```python
from tictactoe_board import *

def main():
    the_board = Tictactoe_board(['XOX', 'OXO', 'XOO'])
    print(the_board)
    print("The winner is %s" % the_board.get_winner())
    print()

    the_board.place_piece(2, 0, 'O')
    print(the_board)
    print("The winner is %s" % the_board.get_winner())

if __name__ == "__main__":
    main()

```

```
"""
Testing utilities.  Do not modify this file!
"""

num_pass = 0
num_fail = 0

def assert_equals(msg, expected, actual):
    """
    Check whether code being tested produces
    the correct result for a specific test
    case. Prints a message indicating whether
    it does.
    :param: msg is a message to print at the beginning.
    :param: expected is the correct result
    :param: actual is the result of the
    code under test.
    """
    print(msg)

    global num_pass, num_fail

    if expected == actual:
        print("PASS")
        num_pass += 1
    else:
        print("**** FAIL")
        print("expected: " + str(expected))
        print("actual: " + str(actual))
        num_fail += 1

    print("")

def start_tests(header):
    """
    Initializes summary statistics so we are ready to run tests using
    assert_equals.
    :param header: A header to print at the beginning
    of the tests.
    """
    global num_pass, num_fail
    print(header)
    for i in range(0,len(header)):
        print("=",end="")
    print("")
    num_pass = 0
    num_fail = 0

def finish_tests():
    """
    Prints summary statistics after the tests are complete.
    """
    print("Passed %d/%d" % (num_pass, num_pass+num_fail))
    print("Failed %d/%d" % (num_fail, num_pass+num_fail))

```

```
Lab Question 1: What methods are private?

__row_as_string(self, row)

__three_in_row(self, player, start_x, start_y, dx, dy)

__is_winner(self, player)


Lab Question 2: What instance variables does it have?

self.__board


Lab Question 3: Write a short description of the internal
representation of a board.

Tictactoe_board init:

1. creates an empty list to represent the board
2. checks if the parameter rows needs to be added to the
board or if there are None to add
3. If None, then it fills the board with lists of 3 single
space chars. Number of lists depends on number of rows the
board will have (which is a static variable)
4. Otherwise, an empty row list is created
5. then it loops through each character of each string in
the rows parameter, adding each character individually to
the row list
6. each row list is then appended to the board list
```

```python
"""
defines the behavior of a tic-tac-toe board
"""

NUM_ROWS = 3

class Tictactoe_board:

    def __init__(self, rows):
        """
        Constructor. Creates a tictactoe board with given cell values.
        If no initial cell values are given, creates an empty tictactoe board.

        :param rows: A list of three 3-character strings, where each character
        is either 'X', 'O', or ' '.  Each of the
        3-character strings represents a row of the tictactoe board.
        Example: [" X ", "O O", "XXO"] is the board
           | X |
        -----------
         O |   | O
        -----------
         X | X | O
        """
        self.__board = []
        if rows is None:
            empty_row = [' ', ' ', ' ']
            for i in range(NUM_ROWS):
                self.__board.append(empty_row)
        else:
            for i in range(NUM_ROWS):
                row = []
                for j in range(NUM_ROWS):
                    row.append(rows[i][j])
                self.__board.append(row)

    def place_piece(self, i, j, piece):
        """
        Places a piece (either 'X' or 'O') on the board.

        :param i: The row in which to place a piece (0, 1, or 2)
        :param j: The column in which to place a piece (0,
1, or 2)
        :param piece: The piece to place ('X' or 'O')
        """
        self.__board[i][j] = piece

    def clear_cell(self, i, j):
        """
        Clears a cell on the tictactoe board.

        :param i: The row of the cell to clear
        :param j: The column of the cell to clear
        """
        self.place_piece(i, j, ' ')

    def __row_as_string(self,row):
        """
        returns row in a format suitable for printing
        :param row: row of board as list of strings
        :return: row in prettified string format
        """
        str = ''
        for column in row[:len(row)-1]:
            str += column + ' | '
        str += row[len(row)-1]
        return str

    def __str__(self):
        """
        Produces a string representation of a board, returns it.

        :return: The string version of the board.
        """
        result = ''
        for row in self.__board[:len(self.__board)-1]:
            result += self.__row_as_string(row)
            result += '\n----------\n'
        result += self.__row_as_string(self.__board[len(self.__board)-1])
        result += '\n'
        return result

    def __three_in_row(self, player, start_x, start_y, dx, dy):
        """
        Determines if a player has three in a row, starting
        from a starting position (start_x, start_y) and going
        in the direction indicated by (dx, dy)
        """
        x = start_x; y = start_y
        for i in range(0,NUM_ROWS):
            if self.__board[y][x] != player:
                return False
            x += dx; y += dy

        return True


    def __is_winner(self, player):
        """Returns True if and only if the given player has won"""

        if self.__three_in_row(player, 0, 0, 1, 1):
            return True
        elif self.__three_in_row(player, 2, 0, -1, 1):
            return True
        else:
            for i in range(0, NUM_ROWS):
                if (self.__three_in_row(player, 0, i, 1, 0)
                    or self.__three_in_row(player, i, 0, 0, 1)):
                    return True
            return False


    def get_winner(self):
        """
        Determines if there is a winner and returns the player who has won.
        :param board: A tictactoe board.
        :return: 'X' if player X is the winner; 'O' if player O is the winner; None if there is no winner.
        """
        if self.__is_winner('X'):
            return 'X'
        elif self.__is_winner('O'):
            return 'O'
        else:
            return None

```

```python
"""
:author: Ian Sulley

Honor Code Statement:
I affirm that I have carried out the attached academic endeavors with full academic honesty,
in accordance with the Union College Honor Code and the course syllabus.

"""

from tictactoe_board import *
from testing import *


def test_get_winner():
    start_tests("Tests for tictactoe_board.get_winner()")
    test_get_winner_horiz_X()
    test_get_winner_horiz_O()
    test_get_winner_vert_X()
    test_get_winner_vert_O()
    test_get_winner_diag_l_to_r_X()
    test_get_winner_dia_l_to_r_O()
    test_get_winner_diag_r_to_l_X()
    test_get_winner_dia_r_to_l_O()
    test_get_winner_incomplete_board()
    test_get_winner_empty()
    finish_tests()

"""
Individual unit tests start here
"""

def test_get_winner_horiz_X():
    a_board = Tictactoe_board(['XXX', 'OOX', 'XOO'])
    assert_equals(str(a_board) + "Three Xs in a row horizontally",
                  'X',
                  a_board.get_winner())

def test_get_winner_horiz_O():
    a_board = Tictactoe_board(['XOX', 'OOO', 'XXO'])
    assert_equals(str(a_board) + "Three Os in a row horizontally",
                  'O',
                  a_board.get_winner())
```

```python


def test_get_winner_vert_X():
    a_board = Tictactoe_board(['XOX', 'XXO', 'XOO'])
    assert_equals(str(a_board) + "Three Xs in a row vertically",
                  'X',
                  a_board.get_winner())


def test_get_winner_vert_O():
    a_board = Tictactoe_board(['XOX', 'OOX', 'XOO'])
    assert_equals(str(a_board) + "Three Os in a row vertically",
                  'O',
                  a_board.get_winner())

def test_get_winner_diag_l_to_r_X():
    a_board = Tictactoe_board(['XOX', 'OXO', 'OOX'])
    assert_equals(str(a_board) + "Three Xs in a row diagonally from top left to bottom right",
                  'X',
                  a_board.get_winner())


def test_get_winner_dia_l_to_r_O():
    a_board = Tictactoe_board(['OXX', 'OOX', 'XOO'])
    assert_equals(str(a_board) + "Three Os in a row diagonally from top left to bottom right",
                  'O',
                  a_board.get_winner())

def test_get_winner_diag_r_to_l_X():
    a_board = Tictactoe_board(['XOX', 'OXO', 'XOO'])
    assert_equals(str(a_board) + "Three Xs in a row diagonally from top right to bottom left",
                  'X',
                  a_board.get_winner())


def test_get_winner_dia_r_to_l_O():
    a_board = Tictactoe_board(['OXO', 'XOX', 'OXX'])
    assert_equals(str(a_board) + "Three Os in a row diagonally from top right to bottom left",
                  'O',
                  a_board.get_winner())
```

```python

def test_get_winner_incomplete_board():
    a_board = Tictactoe_board(['XXX', 'OOX', 'XOO'])
    a_board.clear_cell(0, 0)
    assert_equals(str(a_board) + "Incomplete board, no winner yet",
                  None,
                  a_board.get_winner())


def test_get_winner_empty():
    a_board = Tictactoe_board(None)
    assert_equals(str(a_board) + "Empty board, no winner yet",
                  None,
                  a_board.get_winner())


if __name__ == "__main__":
    test_get_winner()

```