i.e., tictactoe.py)
15 percent - working program (must demo working program on due date to instructor); this will involve direct code inspection of the computerTurn() function
*45 percent - pass grader unit tests for minimax (recursive algorithm)*
10 percent - pass grader unit tests for board equality operator
10 percent - pass grader unit tests for board eval function
5 percent - pass grader unit tests for full board test

Sample unit tests:

```
from tictactoe import *
import unittest

class TestTicTacToe(unittest.TestCase):
  def test_board_equality_operator(self):
    b0 = Board()
    b0[0][0] = X(); b0[0][1] = X(); b0[0][2] = X()
    b0[1][0] = O(); b0[1][1] = X(); b0[1][2] = O()
    b0[2][0] = X(); b0[2][1] = O()

    b1 = Board()
    b1[0][0] = X(); b1[0][1] = X(); b1[0][2] = X()
    b1[1][0] = O(); b1[1][1] = X(); b1[1][2] = O()
    b1[2][0] = X(); b1[2][1] = O()
```

```
    self.assertEqual(b0, b1, 'Boards are equality.')
    self.assertEqual(b1, b0, 'Boards are equality.')

  def test_board_eval(self):
    b = Board()
    b[0][0] = X(); b[0][1] = X(); b[0][2] = X()
    b[1][0] = O(); b[1][1] = O()
    b[2][0] = O()

    # Note: Computer plays X's
    self.assertEqual(b.eval(), 1, 'X wins.')

  def test_board_full(self):
    b = Board()
    b[0][0] = O(); b[0][1] = X(); b[0][2] = O()
    b[1][0] = X(); b[1][1] = X(); b[1][2] = O()
    b[2][0] = O(); b[2][1] = O(); b[2][2] = X()

    self.assertTrue(b.full(), 'Full board.')

  def test_minimax(self):
    b = Board()
    b[0][0] = X(); b[0][1] = X()
    b[1][0] = O(); b[1][1] = O()
    b[2][0] = O()

    self.assertEqual(minimax(Computer, b), 1, 'Board contains a win for X')

if __name__ == '__main__':
  unittest.main()
```