Should I attack in Risk? (The Board Game)

tl;dr

I played around a bit trying to calculate what can happen when a country attacks another one in risk. It turned out to be a fun problem to think about, so I did a bit of an overkill. Scroll down to see a few exemplary results!

Motivation and Rules

The Risk games in my circle of friends rarely came to an end. Furious negotiations, out-of-game brawls over broken contracts, but also endless dice rolls in epic 30 vs. 20 troops battles often kept the game going until deep into the night. Most of the time we would break off, document the score in order to play it out on another day and then never finish the game. The dice rolling in particular is a big problem, because we play with what we call "Die Michelsonsche Verzögerungstaktik". This is just like the official one, except that only one troop can be lost per dice roll:

The attacker rolls

- 1 dice with 1 attacking troop (if there are x troops on a land, he can attack with x-1)
- 2 dice with 2 attacking troops
- 3 dice with 3 or more attacking troops

The defender rolls

- 1 dice with 1 defending troop
- 2 dice with 2 or more defending troops

After attacker and defender rolled their dice, the highest of each is compared. If the defender's highest dice is equal to or higher than the attacker's highest dice, the attacker looses a troop. Otherwise the defender looses one.

When 30 attackers fight against 20 defenders, this really takes some time. So naturally the question arises: Can this be calculated? Of course!

Single Probabilities

The possible outcomes of a single dice roll can be calculated by simple counting of possible outcomes. In total there are 6 possible scenarios for (1v1, 1v2, 2v1, 2v2, 3v1 or 3v2 dice). With a couple of for-loops, this is quickly done, e.g. for the case 2 attackers vs 1 defender:

```
wins_for_attacker_2v1 = 0
for a1 in range(1, 7):
    for a2 in range(1, 7):
        for d1 in range(1, 7):
            if d1 < max(a1, a2):
                wins_for_attacker_2v1 += 1
p_2v1 = wins_for_attacker_2v1 / 6**3
```

The probabilities for the attacker to win a single dice roll are:

attackers \ defenders	2 or more	1
3 or more	47%	66%
2	39%	58%
1	25%	42%

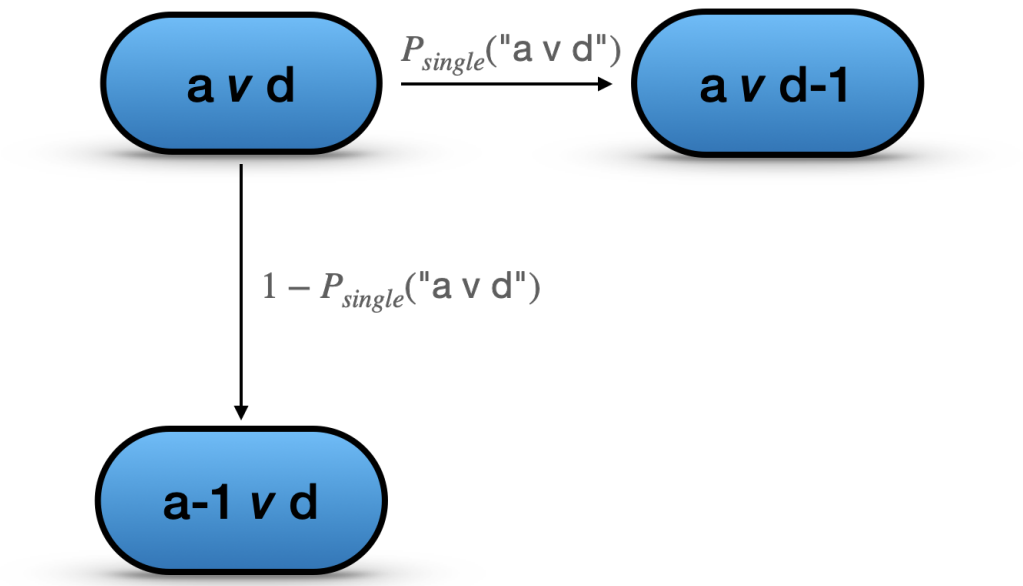
Not too impressive for the attacker if you ask me!

Complete Attacks

Now that we have the single probabilities, the next step is to think about an entire attack: What is the probability to win with **a** attackers against **d** defenders?

Top Down

The first idea that comes to mind is recursion:
To lead from **a** vs **d** to either victory (? vs 0) or defeat (0 vs ?), all possible paths have to be calculated, weighed with their individual probability and summed up. But from a local perspective, this simplifies as there are really only two possibilities: You either win a single roll or you loose one:



So the relation is obvious:

\[P_{total}(\text{"a vs d"}) = P_{single}(\text{"a vs d"}) * P_{total}(\text{"a vs d-1"}) + (1 - P_{single}(\text{"a vs d"})) * P_{total}(\text{"a-1 vs d"}) \]

The base cases are where no attacker or defender is left:

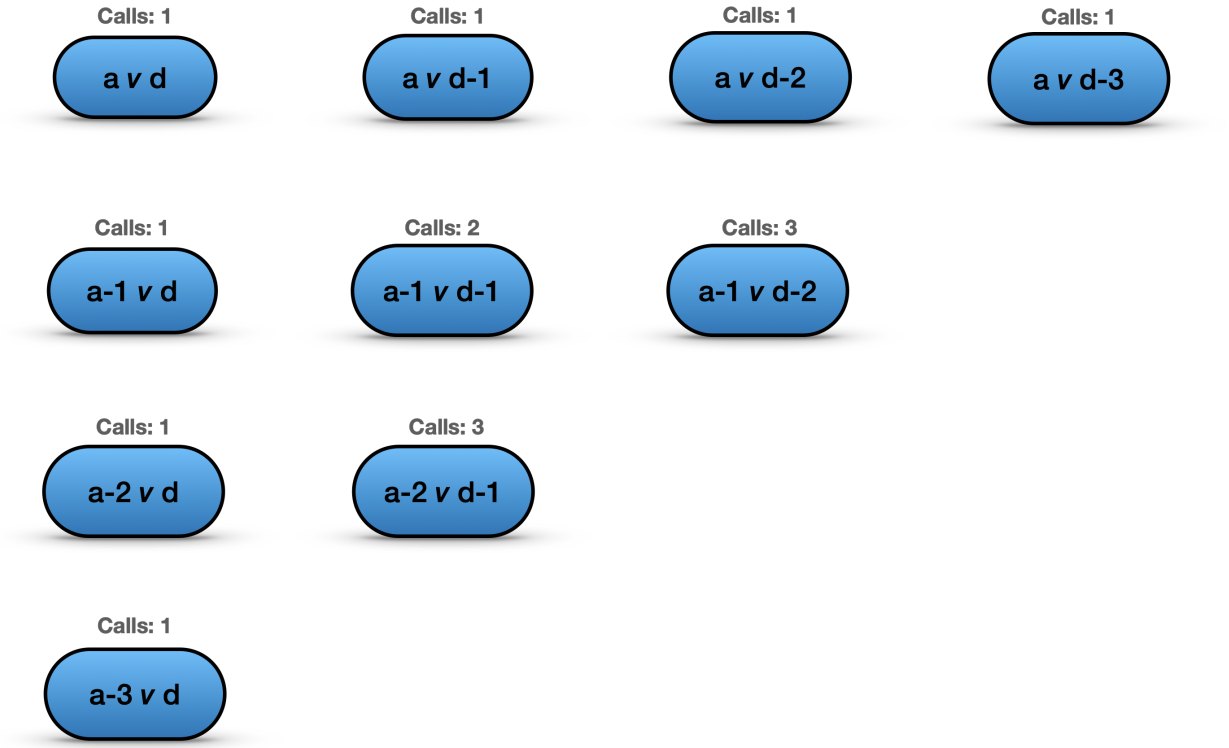
\[\begin{aligned} P_{total}(\text{a vs 0}) &= 1 \; \text{ and}\\ P_{total}(\text{0 vs d}) &= 0 \; \\ \forall{a, d \geq 1} \end{aligned} \]

In code, this is even easier to read:

```
def p_recursive(self, a, d):
    """Returns the probability to win an attack with a attackers against d defenders.
    """
    if a == 0:
        return 0
    elif d == 0:
        return 1
    else:
        return self.basic_probs.p_swin(a, d) * self.p_recursive(a, d - 1) + \
            (1 - self.basic_probs.p_swin(a, d)) * self.p_recursive(a - 1, d)
```

For small attacks, this works just fine. For example, $P_{total}(\text{10 vs 10}) = 36.9\%$ (which is surprisingly low). But when the numbers increase, e.g. with 15 vs 15 my laptop already gives up. But why?

It turns out, by design this 2d recursion scales pretty catastrophically. The first node gets called only once, but subsequent ones get called increasingly often:



Notice that each number of calls is of course just the sum of the number of calls for the node above and the one on the left of the current one. The ones on the outside can only get called once each, as they only have one left or upper neighbor.

Pascal's triangle appeared out of nowhere!

So how many possible paths are there to get from the starting field to another one, assuming that they have a difference of **a** attackers and **d** defenders?

$$ M(a, d) = \frac{(a + d)! }{a! d!} $$

This can be shown with induction, using the previous equation as Induction Hypothesis (IH):

The start with (a, d) = (0, 1) or (1, 0) is trivial. Now going from (a, d) (a+1, d):

$$ \begin{equation} \begin{aligned} M(a+1, d) & = M(a, d) + M(a+1, d-1) \\ & \overset{\mathrm{IH}}{=} \frac{(a + d)! }{a! d!} + \frac{(a+1 + d/1)! }{(a+1)! (d-1)!} \\ & = ... \\ & = \frac{(a+1 + d)! }{(a+1)! d!} \end{aligned} \end{equation} $$

This is symmetric w.r.t exchanging a and d, since the critical step in line 2 applies equally (any field can be reached from the fields "left" and "above" of it).

When fighting with 30 against 20, the field (1, 1) alone is called $\frac{(29 + 19)!}{29! * 19!} \approx 10^{13}$ times. No wonder my laptop gave up!

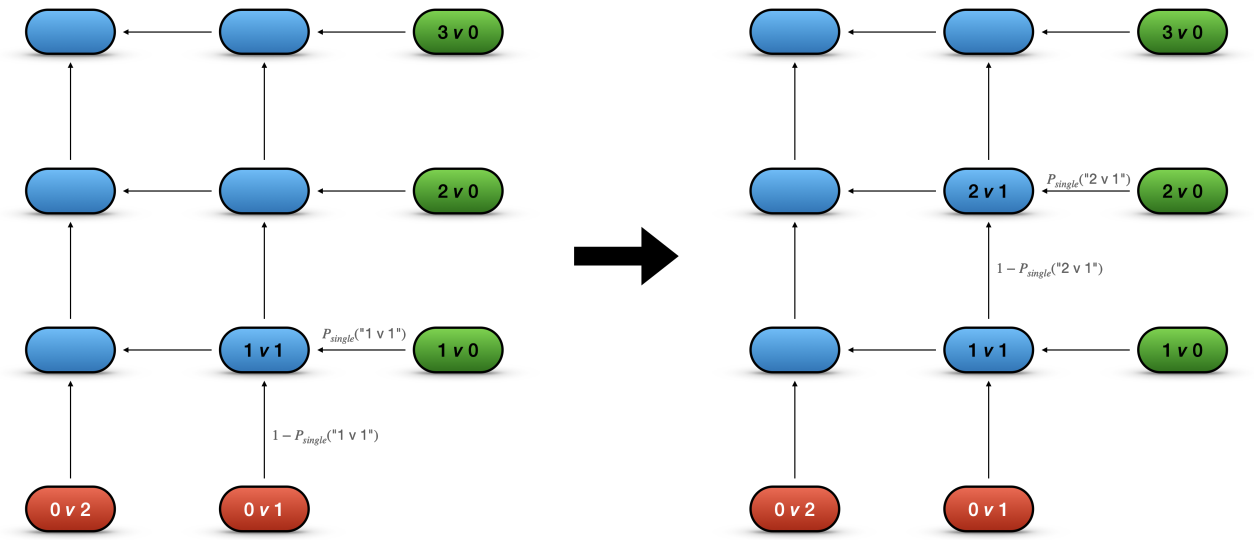
Storing the results of the function for any given set of input parameters, e.g. with a decorator like `functiontools.lrc_cache` doesn't really help:
The calculation may only be done once per node, but the calculation part was never really the problem. The real problem of getting called just way too often still remains.

One solution I found is to change the conceptual view of going top down towards starting at the possible outcomes and working one's way up towards the initial starting point:

Bottom Up

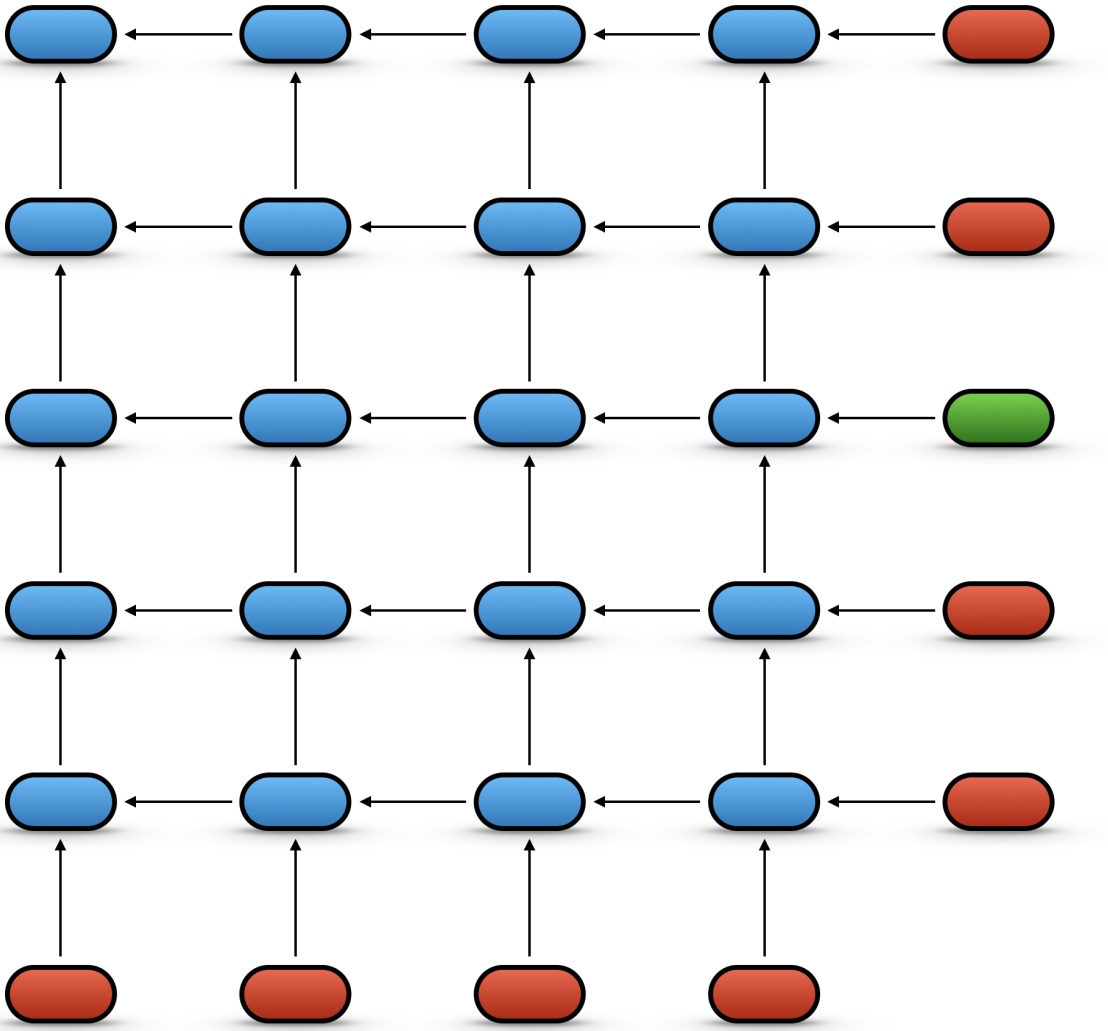
The basic idea is to work from the bottom up. Given the probabilities in the fields under and on the right of any field, the probability to win starting from there can be calculated with exactly the same relation as before. The difference is just to start at the bottom. There the probabilities are already known, which sets up the boundary conditions:

- Wherever a=0, no attackers are left and P=0 (visualized in red, for unsuccessful attack)
- Wherever b=0, no defenders are left and P=1 (visualized in green)



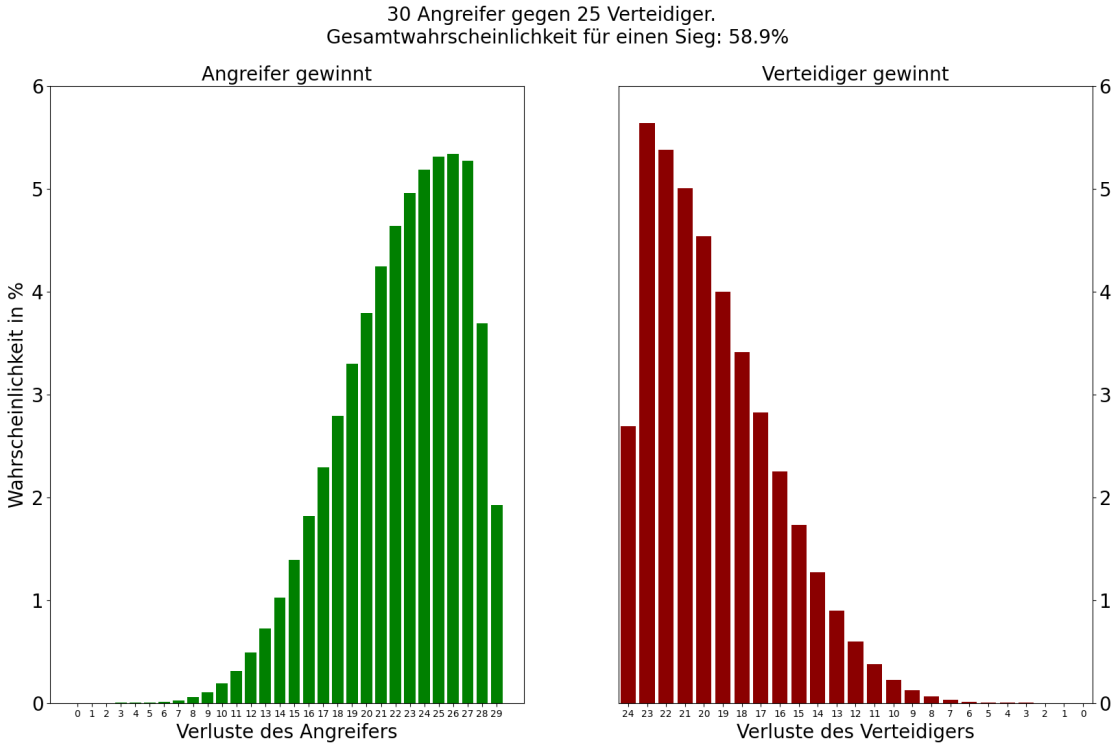
And so on: Now it can just be iterated from the bottom right going up the columns right to left or the rows bottom to top. All **a** * **d** fields must be filled up, but this time only once! Using this, values up to ~1000 can easily be reached: \(P_{total}(\text{1000 vs 1000}) = 0.05\% \) (for such large numbers the 47% of a single 3v2 become very small)

The idea of boundary conditions is also convenient because they now can easily be changed. For example to answer questions like "How probable is it to win with the attacker having exactly 3 remaining troops?", this would just mean changing all outcomes to red (or P=0), except for the one where **a=3**:



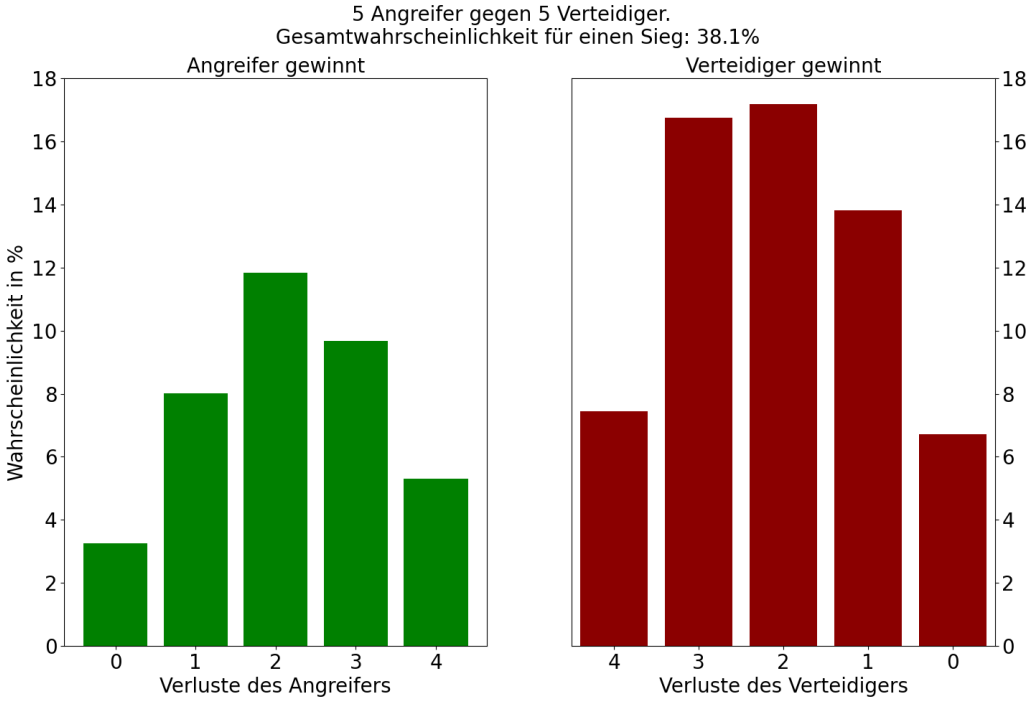
Results

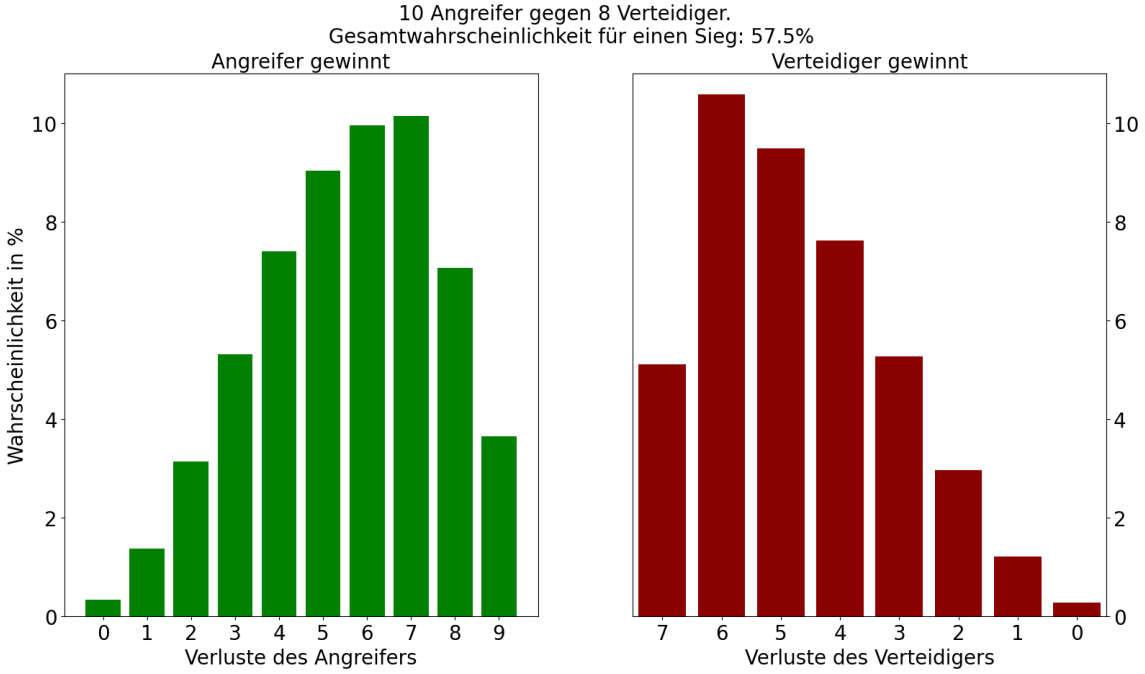
Using this, here are some examples of attacking scenarios and the probabilities of each of their possible outcomes:



It is of course almost a binomial distribution, but with dips where the attacker or defender runs into cases where the number of dice that can be used gets reduced.

For smaller troupe sizes, what strikes me is that the curves are pretty flat. This reflects what we all know: Attacks are often just wildcards and everything can happen.


5 Angreifer gegen 5 Verteidiger.
Gesamtwahrscheinlichkeit für einen Sieg: 38.1%
Angreifer gewinnt
Verteidiger gewinnt
Wahrscheinlichkeit in %
Verluste des Angreifers
Verluste des Verteidigers


10 Angreifer gegen 8 Verteidiger.
Gesamtwahrscheinlichkeit für einen Sieg: 57.5%
Angreifer gewinnt
Verteidiger gewinnt
Wahrscheinlichkeit in %
Verluste des Angreifers
Verluste des Verteidigers

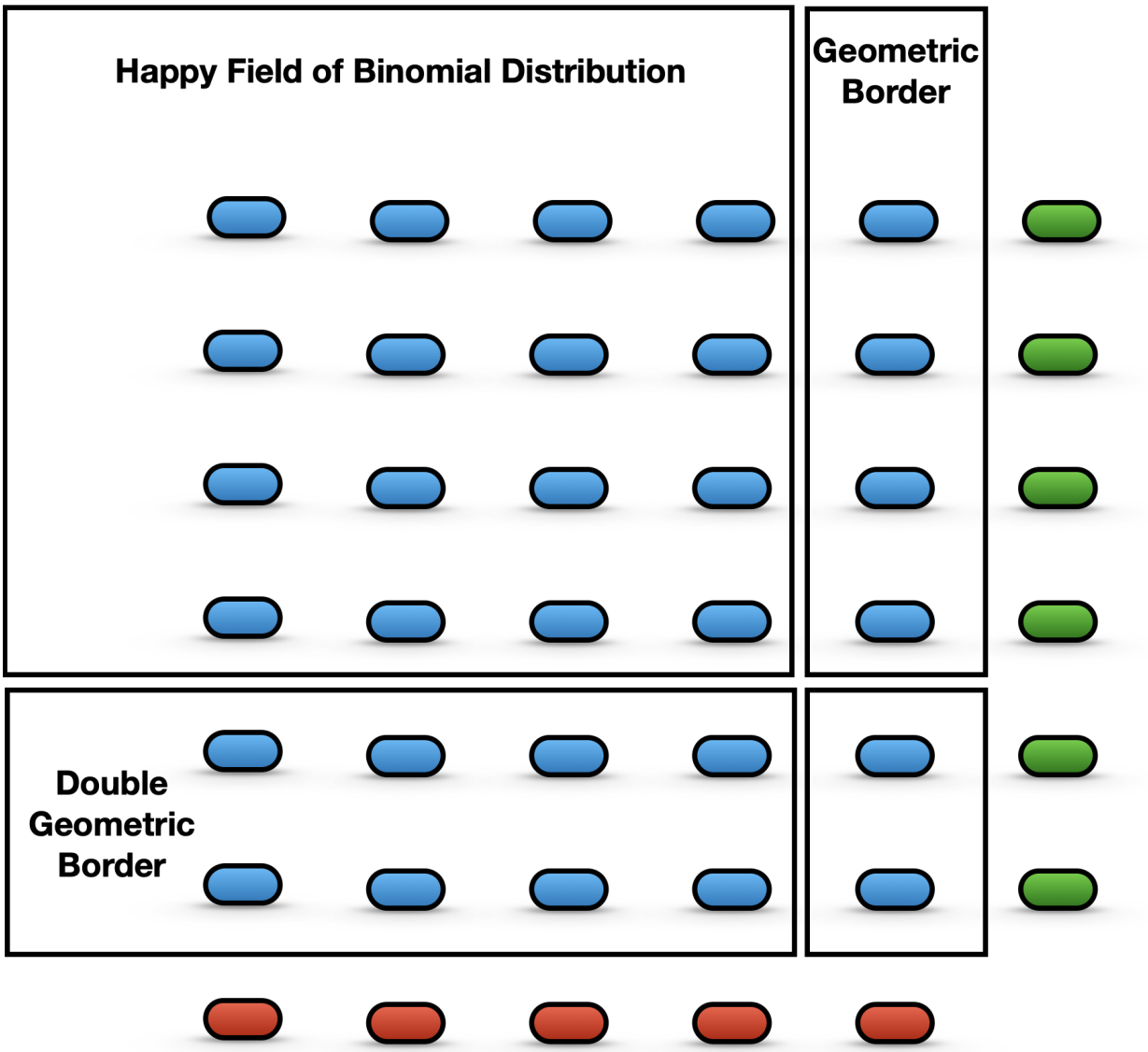
Conclusion

It is surprising how much time one can dump into something like this. But at least now we have the the possibility to simulate the outcome of an attack within seconds instead of fifteen minutes. So for the first time, we started actually finishing games! Now that extinguishing a player happens within a second instead of in a 15 minutes crushing faceoff, the threshold of doing so significantly reduced. Funnily enough a new threat came up in the later stages of the game: "If you attack me, I will not simulate", which means the whole table will have to wait for quite some time until they are done...

Outlook

Now that I've got a way to simulate 1000 vs 1000 one might say that's enough. This is of course **not** enough!

To reach the really high numbers, an analytical approach would be necessary. It might work like this:



Within the Happy Field of Binomial Distribution, the probability of winning a single attack stays constant as the dice numbers don't change. So the probability to "go from" point A to point B can instantly be calculated using the binomial distribution. Then one would sum up all paths from the initial point to the border and weigh it the the probability on the respective field on the border. For the right border ("Geometric Border"), this would come down to something like the geometric series, so it could be written down for each point. For the lower border, maybe some sort of "double" geometric series will come out... In total this seems pretty doable to me!