# PA2

PROGRAMMING ASSIGNMENT 2
BME 121, FALL 2016

Due Fri Oct 7 at 4:30 pm

## ASSIGNMENT

The file pa2Start.cs contains a nearly working implementation of a game called Connect Four. It is written as a collection of static helper methods in the Program class. The Main method plays the game by calling the other methods. Game information, such as the current player and the state of the game board, is held in a set of static fields of the Program class which are accessible from every method. Six of the methods are implemented as stubs which do nothing and, if a result is required, return a constant.

You can try a complete working implementation of the game using the file SampleSolution.zip. Download this file, extract its contents into a folder, open PowerShell in that folder, and use "dotnet .\pa2.dll" (Windows) or "dotnet ./pa2.dll" (Linux/Mac). A convenient way to demonstrate the game is to use the "ai" kind for both players.

If you run pa2Start.cs under the same conditions, it will endlessly loop with one player attempting to play in column 0. (You can interrupt such a misbehaving program by typing the "Ctrl" and "C" keys together.) A logical sequence of fixing the six methods and testing with two "ai" players, is as follows.

1. Complete the SwitchPlayers method. If this is working properly, the endless loop will be back and forth between the "O" player and the "X" player, always selecting column 0.
2. Complete the SelectRandomColumn method. If this is working properly, the players will randomly select columns from 0 to 6.
3. Complete the PlayInColumn method. If this is working properly, the board will show the players' symbols added to the selected columns.
4. Complete the IsValidPlay method. If this is working properly, the column selections will be limited to columns which are not already filled. Eventually, the program will freeze with one of the players endlessly seeking a valid column because the game board is full.
5. Complete the IsBoardFull method. If this is working properly, the game will end in a draw.
6. Complete the CurrentPlayerWins method. If this is working properly, the game will now play as expected.

The sections below give more details on each of the six methods.

### SWITCHPLAYERS METHOD

The SwitchPlayers method will access the fields describing the current player. If the current player is player "O", the fields will be changed to represent player "X". If the current player is player "X", the fields will be changed to represent player "O". For any error situation, the method can simply do nothing.

### SELECTRANDOMCOLUMN METHOD

The SelectRandomColumn method will use the static randomNumberGenerator field to randomly generate a column number in the random from zero to one less than the number of columns in the game (field gameCols). It will repeatedly generate such random columns until method IsValidPlay evaluates as true for that column. It will then return the column.

### PLAYINCOLUMN

The PlayInColumn method puts the symbol of the current player into the specified column of the game board. The array gameBoard is two dimensional and iniatially filled with "null" entries. This method should copy the current player's symbol, i.e., "O" or "X", into the smallest unoccupied row of the specified column. If the result would be outside the bounds of the array, the method should simply do nothing.

### ISVALIDPLAY

The IsValidPlay method checks whether the indicated column has space remaining to play. It can return true if the top row of that column is still "null". For a column out of range or a full column, it can return false.

### ISBOARDFULL

The IsBoardFull method checks whether any column on the board has space remaining to play. It can return false if the top row in any column contains "null" or false otherwise.

### CURRENTPLAYERWINS

The CurrentPlayerWins method checks the game board to determine if there is a sequence of four-in-a-row of the current player's symbol vertically, horizontally, or diagonally. This can be done by looping over every game-board position as the starting position. From each starting position, if there are three more positions above, check for a vertical four-in-a-row. If there are three more positions to the right, check for a horizontal four-in-a row. If there are three more positions both above and to the right, check for a diagonal (up and to the right) four-in-a-row. If there are three more positions below and to the right, check for a diagonal (down and to the right) four-in-a-row. If any of these checks finds a four-in-a-row sequence of the current player's symbol, return true. Otherwise, return false.

## LEARNING OUTCOMES

You are basically doing the last step in an iterative design cycle. This design went through four or five such iterations to settle on the given data representation of the game and its decomposition into methods. For each such cycle, we had to program all the methods at least to a point where we could assess the whole design. Some desired outcomes from the pieces you are doing are as follows.

1. Appreciate the value of good comments and good method/variable/field names.
2. Experience how a modular decomposition makes programming easier by allowing you to focus on one small task at a time.
3. Gather more experience with loops, selection, parameter passing, and expressions.
4. Experience a situation where many methods accessing shared data fields makes sense.
5. Experience the cognitive difficulties that come with programs which don't fit on one screen.
6. Gain experience reading and understanding C# code which you did not write.
7. Make a program which does something at least vaguely interesting.

You will see and do iterative design and design decompositions throughout engineering. Because programming is an activity drowning in detail, it is difficult to start doing such full-scale design until you get reasonably comfortable with enough low-level pieces. Thus, in this course, we mostly give you the high-level design decompositions.

## SUBMISSION

Name your C# program file as pa2.cs. Use Bme121.Pa2 as your namespace identifier. Include the standard doc-comment block. Submit pa2.cs at the following url.

https://georgefreeman.ca/fileuploader