**Forest Fire FAQ**

***Why is my*** *makeChange* ***method returning such a large value?***

Your program is counting tons of duplicate solutions. Suppose you are making change for nine cents; some duplicate solutions would be:

1, 1, 1, 1, 5

1, 1, 1, 5, 1

1, 1, 5, 1, 1

Four pennies and one nickel is the same combination of change as three pennies, a nickel, and another penny. One way around this problem is to use a List of coin denominations, such that you only use each coin, in all combinations, in order, without returning to it again for the same combination in a different order.

Here is my "client" method for makeChange:

/\*\* returns number of ways to make change given denominations in *coins* \*/

public int makeChange(List<Integer> coins, int amount)

{

return makeChangeHelper(coins, 0, 0, amount);

}

Where coins, for normal US coins, would be supplied as Arrays.asList(1, 5, 10, 25). Here is the signature of the private method that does the actual recursion:

private int makeChangeHelper(List<Integer> coins, int i, int current, int amount)

Rather than the "normal" backtracking calls of, e.g. for a nickel, makeChangeHelper(total + 5, amount), you would instead traverse the list, adding the element at the current index to the current total. This will prevent the duplicates as, once you've tried all combinations, you don't return to the same denomination and try all other permutations of the same combination.

If the above doesn't make sense, try watching the program run with a small amount. If that doesn't work, panic for a little while then move on.

***How do I animate the ForestFire program (to watch it run)?***

There are many ways this could be accomplished. Here are some hints for one way, in no particular order:

* As is, the controller class (Fire.java) makes the frame (which contains the panel (the view class, FireView.java) that does the drawing) visible *after* the model has finished solving. This would need to be changed.
* The view class should use a javax.swing.Timer to update the view every so often (e.g. 1ms).
  + The FireView class should implement the ActionListener interface, to listen for Timer events.
  + The model class (FireModel.java) should start and stop the timer as necessary.
* The model class should sleep a small amount, e.g. 1ms, each iteration of the solve method, so that it doesn't finish solving the problem before the view has time to repaint itself.

Don't ask me to tell you how to accomplish the above; you'll have to learn on your own. If Google isn't working for you, here is a good resource for learning the basics of the Java graphics libraries:

* <http://chortle.ccsu.edu/java5/index.html#07>
  + Part 7 and Part 11 cover Java graphics / GUI components