My search agent calculates win probabilities by maintaining a count at each node of the total number of leaves in the subtree with that node as the root (which represents total number of possible outcomes) and the number of those leaves that represent a win for the player who is drawing. At any given node, the total number of outcomes can be found by summing the number of outcomes of each of its children. The number of successes for any parent node can be found by summing the successes of the child nodes and subtracting the resulting figure from the number of outcomes for the child node. This essentially yields the total number of losing outcomes for the child nodes which is in turn the number of winning outcomes for the parent node.

The work is done by the win\_prob() function. It takes in an integer parameter num\_tokens, which is the number of tiles on the board, and an array of pairs of doubles which will store the probabilities for each num\_tokens once they are solved for. The second element in each pair is the number of leaves (outcomes) in the subtree whose root has num\_tokens and the first element is the number of those that are successes. Doubles are used to store the integer values because some of the numbers generated are too large to be stored by any integer data type. The solutions array has one extra element so that I can use indexes 1 – 500 to correspond to the num\_tokens directly. Index 0 is unused but seems like an acceptable memory cost to achieve the convenience mentioned above.

The win\_prob() function first checks the solutions array to see if the probability for current num\_tokens has already been solved (only way 2nd element of a given pair is non-zero is if it’s been solved so that’s what it checks for). If num\_tokens has not been solved, it checks for the base case which is num\_tokens == 1. If this is true there is only one possible outcome and it is a lose so the probability is 0/1 and this will be saved to the solutions array at index 1 and returned.

If num\_tokens has not been solved and is not the base case the children of num\_tokens will be generated, starting with the child generated by removing the largest allowable number of tokens. I start with the largest removal first so as to reach a solution as fast as possible. The more solutions that are stored in the array, the less time will be spent on each successive calculation (due to table lookup short circuiting the recursion) so getting a solution as soon as possible is desirable. Upon generation of each child, it is checked to see whether the win probability of the child is the worst of the children created so far. If it is, the probability is stored as is the number of tokens taken to achieve it. Regardless of whether the probability is the worst, the total outcomes and losing outcomes (total outcomes – winning outcomes; aka winning outcomes of parent) are added to the working probability of the parent.

Once all children of the node are generated and summed as above, the result is saved to the solutions array and the information is printed out. The value that was saved to the solutions array is then returned.