Code Jam 2012 - Round 3

Analysis: Perfect Game

Preamble

It will come as no great surprise that this problem was inspired by a video game that the author played: Super Meat Boy. Turning the real achievement challenge into a Code Jam problem involved a little simplifying: it doesn't take the same amount of time to lose a level as to win it, for example.

Total Expected Time and Number of Attempts

Your total expected time to complete the achievement will be the expected number of attempts, times the expected time per attempt.

First, let's figure out how many attempts it's going to take us to complete the achievement. The probability of success in any given attempt is easy to compute, since it's simply the probability of succeeding on every level once without failing any of them: $(1-P_1)*(1-P_2)*...*(1-P_N)$. The expected number of attempts won't depend on anything other than this probability, and this probability doesn't depend on the order of the levels; so the expected number of attempts doesn't depend on the order of the levels, and we can ignore it when figuring out what order to attempt them in.

In case you're curious about what the expected number of attempts is anyway, read on. Let's say that we're going to try to do something that might fail, like complete this achievement, and that we're going to keep trying until we succeed. If the probability of success is P, then what we're doing is called a Bernoulli Trial, repeated until we achieve success. The number of attempts that will take falls into a random distribution called the Geometric Distribution. The expected value of this distribution–the number of attempts it will take–is $1/P$.

Knowing about the Geometric Distribution lets us compute the expected number of attempts easily. The chance that we'll complete every level successfully, one after another, is the product of the probabilities of success, which we'll call P; the number of attempts it will take us to complete that is $1/P$. As promised, this number doesn't depend on the order in which we attempt the levels, and since what we're trying to do is compute the best order, we're going to ignore it.

Time Per Attempt

Let's choose an arbitrary order for the levels. Suppose the i^{th} of those levels takes t_i time, and you die in that level with probability p_i. In any attempt, you'll definitely reach level 1; you'll reach level 2 with probability $1-p_1$; you'll reach level 3 with probability $(1-p_1)*(1-p_2)$; and so on.

Based on those calculations, the amount of time it takes you to make one attempt will be $\text{expected time} = t_1 + (1-p_1)t_2 + (1-p_1)(1-p_2)t_3 + \ldots$. This is because you will try level i only if you pass the first $i-1$ levels.

Now, let's consider what would happen to that expected time if we swapped levels i and $i+1$. Only two of the terms of the expected time equation would be affected—the others would simply reverse the order of $(1-p_i)$ and $(1-p_{i+1})$, which doesn't matter. Those two terms themselves have several multiplicative terms in common:

Pre-swap:
$(1-p_1)(1-p_2)...(1-p_{i-1})t_i +$
$(1-p_1)(1-p_2)...(1-p_{i-1})(1-p_i)t_{i+1}$.
Post-swap:
$(1-p_1)(1-p_2)...(1-p_{i-1})t_{i+1} +$
$(1-p_1)(1-p_2)...(1-p_{i-1})(1-p_{i+1})t_i$.

So we're better off post-swap iff:
$t_i + (1-p_i)t_{i+1} > t_{i+1} + (1-p_{i+1})t_i$
$t_i p_{i+1} > t_{i+1} p_i$

Now we can compare two adjacent levels to see whether they should be swapped. Doing this greedily results in a stable sort of the levels. With a little more work, you can prove that non-adjacent levels should be swapped under the same conditions, and that the stable sort is thus optimal.

Implementation Details

Our last inequality above said we should swap iff $t_i p_{i+1} > t_{i+1} p_i$. It's tempting to go one step further, to $t_i/p_i > t_{i+1}/p_{i+1}$, but we can't: p_i or p_{i+1} could be zero, and dividing by zero isn't a mathematically valid thing to do.

Another reason not to make that final step–though the final step actually kind of works if you don't mind special-casing $p_i=0$ to give an infinite score–is because division almost inevitably makes us deal with floating-point numbers. In a problem like this where we're trying to make a decision based on numbers, we want to make sure we're doing exact comparisons. We wouldn't want to reverse two equivalent levels because $t_i/p_i = 10.0000000001$ and $t_{i+1}/p_{i+1} = 10.0$.