Dynamic Programming Notes: World Series Problem

The World Series involves two teams that play a best of 7 games contest. The first team to win 4 games wins the World Series. (In the most recent year, 2007, that team was the Boston Red Sox!!!)

To generalize the problem slightly, we might as the following question:

Given that in a single contest between team A and team B, the probability team A wins is p, (and the probability that team B wins is 1-p, so we are precluding ties), what is the probability in i+j games played, that team A will win i games and team B will win j games. Let this value be denoted by the function p(i,j).

Using some mathematics, we can answer the question with the following formula (from the binomial theorem):

$$p(i, j) = \binom{i+j}{i} p^i (1-p)^j$$

An alternative way to solve the problem involves dynamic programming. In order to obtain the dynamic programming solution, we must first develop a recursive formula for the function p(i,j). In order for team A to have won i games and team B to have won j games, before the last game, either A won i and B won j-1 OR A won i-1 and B won j. Here is a recursive formula that captures that reasoning:

$$p(i, j) = p(i, j-1) * (1-p) + p(i-1, j) * p$$

Now, instead of actually using recursion in our code to figure this out, we can convert our algorithm into dynamic programming by creating a two-dimensional array that stores the answers to all necessary recursive calls. Here's some (uncompiled) Java code to solve the problem:

```
public static int WS(int i, int j, double p) {

  double[][] prob = new double[i+1][j+1];

  prob[0][0] = 1;

  // Fill in probabilities for team A sweeping.
  for (int Awin=1; Awin<=i; Awin++)
    prob[Awin][0] = p*prob[Awin-1][0];

  // Probabilities for team B sweeping
  for (int Bwin=1; Bwin<=j; Bwin++)
    prob[0][Bwin] = (1-p)*prob[0][Bwin-1];

  // Here's where we fill in the table, using the recursive
  // formula.
  for (int Awin=1; Awin<=i; Awin++)
    for (int Bwin=1; Bwin<=j; Bwin++)
      prob[Awin][Bwin] = p*prob[Awin-1][Bwin] +
                         (1-p)*prob[Awin][Bwin-1];

  // Here's our answer.
  return prob[Awin][Bwin];
}
```

Typically, this one's not too fun to trace through by hand...