**Final Project:** Tic Tac Toe **Group Partners:   
Zobia Khalid 3355 (BSSE)  
Syeda Wajiha Zahra 3277 (BSCS)**

CS210 Data Structure and Algorithms (DSA)

The game is basically AI based game. This game is coded without STL and a single player game. The console plays its movement using a minimax algorithm. It also has the concept of alpha-beta pruning (search algorithms) to improve CPU efficiency and form a perfect BST. The data structure basically used in the project is arrays.  
**Tools used:**   
Visual Studio 2013, Dev C++  
**Purpose:**   
The Main purpose of the project is converting the normal two player tic tac toe game into single player game with the help of algorithms. Minmax or minimax, searching algorithm and game tree concept.

**Minimax Algorithm:**

The Mini-max algorithm is a repeating or reverse algorithm used to select the right move for a player who thinks another player is also playing well. It is used primarily for decision making and game ideas to determine a player's good performance, assuming your opponent is also playing well. E.g. Toe, chess etc.  
Uses backlinks to search through the game tree.

In Minimax these players are called maximizer and minimizer. The maximizer tries to get the highest score while the minimizer tries to do the opposite and get the lowest points.

**Alpha Beta Pruning:**

Alpha-beta pruning is a search algorithm that seeks to reduce the number of nodes tested by the minimax algorithm in its search tree. It is a controversial search algorithm used for two-player gaming machines (Tic-tac-toe, Chess, Go, etc.). It stops moving the test when at least one thing is found that proves the movement is worse than the previously tested step. Such movements do not need to be checked continuously. When applied to a standard minimax tree, it returns the same resemblance to minimax, but removes branches that can influence the final decision.

**Game trees:**

To solve games using AI, we will introduce the concept of the game tree. The various aspects of the game are represented by locations in the game tree.

In the game tree, the nodes are arranged in levels corresponding to each player's turn in the game so that the "root" node of the tree is the first place in the game. In tic-tac-toe, this could be an empty grid with no Xs or Os currently playing. Under the roots, in the second level, there are possible situations with the departure of the first player, either X or O. We call these numbers the “child” of the root node.

Each node, in the second level, will continue to have its children in places that can be reached by the movement of the opposing player. This continues, level by level, to countries where the game is over. In tic-tac-toe, this means that one of the player’s gets a third line and wins, or the board is full and the game ends in a draw.

**The Minimax and maximize:**

In order to create an AI game that tries to win the game, we attach a number of numbers to each possible end result. For the board positions where X has three lines for Max to win, we attach the number +10, and similarly, for the positions where Min wins by three Os in a row we attach the number -10. In positions where the board is full and no player wins, we use a neutral number 0 (it doesn't matter what the values ​​are in this order for Max to try to increase the value, and then Min to try to reduce it).  
**Pseudo Code:**

(score, move) maxTurn(game, depth) :

if game is in terminal state :

return (score(game, depth), none)

max = (none, none)

foreach emptySpace in game :

game[emptySpace] = X

currentMove = minTurn(game, depth + 1)

if currentMove.score > max.score:

max = (currentMove.score, emptySpace)

game[emptySpace] = none // reverting change

return max

(score, move) minTurn(game, depth) :

if game is in terminal state :

return (score(game, depth), none)

min = (none, none)

foreach emptySpace in game :

game[emptySpace] = O

currentMove = maxTurn(game, depth + 1)

if currentMove.score < min.score :

min = (currentMove.score, emptySpace)

game[emptySpace] = none // reverting change

return min

int score(game, depth) :

if X has won :

return 10 - depth

else if O has won :

return depth - 10

return 0