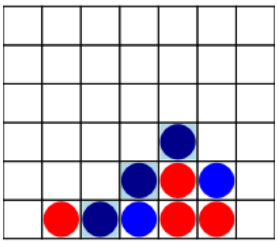
# 159.272 Programming Paradigms
# Assignment 2
# Playing a game: Four in a Line

**Total marks:** 15 marks
**Due:** 8pm Tuesday - June 5th 2018

Playing games interactively is a common computational activity. We can use abstraction to aide in the design of games by separating the game logic from the actual user interaction required to play the game. No matter the type of user interface provided to play the game (text-based, desktop environment, web browser), the underlying logic remains the same.

In this assignment, you will complete Scala implementation of Four in a Line (also known as Connect Four, Captain's Mistress, Four Up, Plot Four, Find Four, Fourplay, Four in a Row), a two-player connection game in which the players first choose a color and then take turns dropping colored discs into a seven-column, six-row, vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The object of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before your opponent. Four in a Line is a strongly solved game. The first player can always win by playing the right moves.



See Maths is Fun to practice the game, or read the wikiHow entry for tips on how to win.

The Scala code for this assignment consists of three Scala files (assignment2_2018.zip):

**`FourinLine.scala`**

**`GameTree.scala`**

**`Game.scala`**

The game logic is contained in the module **`FourinLine.scala`**. Please download all three files and include them in your Eclipse project.

## Instructions:

1. Create a new Scala project in Eclipse.
2. Add the files (FourInLine.scala, Game.scala, GameTree.scala) to the src folder of the project.
3. Complete the functions in FourInLine.scala. Replace the ??? with your definitions.
4. Game.scala defines the main class of the application with the main method. Run and test your changes in the main method defined in Game.scala. The final function call in the main method starts the game. Right click and run the main class as a Scala

application as you did at the tutorials.

Optionally (if you want to add the provided tests.)

1. Add the file FourInLineTests.scala to the src folder of your project.
2. Unzip the contents of testBoardFiles.zip and copy the files to your project's src folder (These text files are a single board state of a game as they would be represented by the showGameState function.)
3. Add calls to the functions defined in FourInLineTests.scala to the main method in Game.scala.

Before you submit your solution, remove any calls to functions in FourInLineTests.scala from any submitted code.

## Task

**Your main task** is to define the functions in this module using the provided data types.

The other two modules have been provided for you. **`GameTree.scala`** constructs a game tree and uses the minimax strategy with limited lookahead to determine the best move for the computer to make. The **main** method is in **Game.scala** module, so to run the program you need to run your **Game.scala** object (*right-click ➔ run as Scala application*). This will start a new game and will show you the following message on the screen:

```
Welcome to four-in-line
Is Red player to be human or computer?
```

The game is yet to be implemented. After selecting players, the program will throw an exception (`scala.NotImplementedError`), as the methods in **FourInLine.scala** are not implemented yet**.** You will need to write your own code in the functions given in **FourInLine.scala .**

In lectures, we will explain the concepts behind the **`GameTree.scala`** code. You should work through the code and make an effort to understand how these concepts have been implemented in this module. The most important concern is to ensure that your functions defined in **`FourinLine.scala`** return sensible values of the correct types, so that the other modules can sensibly use your completed module.

**Your second task** considers the **`estimate`** function in **`GameTree.scala,`** which takes a **`Player`** and **`GameState values`** and returns an integer value, which "rates" the result of the current game configuration, for that player.

**Your task is** to "make this function smarter". The vanilla version returns a score for the given player and the given configuration, based on whether or not the player has won the game. Can you make this function smarter? This function can be modified to account for strategic concerns, e.g. placing pieces in the centre columns gives you more options, clumping your pieces together can provide more ways to make four in a line. The important point is that you can modify this function without changing **any** of the other code, and end up with a smarter program. Think about what you would need to do to alter this program behavior using other programming paradigms.

## What to submit

When you have a finished version, submit your Eclipse project (Export as an Eclipse Archive file) for marking. You will get (up to) 12 marks for a correct working version of **`FourinLine.scala`**, modulated by the use of good functional programming style, and (up to) 3 marks for an updated version of **`GameTree.scala`**, modulated by the sophistication of your improvements.

Submit your work on Stream by **Tuesday June 5th**.

There is a penalty for late submissions. The penalty is 10% deducted from the total possible mark for every day delay in submission (one day late – 90%, two days – 80% etc).

You are expected to manage your source code, this includes making frequent backups. "The cat ate my source code" is not a valid excuse for late or missing submissions. Consider managing your code in a *private* repository (bitbucket or similar). Private is essential here to avoid plagiarism, we reserve the right to deduct marks from your submission if your repository was public.