Name: ThankGod Ofurum

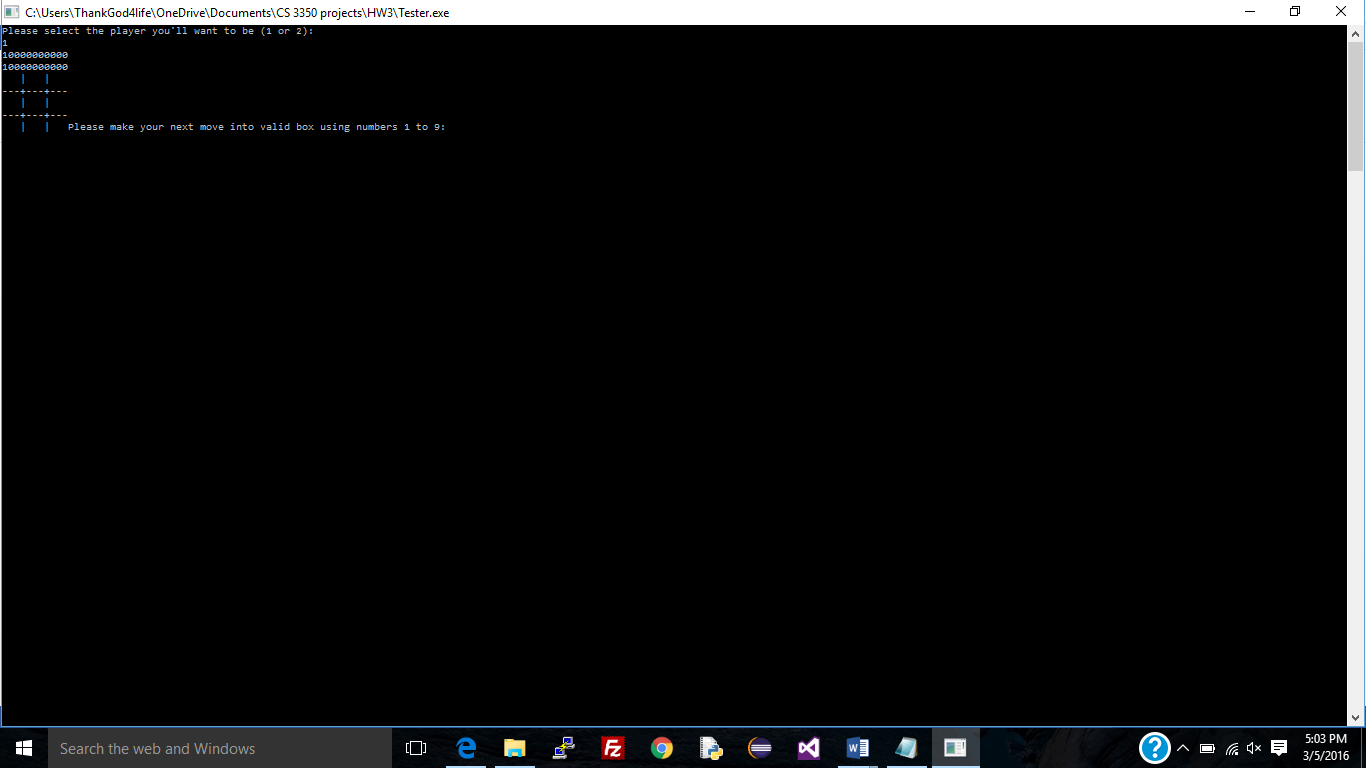
Professor: Francois Neville

Class: CS 3350

Title: TIC TAC TOE

For this assignment, we were required to implement a fully functional Tic Tac Toe console application from an almost fully implemented skeleton code and console application provided for us. This console application was to resolve most of the problems encountered in the skeleton console application such as selecting occupied locations and was to include as many special features as desired by its designer. I, being the over achiever that I believe I am, once I heard of the possibility of implementing an AI (artificial intelligence) into the console application, just couldn't resist the challenge of accomplishing this. In this report, I'll describe the various features and implementations included in the application code, the difficulties and challenges involved in completing these implementations and the description of the commands provided in the assignment and when they should be used.

I'll begin by describing the process and coding involved in my implementation of the Tic Tac Toe application. First I had to understand how the bitwise AND worked, as up till now I hadn't encountered this in any coding format, yet this was the most frequently occurring command in the entire application code. I accomplished this by making use of the C# help website provided in the class assignment web page which explained the concept of bitwise AND including its use in bit masking. Second I had to understand the concept of object-oriented programming in an IDE environment which included assigning types to each object in the code, this proved to be the most annoying aspect of the entire coding process as I wasn't used to thinking of each value as an object, but thankfully I began to realize a pattern emerging thus making the IDE a lot easier to deal with. Finally, I had to understand the logic behind winning in the game of Tic Tac Toe and all the possibilities available to accomplish this. I then had to find a way to counter all these possibilities while still providing a gateway for the program to win, my best friend in all this was the bitwise AND command. Below is a display of the console application.



In order to make my implementation of the Tic Tac Toe game as original from the skeleton application as possible, I not only decided to implement an AI, but I also allowed the user to decide whether to go first or second - in order to be fair (the AI still doesn't lose). Below is the body of code responsible for this implementation.

//select play number

System.Console.WriteLine("Please select the player you'll want to be (1 or 2): ");

int Player = int.Parse(System.Console.ReadLine());

while (Player != 2 && Player != 1)

{

System.Console.WriteLine("Please select the player you'll want to be (1 or 2): ");

Player = int.Parse(System.Console.ReadLine());

}

I also decided to make use of bit masking in order to separate between player's spots in order to prevent placing symbols in the exact same spot. Below is the code responsible for this implementation.

if (int.Parse(input) <= 9 && int.Parse(input) >= 1)

{

result = System.Convert.ToString((playerPositions[0] + playerPositions[1]) & (int)(Math.Pow(2, double.Parse(input) - 1)));

if (double.Parse(result) != Math.Pow(2, double.Parse(input) - 1)) {

validMove = true;

}

else

{

System.Console.WriteLine("You entered an invalid number ");

correct = false;

}

}

I also made use of bit masking in drawing the symbols on the board and in determining who the winner is. Below is the code responsible for implementing both the actions described above.

int winner = 0;

int[] winningMasks = { 7, 56, 448, 73, 146, 292, 84, 273 };

// Determine if there is a winner

for (int i = 0; i < winningMasks.Length; i++ ) {

string p1 = (playerPositions[0] & winningMasks[i]).ToString();

string p2 = (playerPositions[1] & winningMasks[i]).ToString();

if (p1 == winningMasks[i].ToString())

{

winner = 1;

}

else if (p2 == winningMasks[i].ToString())

{

winner = 2;

}

}

char token;

string p1 = (playerPositions[0] & position).ToString();

string p2 = (playerPositions[1] & position).ToString();

if (p1 == position.ToString()) {

token = char.Parse("X");

}

else if (p2 == position.ToString())

{

token = char.Parse("O");

}

else {

token = char.Parse(" ");

}

Finally was the implementation of the AI, and this was the trickiest and most time-consuming aspect of the entire assignment as there're multiple possibilities available to a player of Tic Tac Toe, and inhibiting each and every one of these possibilities while still seeking to win the game is truly difficult, unless you're able to account for each and every position in the board. That's exactly what I did, by bit masking each move of the human player with the available positions in the board, I was able to keep track of each and every move made by the player each turn. Then there was the issue of making sure the AI made the right move each turn (which is generally middle position or edges), and this I accomplished by setting conditions on the outputs of the AI based on the current turn. Below is a partial listing of the code that accomplished all this (Feel free to play against the AI, bet you can't beat it).

if (turn == 1)

{

output = System.Convert.ToString((Human) & (int)(Math.Pow(2, midBoard)));

if (int.Parse(output) != (int) Math.Pow(2, midBoard)) { input = midBoard.ToString(); }

else {

value = rnd.Next(0, 8);

do { value = rnd.Next(0, 8); }

while (value % 2 != 0 || value == midBoard);

input = value.ToString();

}

}

As for the commands listed in the provided code, most of them are preprocessor directives and are specific to the C# system compiler, and their meanings are as follows. The #define CSHARP2 command is a token/ symbol that can be used later in the code to alter the way the compiler behaves, in this case, to clear the console, the token (CSHARP2) must be used with a #if #end if block. Using System; refers to a namespace which is a construct that acts as a container to provide a way to group similar classes, and System is a namespace prebuilt into the C# compiler consisting of commonly used classes such as Math. #pragma warning disable 1030 instructs the compiler about compiling the file under consideration, in this case, it stops the compiler from reporting warning number 1030. Namespace yy refers to the construct which contains the group of classes that are specific to yy-every program must have a namespace in order to gain access to its classes. #region, #end region helps a programmer, when coding in the visual studio IDE, to compress a region of code into a single block which can then be collapsed or expanded at will-useful when writing large bodies of code. #if CSHARP2, #end if allows a programmer to conditional compile a piece of code based off the defined symbol, in this case, CSHARP2-useful for testing and debugging when compiling the code.