Code Jam 2008 - APAC Semifinal

Analysis: Apocalypse Soon

This problem is a sheep in wolf's clothing. However, it must have been very convincing wolf's clothing! Fewer competitors solved it than any other problem, despite the simplicity of the actual solution. Since this was an onsite round, they were all experienced competitors; in some sense, that worked against them.

Why does the problem seem so hard? To a practiced eye, it has all the trappings of a horrible exponential search. You have 5 actions to choose from on each day, and the results of those actions don't tend to overlap, leading to an exponential number of possible states. Furthermore, even the subtlest change in a nation's strength can drastically affect the final outcome, so at each step the entire state of the world matters. This makes a Dynamic Programming-based solution look unlikely. And due to the chaotic nature of the rules, there's unlikely to be a greedy strategy that works in all cases.

So a horrible exponential search seems essential. On a 50x50 grid of numbers between 0 and 1000, this is clearly intractable! Give up and go home. No, wait... it's supposed to be solvable somehow, right? How could that be?

Well, if you try a few cases by hand, you'll soon realize why: everyone ends up killing their neighbors really, really quickly. A random map tends to stabilize (no living neighbors) in only 4-6 days. It's actually very difficult to come up with a case that takes longer. The more you try, the more you'll realize just how difficult it is; the army sizes required grow exponentially, and the tighter you pack them the sooner they all die. The map size doesn't turn out to matter much - it's the army size bound of 1000 that really sets the cap on how long you'll need to search.

What exactly is this cap? That's a really tough question. For instance, if army sizes are unbounded, there *is* no small limit. Here's a simple (2*n*-2)x2 case that takes *n* days to stabilize:
1 2 4 8 ... 2^{2n-3}
1 2 4 8 ... 2^{2n-3}

However, the army sizes *must* grow exponentially relative to the number of days. Here's a simple proof: on day *n*, let S_n be the strength of the strongest nation that still has living neighbors. Let *N* be any nation with strength ≥ $S_n/2$. Suppose that after 8 days *N* still has strength ≥ $S_n/2$. Then it must have killed all its neighbors, since none had strength ≥ S_n (that is, any neighbor would die after at most 2 attacks). So every such *N* either ends up isolated or weaker than $S_n/2$ after 8 days. Thus, $S_{n+8} \le S_n/2$. This puts a strict bound of $8(\log_2 S_n+1)$ more days before all nations are dead or isolated.

Unfortunately, for army sizes of 1000 this bound of 80 days still isn't very helpful. The bound can be improved with tighter reasoning, but due to the compact, constrained nature of the grid, the "true" bound is probably much smaller - around 9-10 days! However, we don't know how to prove a bound anywhere near this. If you can think of a way, let us know! :)

The worst test case in our data had an answer of "9 day(s)". So as it turns out, even a straight brute force solution (on the worst-case order of $50x50x5^9$) would suffice to solve it. Adding simple pruning heuristics and memoization can potentially also help, but isn't necessary.

Incidentally, one potential coding error you could make is to assume that effects can only propagate at one grid square per day. The "speed of light" (as it's known in cellular automata) is actually *two* grid squares per day, because if an army's neighbor changes, it may choose to attack its opposite neighbor instead. In the "9 day(s)" test case, there are nations 13 squares

away from yours that end up affecting the answer. So if you tried to speed up your solution by considering only the local region around your nation, you'd have to be careful not to make it *too* local.

In the end, all it took to get this problem was bravery (or foolishness). If you code it properly and try it, it works! It's just a question of how well you can convince yourself that the simple solution has a hope of working. You certainly wouldn't want to download the Large dataset and *then* realize your solution was too slow...