George Rossney
Project 1: Tic Tac Toe
CSC 242
2/15/16

Tic Tac Toe Project

The goal of this project is to design and build a Tic Tac Toe game in which the program can beat a human user. I have implemented the game in Java by setting up a state-space search problem. I have designed games for both the classic 3x3 version and the expanded Nine-Board version of Tic Tac Toe. In order for the program to win against a human user, I have implemented the minimax algorithm with heuristic search and alpha-beta pruning. I started with pseudo code from our textbook, then coded it in Java. The Tic Tac Toe game is played depending on user input in the console. The output is a Tic Tac Toe board constructed with standard keyboard characters in the output window. The user is prompted to enter a move, then the output is the updated board with their move and the computer's next move. The game runs until a win on the board is detected. After each move, methods check for a win on any of the rows, columns, left diagonals, or right diagonals. I have designed the game so that the computer always plays as "X" and the user always plays at "O". At the beginning of the game, the user is prompted on whether they would like to make the first move, or not.

In order for the computer to beat the human user, it has to make smart moves. Generating random moves may work for certain cases, but there has to be an effective algorithm for the computer to calculate and make the best move possible. The goal of Tic Tac Toe is to have three X's or three O's in a row, but the main strategy has to involve making good moves while blocking a win for your opponent. I developed a program that

will consistently enter a third symbol to block the human user's win. If the user happens to win, it was because they had two possible winning moves and the computer could only block one. For both versions of Tic Tac Toe, it is possible for there to be a tied game. For the 3x3 version, the computer will always want to complete its final move even if a draw is evident by the eighth move.

In order to design and code a Tic Tac Toe game in Java, I have followed the five-state model we learned in class. Since there are two players who each want to win, winning Tic Tac Toe is an adversarial search problem. It is also a zero-sum game, since each move that benefits one player hurts the other. The initial state of Tic Tac Toe is an empty board, with nine move options for the first turn, and one less for each consecutive turn. It terms of a game tree, the next move is represented as an option from multiple child nodes at the same depth. Each player move increases the depth by one level. Following the minimax algorithm, one user wants to maximize their utility at each move and the other wants to minimize their utility at each move. I have designed a game that can only be played with one human user and a computer user, but the same algorithm would apply as a game strategy if there were two human users playing.

I will now discuss the code structure of my project. As described in my README file, I have submitted two package folders, one for each version of the game, both including a node file, main file, and Tic Tac Toe file. Each main file is primarily used to call a new instance of the game and to ensure that the program automatically runs again after a game ends. The node files are used to construct the tree structure of the program. While incorporating heuristic search, the root of the tree is an empty board, and each child node is the next move or player turn. The The board is constructed as a 2-D array of size

0 to 2 for the 3x3 board. The computer calculates the heuristic value, or cost, at each depth, as an algorithm for choosing a move to beat the human user.

In order to setup a game board, the Tic Tac Toe “X” and “O” symbols have to be handled properly. For each row or column in the 2-D array, each spot either has an “X”, “O”, or a blank space character. My “addSymbol” string method handles the character input on each row by iterating thru each column position. An error will be returned if the input or array contents are not valid.

There are four ways to win Tic Tac Toe: three matching characters on one of the rows or columns, or three matching characters on the left or right diagonal. After each user or computer turn, four corresponding methods are used to check for a win on the board. The program iterates thru the board checking for three matching symbols in a row. If there are not three matching symbols, then the method will return false and the came will continue. If a win has been detected, the program will output the winner and restart the game with an empty board.

At the core of this project is the AI of the computer player. In order for the computer to win a game of Tic Tac Toe against a human, it has to make optimal moves each turn. I have implemented the minimax algorithm to allow the computer to determine its best move. Our textbook, *Artificial Intelligence: A Modern Approach*, gave me the idea to implement the minimax algorithm using vector and array list structures. The algorithm assumes that the human user is making the best possible move, but it will still work if the human user is not playing optimally. The goal is to determine the next move, or successor node in the game tree. The max or min node is returned after iterating thru the array list based on the heuristic cost values. To improve upon the minimax algorithm, our textbook,

AIMA, suggests implementing alpha-beta pruning. Pruning should effectively ignore parts of the game search tree that do not affect the final state. With pruning, it is possible for either the human or the computer to win with empty spaces left on the game board. The computer user is on the alpha side with a goal of negative infinity and the human user is on the beta side with a goal of positive infinity. The methods involving alpha-beta pruning also use vector structure and return the next best move. The combination of the minimax algorithm using heuristic search and alpha-beta pruning is effective, since the computer is consistently able to win against a human user.

I have described that my Tic Tac Toe program is structured as a 2-D array containing “X” and “O” characters, but I have not described how that corresponds to the console output. Since I did not make a GUI, the entire board has to be reprinted after being updated. In my display method, I have used the +, |, and – characters to form a game board visual in the console. The “X” and “O” characters appear in their correct array positions on the board in the console. In the console, the user is not seeing what is actually stored in the 2-D string array, rather they are viewing a representation of the characters being stored and the occupied game board spaces.

Thus far I have discussed my implementation of Tic Tac Toe in general, mainly focused on the 3x3 board version. The reason for focusing on the 3x3 version is that its simpler structure and algorithms directly correspond to the larger Nine-board version. I had much more success with coding the 3x3 version of the game since the textbook covers the 3x3 game specifically. In order to expand my game for the larger version, I copied my own code from the 3x3 version and modified it. Instead of a 2-D array with three rows and three columns, I changed it to be nine rows and nine columns. Each

method involving checking for a win had to be modified so that (4,4) is the center of the board and so that (2,2) is not the max 2-D array position. I did not modify any of my methods involving the tree structure, minimax algorithm, heuristic search, or alpha-beta pruning. Overall, I kept the program structure the same except for allowing it to handle the larger board size.

In conclusion, I think I successfully designed and coded a Tic Tac Toe game in which the computer AI can beat a human user. I feel confident about my 3x3 game version, but my Nine-Board version is flawed. I thought I could just adjust the 2-D array size to be larger, but the game is more complicated with additional rows and columns. With the center array coordinate of the 9x9 board being (4,4), the fourth row and fourth column become much more probable for a winning arrangement of X's or O's. This changes how effective the search for optimal moves is. I believe that the human user or computer user can win the Nine-Board version, but not consistently. I attempted to create the actual Nine-board game, in which there is a mini game board in each of the spaces of a giant 3x3 board, but I was mostly unsuccessful. My 3x3 Tic Tac Toe game shows that I have successfully implemented the minimax algorithm so that the computer user can make optimal moves to consistently win against a human user.

Note: On the next two pages I have included sample console output and my sources used.

Sample Output

The following images are screenshots of me testing the 3x3 version of my Tic Tac Toe game in Eclipse. In this game, I made the computer go first and it won. At the bottom, you will see that the game automatically replays.

```
Welcome to 3x3 Tic Tac Toe.
The board ranges from 0 to 2 on rows & columns.
Enter 1 to move first, or 2 to go second: 2
Game Board:
-------------
| X |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter row: 2
Enter column: 2
Game Board:
-------------
| X |   |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
Game Board:
-------------
| X | X |   |
-------------
|   | O |   |
-------------
|   |   |   |
-------------
Enter row: 3
Enter column: 1
Game Board:
-------------
| X | X |   |
-------------
|   | O |   |
-------------
| O |   |   |
-------------
```

```
Game Board:
-------------
| X | X | X |
-------------
|   | O |   |
-------------
| O |   |   |
-------------
Computer won! Try again.
Enter 1 to move first, or 2 to go second:
```

Sources Used

1. *Artificial Intelligence: A Modern Approach*, third edition, by Stuart Russell and Peter Norvig. Pages 165-180.
2. Professor Ferguson's lecture slides
3. I used the following website to help me understand the minimax algorithm more and how the tree structure for Tic Tac Toe works:

   http://neverstopbuilding.com/minimax