CS 158 Lab 22 April 27, 2010

n-ary Trees

1. In this lab we will obtain a little experience in dealing with trees that are not binary. There are many ways to implement them, but we will use one of the more obvious ones: each node will contain an **ArrayList** of nodes in addition to its data. We will do this in the context of the game tic-tac-toe. Begin by creating a new Java project called "N-ary Trees", and add a package called **tree**. In this package, add a **Node** class that has a **Board** as data and an **ArrayList** of **Node**s. Create the **Board** class in this package as well, and in it place an array of 9 characters, and use a constructor to instantiate the array and initialize it to contain 9 period characters ('.'). You could use a blank instead, but they may not show up nicely on some terminals when you try to print out a board. Add a constructor to the **Node** class that instantiates the **ArrayList** and the **Board**. You will find it helpful to have a couple of methods in the **Board** class as well. Create a method called **copy** that takes no arguments, but returns a new **Board** that is identical to *this* one (the encapsulated arrays must be identical). Create another method called **print** that prints out all of the characters in the encapsulated array on one line.

2. We will be constructing a "game tree" that consists of all the possible moves starting at a given board position. So it will be helpful to have a method that makes all possible moves starting from a given node. This method can be a responsibility of a node, but the method will also need to know whose move it is. So create a method called **makeAllPossibleMoves** that takes a character argument that represents either an 'X' or an 'O' and then places that character in a new board that is identical to the node's encapsulated board, and does this repeatedly, creating new boards for every space in the board that is not already taken up with an 'X' or an 'O'. Each time a new board is built, also create a new node and put that board into it and add it to the **ArrayList**. So at the end of this method, the node will acquire up to 9 children. We will test this method as we go along, so for now, create a new class in this package called **TicTacToe** with a main method. In that main, create a root node, tell it to make all possible moves with 'X' as the player to move, and then print on the console all of the boards, one per line. Your output should look like this:

```
X........
.X.......
..X......
...X.....
....X....
.....X...
......X..
.......X.
........X
```

3. Add a method to the **Node** class that returns the number of children (just "immediate children", not grandchildren, etc.). Add a JUnit test class that contains a test for this method that creates a root node, makes all possible moves, and asserts that the root now has 9 children. Also in this test, continue by getting the first child of the root, having it make all possible moves using 'O' as the player, and then assert that this child has 8 children.

4. It will be helpful to be able to print the boards that are possible from a given position (you already did this in the main method of the **TicTacToe** class). So add a method to the **Node** class called **printChildrenBoards** that prints all the boards of all the children of the node, one per line. To test this method, replace the code in the main of **TicTacToe** that prints the boards with a call to **root.printChildrenBoards( )**. If that works correctly, then get a child of the root, have it make all possible moves with 'O' as the player, then ask it to print all its children's boards. The output should look like this:

```
X0.......
X.0......
X..0.....
X...0....
X....0...
X.....0..
X......0.
X.......0
```

5. The size of game trees gets to be very big. We need to know how big a tree is starting from any given node. Add a static method to the **TicTacToe** class called **sizeOfTree** that takes a **Node** argument and recursively computes the total number of nodes in the entire tree starting from that node (preorder traversal works well for this). Use a JUnit test for this method that creates a root, asserts that the size of the tree is 1, then has the root make all possible moves, asserts that the size of the tree is now 10. Then make each node in the root's list of nodes make all possible moves, and then assert that the size of the tree is now 82 (make sure you understand why that is).

6. Write a method of the **Board** class called **prettyPrint** that prints the board on the console in an easily understandable form. For example, after making all possible moves for the first two players and getting the first node each time, the board position should look something like this:

```
X0.
...
...
```

7. On Blackboard under Content you should find a folder called Tic Tac Toe. Inside that folder are the **Board** class (with an extra method or two), another class called **Player** that contains a method for getting moves from a person player, and several new methods in the **TicTacToe** class that you should find very useful in building a working program that plays somewhat intelligently with a human player. Put the "puzzle pieces" together, adding anything else you need to obtain a working program. Have fun!