Assignment 2

CSE 473 – Introduction to Artificial Intelligence, University of Washington, Spring 2010.

Due Friday, April 16, at 5:00 PM.

This assignment starts with some exercises about the combinatorics of search. It continues with a problem-formulation exercise, and then a series of Python programming steps focused on the game of Tic-Tac-Toe. Although Tic-Tac-Toe is a simple game, it serves as a convenient stepping stone to more complicated games, such as the games to be considered in Assignment 3.

Turn in two files: (i) a text file in PDF form, named **A2text.pdf**, for exercise answers and program test runs, and, (ii) source code for your Tic-Tac-Toe agent, final version (after completing exercises 8 and 9). See exercise 8a for specifying the name of this file.

1. Counting a Few States (10 points). In the game of Tic-Tac-Toe there is one initial state, said to be at Ply 0, and 9 children of the initial state. These 9 children are said to be at Ply 1. How many states are at Ply 2? How many distinct states are at Ply 3? Ply 4? At Ply k? (Justify your answer. You may assume for this exercise that having 3 Xs in a row or 3 Os in a row does not end a game and thus such a state might still have successors.)

2. Counting More (5 points). In a Painted Squares puzzle with 5 pieces, a plus-shaped board (one center square and squares neighboring it on each of its four sides), how many distinct states are there? Assume all 5 pieces are different, and each piece has all of its sides differently painted, i.e., no piece has two sides painted the same way. Also assume that two arrangements of squares that can be obtained from each other by a rotation or flip of the entire board are considered distinct. (Assume that a state that is not a goal state is allowed to have squares placed in ways that violate the matching-sides constraints.)

3. Representing "Equilibrium" (20 points). At the end of the reading on state-space search (see link from the course home page), a class of puzzles called "Equilibrium puzzles" is described in Exercise 7. Give (a) a possible Python representation for the first piece (number 1), shown in Figure 5.11, and (b) a possible state representation for the arrangement of the 6 given squares in a 3 by 2 array suggested by the diagram. (c) Describe, in English rather than Python, what an adequate set of operators would be for solving puzzles in this class of puzzles. (d) Give a detailed description of each operator in your set. What is its precondition? How does it transform the state?

4. Detecting Wins in Tic-Tac-Toe (5 points). Download the starting code for this assignment from the Assignments page of our course website. Try running it to make sure you have it all and have it set up properly. Write a Python function called **topRowWinForX(state)** that returns **True** if and only if there are three Xs in the top row of the board. Write another Python function called **anyWinForX(state)** that returns **True** if and only if X has three in a row anywhere in the board.

5. Reporting Wins in Tic-Tac-Toe (5 points). Write a variation of your **anyWinForX** function, with the same name, in which there is an optional second argument **describe** that defaults to **False**, but if set to **True** by the call, causes the global variable **utterance** to be changed (if there is a win for X) so that it describes where the win occurs. For example,

utterance might be assigned the string "X wins in the top row" or "There is a win for X on the main diagonal" or "Column 3 now belongs entirely to X".

6. Creating All Children (5 points). Write a function **successors(state)** that returns a list of all the possible legal children of **state**. If there are no successors, then the empty list should be returned. Note that each child must be an actual state.

7. Move Selection (5 points). Write a function **chooseMove(listOfStates)** that returns one of the states in the given list. The list should be of the form returned by the **successors** function. It must return a win state if there is one. That is, it can just return the first state for which **anyWinForX** (or a corresponding **anyWinForO** function) returns True, and if there are no win states, then it can return any of the given states.

8. Your Tic-Tac-Toe Player (15 points). Starting with either the file JohnsTicTacToe.py or MarysTicTacToe.py, create your own file, and use a filename that consists of your UWNetID followed by TicTacToe.py, with no spaces or dashes.

(a) Change the functions **myNameIs** and **myCreatorIs** to reflect your own information.
(b) Modify the **makeMove** function in the file so that, instead of calling **findFirstMove**, it calls your **successors** function and then calls **chooseMove** on the successors.
(c) Change the code that creates the **utterance** in each place it is assigned, so that it corresponds to something of your own creation.

9. Using a Static Evaluation Function (10 points). Modify your player to return the "best" move, where by "best" we mean the move whose resulting state optimizes the value of the static evaluation function described on page 200 of the reading. As part of this, implement a function staticEval(state) that returns the value of the static evaluation function on the given state. The player should also emit (by printing to the console—the usual standard output stream) an utterance that depends directly on the static value of the new state.

10. Matches (20 points). Modify the file **TicTacToeMatch.py** so that it loads your program and your partner's program as player1 and player2 and runs a game. Run the game. Capture the output and include it in your PDF file **A2text.pdf**.