To all CIS 330 Students –

We got a little bit behind yesterday, so I am sending along via email some of the Tic-Tac-Toe code you will need AND (in a separate message) portions of the Activity (or Behavior) Diagram you need to complete by next Wednesday (for HW #1).

First, the code. At the end of this message is the code for the two classes you need for TTT: the Player Model and the Internal Board Model. When the time comes (NOT YET!!) I urge you to create a folder named Classes in your project and create within this folder two C# class files, one for each – perhaps named BoardModel and PlayerModel (or something similar). Copy the code included in this message directly into these classes.

For now: Study these two classes and extract out for each the Class Diagram information you need for your Wednesday 9-20 homework. As indicated in the TTT specifications document (on the website) each of your two class diagrams should contain 1) an annotated record (a brief description of the purpose) of all of the attributes of each class, and 2) an annotated record of each of the methods in the class. For the methods, your annotation should include:

1. The name of the method
2. A description of the purpose of the method
3. A description of what (if anything) is returned by the method
4. A description of what each method argument (if any) represents.

NOTE: We will not be keeping track of the count of the number of wins per player for your game, but I left this attribute in the Player Model class anyway.

Tomorrow I will send you a sample of what I expect your Activity (Behavior) Diagram to look like.

// Board model class

// Class that models the attributes and methods for the internal represenation of the TicTacToe Board

// Responsible for data stores and methods required to model the TicTacToe Board

// Frank L. Friedman

// January 11, 2016 (Rev 3)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_TicTacToeV1

{

class BoardModel

{

const int SIZE = 9; // Size of board representation

private char [] internalBoardRep = new char[SIZE]; // The board representation itself

// Constructor - resets the game

public BoardModel ()

{

resetGame();

} // end New

// Records information each time a player takes a turn

public void recordTurn

(PlayerModel currentPlayer, // IN: represents the player (object) making the current move

int cellId) // IN: represents ID of the cell selected by the player

{

internalBoardRep[cellId] = currentPlayer.Icon;

return;

} // end recordTurn

// Resets the board for a new game

public void resetGame()

{

clearInternalBoardRep();

} // end resetGame

// Actually clears each cell in the Board by resetting it to Blank

private void clearInternalBoardRep()

{

for (int i = 0; i < SIZE; i++)

{

internalBoardRep[i] = ' ';

}

} // end clearInternalBoardRep

// Determines if there is a winner by checking rows, columns and diagonals

// Returns: True if there is a winner and False otherwise

public Boolean hasPlayerWon

(char currentPlayerIcon) // IN: icon for the current player

{

Boolean result = false;

if (isRowWin(currentPlayerIcon) || isColumnWin(currentPlayerIcon) || isDiagonalWin(currentPlayerIcon))

{

result = true;

}

return result;

} // end hasPlayerWon

// Checks rows for a winner

private Boolean isRowWin

(Char currentPlayerIcon) // IN: icon of curent player

{

return isWin(0, 1, 2, currentPlayerIcon) || isWin(3, 4, 5, currentPlayerIcon) || isWin(6, 7, 8, currentPlayerIcon);

} // end isRowWin

// Checks columns for a winner

private Boolean isColumnWin

(Char currentPlayerIcon) // IN: icon of current player

{

return isWin(0, 3, 6, currentPlayerIcon) || isWin(1, 4, 7, currentPlayerIcon) || isWin(2, 5, 8, currentPlayerIcon);

} // end isRowWin

// Checks diagonals for a winner

private Boolean isDiagonalWin

(char currentPlayerIcon) // IN: icon of current player

{

return isWin(0, 4, 8, currentPlayerIcon) || isWin(2, 4, 6, currentPlayerIcon);

} // end isdiagonalWin

// Checks the three Tic-Tac-Toe Board blocks to see if they contain the same character

private Boolean isWin

(int block1, int block2, int block3, // IN: indexes of the three blocks to be checked

char currentPlayerIcon) // IN: icon of current player

{

Boolean result = false;

if (internalBoardRep[block1] == currentPlayerIcon &&

internalBoardRep[block2] == currentPlayerIcon &&

internalBoardRep[block3] == currentPlayerIcon )

{

result = true;

}

return result;

} // end isWin

} // end class BoardModel

} // end namespace

// Player model class

// Class that models the attributes and methods for one player

// Responsible for data stores and methods required to model a player

// Frank L. Friedman

// January 11, 2016 (Rev 3)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CS\_TicTacToeV1

{

class PlayerModel

{

private string hiddenName;

private int hiddenWins;

private char hiddenIcon;

public PlayerModel (string name, char icon)

{

hiddenName = name;

hiddenIcon = icon;

hiddenWins = 0;

} // end New

// Read only properties

public char Icon

{

get {return hiddenIcon;}

} // end property Icon

public string Name

{

get {return hiddenName;}

} // end property Name

public int Wins

{

get {return hiddenWins;}

} // end property Wins

// Increment number of wins for this player

public void won()

{

hiddenWins = hiddenWins + 1;

} // end isWin

} // end class PlayerModel

} // end namespace CIS\_TicTacToeV1