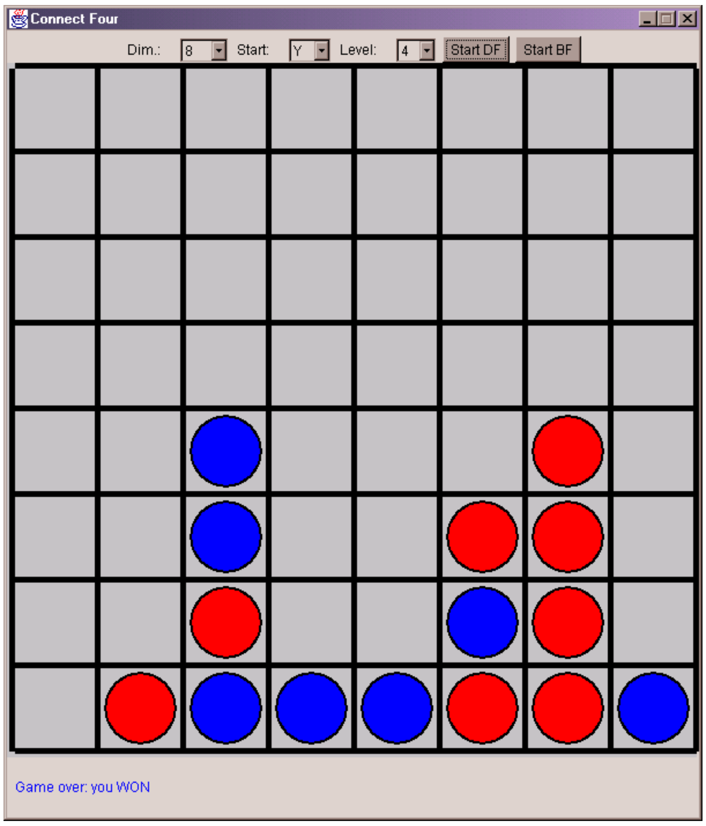
Assignment 2

Context

Connect-4 is a game involving two players. Each player has a symbol (e.g. a coloured disc). The game is played on a grid of equal-length sides containing squares. The players take turns placing their discs on (the lowest-possible) empty squares of the chosen column within the grid until either one player has four of their discs in an uninterrupted line (horizontally, vertically, or diagonally) or until the grid is full. If a line of four like discs can be achieved that player wins and the other loses. If the grid is filled prior to the creation of a line-of-four the game is a draw.

The assignment is to construct a program that will allow the game of Connect-4 to be played. An example implementation of the Connect-4 game is the following:



Dim.: 8 Start: Y Level: 4

Game over: you WON

The application of this game to computing is the design of a program that allows a user to play against the computer. This will ultimately involve the use of game trees. Game trees are general trees in which each tree node is a particular state (i.e. grid) of the game. Each level in the tree possesses as many sibling nodes as there are possible moves from the parent node. The other difference from general trees is that each node will also have a level number;

the root node will be at level 0, its children will be at level 1, their children will be at level 2, and so on.

Once implemented, the program will generate a number of levels of possible moves from the current grid. The 'best' grid will then be selected as the computer's move (and the other grids will be discarded). Consequently the grid representation will not only contain the squares but will also possess a numeric value that indicates the worth of the grid — in terms of producing a win for the computer.

Requirements

Your task is to adopt the sample specification produced for assignment one and implement the abstract data types (ADTs) and additional operations required to implement Connect-4. This will involve the construction/completion/use of thirty separate Java programs; the majority of these will be interfaces and classes for each main ADT or classes required for their implementation (e.g. exceptions or nodes of trees, etc.). The final Java program will be the harness program (`Assignment2.java`).

You are required to consider which move the computer player should make through both depth- and breadth-first expansion of the game tree.

This assignment will require you to utilise the implementation of three high-level data structures: a stack, a queue and a general tree. The tree will be used to store the encoded grid representation representing the current state of the game with successive potential moves being the children of each game tree node. As will be discussed in lectures, the stack will be used for the depth-first traversal and the queue for the breadth-first traversal. You will need to complete the implementation of the `GridType`, `LocationType`, `SquareType`, `SymbolType`, `QueueType`, `StackType`, and `TreeType` ADTs.

You need not implement `IndexType`, `DimensionType`, `LevelNumberType`, or `WorthType` — `int`s will suffice. You need not implement `InputType` either, and `ColourType` can be represented using the existing class `java.awt.Color`.

There are some additional variations from the sample assignment one solution (distributed with this assignment question):

- Any operations marked in the sample solution "*Can be implemented in Java directly.*" need not be implemented.
- The `DisplayType` is implemented using the Java graphics context (an object from the class `java.awt.Graphics`) and you don't need to understand what this is or how this is used. Setter/getter methods are not necessary.
- The `GameTreeType` was specified recursively. This necessitates the use of a node class (`TNode` in this case) to represent the internals of a game tree node.

- Because a depth-first and a breadth-first solution is required, a stack and a queue are needed to complete the `GameTreeType` methods (as discussed in lectures). There are thus two versions of `generateLevel()` and `buildGame()` — one pair for each traversal method.
- In `Connect4Type`, the setter/getter methods have been omitted as the user cannot call these directly. `whoStarts()`, `whatGridSize()`, and `howSmartIsComputer()` have been implemented as `actionPerformed()` and within `play()`; `whatIsHumanMove()` has been implemented as `mouseReleased()` — you don't need to understand the mouse-related methods; `movePossible()`, `makeHumanMove()`, `makeComputerMove()`, and `showConnect4()` have been implemented as `paint()`; and `gameOver()` has been implemented within `play()`.

By default, the program will display lots of output to help with debugging — set `TRACING` to `false` in each class file to prevent this. Also by default, the program will play human *vs* human and not use game trees. Once you have completed `GridType`, `LocationType`, `SquareType`, and `SymbolType` and have it working properly you should complete `QueueType`, `StackType`, and `TreeType` and change `HUMAN_VS_COMPUTER` to `true` to obtain human vs computer behaviour.

Procedure

A number of data structures, types, and methods have been supplied. Many have been supplied as a guide, others have been supplied because the time/complexity involved is too great for you to write them. *Do not alter any supplied code without express written permission. The only modifications required are the completion of* ***methods*** *where the body is entirely missing. All required classes are defined. All required interfaces are defined completely.*

You should obtain the folder entitled `Assignment2.zip` from the unit MyLO web page and study the sample solution to assignment 1. This solution is the specification that you must now implement in this assignment. Ensure you understand how these files are to fit together and what operations must be included within each program file. A project file has also been included for Xinox Software JCreator although you may develop your solution using any Java compiler. (N.B. the final submission must function in Xinox Software JCreator using the supplied project file.) Don't forget: if you amend a class that is used by other classes you must recompile/rebuild all files.

You should attempt to think about this assignment early. Come and seek help when you require it, do not procrastinate.

Submission Instructions

You are required to make a paper *and* an electronic submission. Details will be provided by your lecturer.

The *paper submission* should comprise a signed cover page, print-outs of all altered ".java" files, screen-shots of the output from the solved game. You are required to produce both a depth-first and a breadth-first solution.

For *electronic submission* we require your *complete* project folder to be provided.

Marking Scheme

Correctness	45%
Structure	35%
Comments/Layout	20%

This assignment is worth 10% of the semester's total marks.

Deductions will be made if you use terse or obfuscatory (complicated) code as well as for non-compliance with submission instructions. Your assignment will not be marked if the coversheet is missing or unsigned.

Due Date

Friday of week 14.

Dr Julian Dermoudy
23/4/'11