**Assignment #1 - Tic-Tac-Toe**

Programmer: Victor Espinoza

SID: 010657450

CECS 475, Section 3; Tu/Th 2:00 - 4:15 P.M.

Lab Section 4

Due: Tuesday, January 26, 2016

Source Code:

//Victor Espinoza

//CECS 475 - Application Programming using .NET

//Assignment #1 - Tic Tac Toe

//Due: 1/26/16

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CECS\_475\_Lab1 {

class Program {

static void Main(string[] args) {

//initialize new Tic Tac Toe object

TicTacToe game = new TicTacToe();

//print out the board

game.PrintBoard();

//start playing a game of Tic Tac Toe

game.Play();

}//close main

}//close class Program

}//close namespace CECS\_475\_Lab1

//Victor Espinoza

//CECS 475 - Application Programming using .NET

//Assignment #1 - Tic Tac Toe

//Due: 1/26/16

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CECS\_475\_Lab1 {

class TicTacToe {

private const int BOARDSIZE = 3; // size of the board

private int[,] board; // board representation

private int player = 1; //variable used to keep track of player

//1 => Player 1, 2 => Player 2

public TicTacToe() {

//declare board

board = new int[3, 3];

//initialize board to all 0's

for (int i = 0; i < BOARDSIZE; i++) {

for (int j = 0; j < BOARDSIZE; j++) {

board[i,j] = 0;

}//end inner for loop

}//end outer for loop

}//end TicTacToe

public void PrintBoard() {

//print the game board including the players who have control of each square

Console.WriteLine("\n -----------------------");

for (int i = 0; i < BOARDSIZE; i++) {

Console.WriteLine("| | | |");

for (int j = 0; j < BOARDSIZE; j++) {

if(board[i,j] == 0)

//print a blank square if nobody has occupied the designated square yet.

Console.Write("| ");

else

//else print the number of the player who owns the designated square

Console.Write("| " + board[i, j] + " ");

}//end inner for loop

Console.WriteLine("|\n| | | |\n -----------------------");

}//end outer for loop

}//close PrintBoard()

public void Play(){

bool validRow, validColumn; //used to validate the user row/column inputs

int rowInput = -1, columnInput = -1, gameWinner = 0; //used to hold the user inputs and

//keep track of the game state

string playerOutput; //used for outputting the correct player.

//continue game until somebody wins or board is completely full.

while (gameWinner == 0) {

//determine/output the correct player information.

playerOutput = (player == 1) ? "Player 1" : "Player 2";

Console.WriteLine(playerOutput + "'s turn.");

validRow = false; //initialize user row input to false

validColumn = false; //initialize user column input to false

while (!validRow) {

try {

//prompt user to enter a valid row value and wait for user input

Console.Write(playerOutput + ": Enter row (0 <= row < 3): ");

//attempt to convert the user input into an integer

rowInput = Convert.ToInt32(Console.ReadLine());

//if the conversion was correct but the number is not within the valid

//range of 0 <= input < 3, then re-prompt the user to enter a valid value

if (rowInput > 2 || rowInput < 0)

Console.WriteLine("Desired row value not within appropriate range. " +

"Please enter an integer value between 0 and 2...");

else

//otherwise the user input was valid. Exit the loop

validRow = true;

}//end try

catch {

//inform the user that they did not enter an integer value and re-prompt

//the row value input.

Console.WriteLine("Not a valid row number. Please enter an integer value "

+ "between 0 and 2...");

}//end catch

}//end while loop

while (!validColumn) {

try {

//prompt user to enter a valid column value and wait for user input

Console.Write(playerOutput + ": Enter column (0 <= row < 3): ");

//attempt to convert the user input into an integer

columnInput = Convert.ToInt32(Console.ReadLine());

//if the conversion was correct but the number is not within the valid

//range of 0 <= input < 3, then re-prompt the user to enter a valid value

if (columnInput > 2 || columnInput < 0)

Console.WriteLine("Desired column value not within appropriate range. " +

"Please enter an integer value between 0 and 2...");

else

//otherwise the user input was valid. Exit the loop

validColumn = true;

}//end try

catch {

//inform the user that they did not enter an integer value and re-prompt

//the row value input.

Console.WriteLine("Not a valid column number. Please enter an integer value "

+ "between 0 and 2...");

}//end catch

}//end while loop

//check to see if the user's desired move is valid, meaning that I need to ensure that

//the desired square on the board is not taken by either player already

if (!ValidMove(this.board, rowInput, columnInput))

//inform user of the situation and re-prompt them to enter another move.

Console.WriteLine("That square is already taken. You will now be re-prompted "

+ "to enter a valid move.");

else {

//update the board by filling in the player's valid move with the appropriate

//value that corresponds to that player.

board[rowInput, columnInput] = player;

//print the updated tic tac toe board

PrintBoard();

//determine if the game is over yet (somebody won / there is a tie)

gameWinner = GameStatus(board);

//if the game is not over yet, then alternate between the players.

if (gameWinner == 0)

player = (player == 1) ? 2 : 1;

}//end else

}//end while

//Print out the appropriate winner of the game.

if (gameWinner == 1)

Console.WriteLine("\nPlayer 1 Wins!");

else if (gameWinner == 2)

Console.WriteLine("\nPlayer 2 Wins!");

else

Console.WriteLine("\nTie Game!");

}//close Play()

public bool ValidMove(int [,] gameBoard, int row, int col){

//Check to see if the user entered a valid move. This will be true if the value

//indicated by the approriate row/column indices in the gameBoard array is a 0.

//If it is not a 0, then that means that a player has already claimed that spot

//and the move is not valid.

return gameBoard[row,col] == 0;

}//close ValidateMove()

public int GameStatus(int [,] gameBoard) {

//Determine the winner

for (int i = 0; i < BOARDSIZE; i++){

//check rows and columns for winning Player 1 moves. Also, check both diagonal

//cases (I only do this once because it would be inefficient to

//check these cases three times).

if ((gameBoard[i,0] == 1 && gameBoard[i,1] == 1 && gameBoard[i,2] == 1) ||

(gameBoard[0,i] == 1 && gameBoard[1,i] == 1 && gameBoard[2,i] == 1) || (i

== 0 && ((gameBoard[0,0] == 1 && gameBoard[1,1] == 1 && gameBoard[2,2] == 1)

|| (gameBoard[0,2] == 1 && gameBoard[1,1] == 1 && gameBoard[2,0] == 1))))

return 1;

//check rows and columns for winning Player 2 moves. Also, check both diagonal

//cases (I only do this once because it would be inefficient to

//check these cases three times).

if ((gameBoard[i, 0] == 2 && gameBoard[i, 1] == 2 && gameBoard[i, 2] == 2) ||

(gameBoard[0, i] == 2 && gameBoard[1, i] == 2 && gameBoard[2, i] == 2) || (i

== 0 && ((gameBoard[0, 0] == 2 && gameBoard[1, 1] == 2 && gameBoard[2, 2] == 2)

|| (gameBoard[0, 2] == 2 && gameBoard[1, 1] == 2 && gameBoard[2, 0] == 2))))

return 2;

}//end for loop

//If I reach the end of the loop, then that means that nobody has won yet. I now

//need to check to see if the board still has valid moves left or if it is full.

//If it does still have moves left, then I return a value of 0, otherwise I

//return a value of 3 to indicate that the game ended in a tie.

for (int i = 0; i < BOARDSIZE; i++) {

for (int j = 0; j < BOARDSIZE; j++) {

if(board[i,j] == 0){

return 0;

}//end if

}//end nested for loop

}//end outer for loop

//if I reach this code that means that the board is full and no more moves

//can be made, which results in a tie. I return a value of 3 to signify this.

return 3;

}//end GameStatus

}//close class TicTacToe

}//close namespace CECS\_475\_Lab1