Investigating the relationship between the number of troops and the chances of winning in Risk.

Contents

Research question

How does the initial number of attacking players a $(0 < a \leq 40)$ and defending players d $(0 < d \leq 40)$, where $a = d$, affect the probability for the attacker to win?

Background

Playing board games is my favorite form of entertainment. I always enjoyed playing with my entire family. However, since my parents were not as keen, it was often just me and my brother. To compensate for this, my parents bought us several two-player games. One of these games was 'Risk'. Despite being notorious for its duration, it soon became a favorite, as I would rush home from school to pick up the game where me and my brother left off previously.

The intensity of the game would peak, when I started to accumulate troops in Russia, and he, in the Urals. I would try to conquer his Asia, while he would try to conquer my Europe. Often, we would skip our opportunities to attack, just to place more troops in these territories. This would climax in an all-out battle that would decide the course of the rest of the game. We would often treat this battle more dramatically than it needed to be, with both of us role-playing generals sending troops out to die or reenacting a significant battle in history such as Stalingrad in WW2.

[“Risk Game Rules.”]

Risk serves as a great opportunity to spend time with my brother and is hence a deeply personal board game for me. Hence, I wanted to investigate how the number of attacking and defending troops affects the probability of winning, and the expected number of attacking troops remaining.

Rules

The goal of Risk is to conquer all 42 territories on the map. Both players receive 40 troops, and 14 random territories, across which the player’s troops can be distributed. If only two players are playing, the remaining 14 territories are given to a third player, that cannot attack, but only defend. The game is played turn by turn.

In a turn, a player can do three actions in the following order:

Receive troops: For every three territories conquered, the player receives 1 troop at the start of each turn, rounded down. This means if the player has 15 territories, they will receive 5 troops at the start of their turn. Additionally, if all the territories in a continent belong to one player, they receive bonus points depending on the continent. For instance, if a player controls the entire of the continent Australia, they receive two bonus troops.

Attack: a player can attack any territory that does not belong to them and borders a territory that does belong to them. They can attack as many territories as they wish, provided that the territory they are attacking borders the territory they use to attack. To attack, a territory must have more than 1 troop in it. The player can send out up to three troops to attack, and the defender can send up to two troops to defend. It should be noted that the attacker must always leave 1 troop behind, while the defender may use all their troops. The attacker rolls one dice for each troop sent, and so does the defender. Depending on the values of the dice rolled (explained further in the *Dice* section), three outcomes are possible: the attacker loses 2 troops, the defender loses 2 troops, both lose 1 troop. The attacker can choose to send out another wave of troops. The attacker must stop attacking the territory in two cases: the attacker has only 1 troop remaining, the attacker has defeated all troops in the other territory and has hence conquered it. In the second case, all surviving troops of the last wave must be placed in the newly conquered territory. This means if the defending territory has 2 troops remaining, and the attacker attacks with 3 troops, and the defending territory loses both troops, the attacker must place all three troops in the new territory.

Move troops: a player can move any number of troops from one territory to another, provided that: the territories share a border, at least one troop remains in both territories. Unlike in attacking, a player can only move the troops from one territory to another, and not repeat this with another pair of territories elsewhere on the map.

For the purposes of this investigation, only the 'Attack' step will be considered. This investigation will consider *40* to be the maximum number of troops given as infantry on either side, as the total number of infantry troops the game comes with per player is *40*.

Assumptions

There are two assumptions made for this investigation:

- The attacker and defender always send out the maximum number of troops possible in each turn
- Once an attacker starts attacking, they will continue until they have lost or won the battle

Dice

In each round of a battle, the attacker and defender roll one dice for each troop they send. This means the attacker and defender can role a maximum of 3 and 2 dice per round respectively. To calculate the number of loses on either side, the attacker's and defender's greatest dice are paired. If the attacker's dice is strictly larger than defender's, the defender will lose their troop, else, the attacker will. Then, the next two largest dice are paired up, and the same method is used.

For instance, if the attacker rolls $[6, 3, 2]$, and the defender rolls $[6, 4]$, the attacker will lose two dice as the largest and second largest attacking dice are not greater than the largest and second largest defending dice respectively. If the attacker rolls $[6, 3, 2]$, and the defender rolls $[5, 5]$, the attacker and defender will lose one each, as $6 > 5$ and $5 \leq 5$.

The following notation will be used:

- Let *W* = win, *T* = tie, *L* = loss
- *XvY* refers to the number of attackers (*X*) and defenders (*Y*) in the battle.

There are six cases for rolling the dice:

Table 1

Attacker (no. dice rolled)	Defender (no. dice rolled)	Possible outcomes
3	2	W, L, T
2	2	W, L, T
1	2	W, L
3	1	W, L
2	1	W, L
1	1	W, L

The probabilities of each of the possible outcomes for each case must be calculated. However, I recognized that I only needed to calculate the *W* and *L* cases for *3v2* and *2v2,* as *T* would be complementary to the sum. Likewise, for the other four cases, only *W* would need to be computed to find *L*.

I used two methods to calculate these values.

Method 1: counting the combinations

Realizing that counting could be tedious, especially for the *3v2* case as there would be 6^5 or 7776 combinations, I needed to develop an organized method to count the combinations.

For the rounds of *1v1*, *2v1*, *3v1*¸ and *1v2*, i.e. the rounds where a tie is not possible, the following method was used:

1. Use the smaller of the number of attackers or defenders to organize the combinations. This means for *2v1* and *3v1*, the number of defenders will be used to organize. This allowed me to create the shortest list possible (as if I ordered by the number of attackers for a *3v1*, I would have created a list of 216 combinations compared to just 6).
   a. If ordering by attackers, list the dice values in ascending order, else, in descending order. This is done as, when the combinations are organized based on the attacking value, as the number attacking value increases the number of combinations of when the defender's dice is rolled for the attacker to win also increases. For instance, for a *1v2* round, when a *1* is rolled by the attacker, there are *0* combinations of defender and attacking values for which the attacker will win. However, when a *2* is rolled by the attacker, there is one combination i.e. when the defender rolls a *1*. A higher dice value of the organizing dice includes all combinations of lower values i.e. it is cumulative. Hence, it would be redundant to count the combinations in the reverse order.
   b. If ordering by defenders, do not list *6*, as if a *6* is rolled by the defender, there are zero combinations in which the attacker can win, as the attacker cannot roll a number greater than *6*.

2. Manually count the number of winning combinations for each value of the attacking/defending dice for the first three values.
3. Find a pattern between the number of combinations and use it to find the remaining combinations.

For a *1v1* round, the defending dice cannot be *6* if the attacker is to win. If the defender rolls a 5, there is one combination in which the attacker wins: the attacker rolls a 6. If the defender rolls a 4, there are two combinations. Therefore, the total probability of winning can be calculated as follows:

$$P(W|1v1) = \frac{5+4+3+2+1}{6^2} = \frac{15}{36} \approx 0.417$$

Hence, $P(L|1v1) = 1 - \frac{15}{36} = \frac{21}{36} \approx 0.583$.

For a *1v2* round, the combinations are shown in the table below

Table 2	
Attacker	No. Defender combinations (for attacker to win)
1	0
2	1
3	4
4	9
5	16
6	25

Hence, $P(W|1v2) = \frac{0+1+4+9+16+25}{6^3} = \frac{55}{216} \approx 0.255$

Here, the pattern is $c = (a-1)^2$, which became self-evident after I manually counted up to a = *4*. *c* refers to the number of combinations, and *a* refers to the value of the attacking dice.

For *2v1*,

Table 3	
Defender	No. Attacker combinations
5	11
4	20
3	27
2	32
1	35

$$P(W|2v1) = \frac{35 + 32 + 27 + 20 + 11}{6^3} = \frac{125}{216} \approx 0.579$$

Here, for each decrease in the value of the defender dice, the number of combinations would increase by an odd number that would decrease by two in each subsequent defender value with an initial increase of *9* i.e. *9*, *7*, *5*…

For *3v1*,

Table 4	
Defender	No. Attacker combinations
5	91
4	152
3	189
2	208
1	215

$$P(W|3v1) = \frac{215 + 208 + 189 + 152 + 91}{6^3} = \frac{855}{1296} \approx 0.660$$

Here, for each decrease in the value of the defender dice, the number of combinations would increase by *61* initially. The increase would decrease by multiples of *6* in each subsequent defender value with an initial increase of *9* i.e. 61, 37 (61 – 37 = 24), 19 (37 – 19 = 18), 7 (19 – 7 = 12)…

To count the combinations for a *2v2* or *3v2* battle, I came up with a different counting system. The figure below summarizes my method of counting for a *3v2* battle:

For *2v2*,

2 attackers vs 2 defenders (Table 5)		
Defender	Combinations	
	Attacker loses 2	Attacker wins 2
6, 6	36	0
6, 5 (2)	35	0
6, 4 (2)	32	0
6, 3 (2)	27	0
6, 2 (2)	20	0
6, 1 (2)	11	0
5, 5	25	1
5, 4 (2)	24	3
5, 3 (2)	21	5
5, 2 (2)	16	7
5, 1 (2)	9	9
4, 4	16	4
4, 3 (2)	15	8
4, 2 (2)	12	12
4, 1 (2)	7	16
3, 3	9	9
3, 2 (2)	8	15
3, 1 (2)	5	21
2, 2	4	16
2, 1 (2)	3	24
1, 1	1	25
Total	581	295

The number in parenthesis accounts for the permutation of defenders. For instance, the dice combinations *2*, *1* and *1*, *2* have the same number of attacking combinations, but are distinct. Therefore, *2*, *1* will need to be multiplied by *2* to account for *1, 2* not being in the table.

$P(L|2v2) = \frac{581}{6^4} = \frac{581}{1296} \approx 0.448$ and $P(W|2v2) = \frac{295}{1296} \approx 0.228$.

Hence, $P(T|2v2) = 1 - \left(\frac{581}{1296} + \frac{295}{1296}\right) = \frac{420}{1296} \approx 0.324$.

For *3v2*,

3 attackers vs 2 defenders (Table 6)		
Defender	Combinations	
	Attacker wins 2	Attacker loses 2
6, 6	0	216
6, 5 (2)	0	200
6, 4 (2)	0	160
6, 3 (2)	0	108
6, 2 (2)	0	56
6, 1 (2)	0	16
5, 5	16	125
5, 4 (2)	43	112
5, 3 (2)	64	81
5, 2 (2)	79	44
5, 1 (2)	88	13
4, 4	56	64
4, 3 (2)	98	54
4, 2 (2)	128	32
4, 1 (2)	146	10
3, 3	108	27
3, 2 (2)	153	20
3, 1 (2)	180	7
2, 2	160	8
2, 1 (2)	196	4
1, 1	200	1
Total	2890	2275

$P(W|3v2) = \frac{2890}{6^5} = \frac{2890}{7776} \approx 0.372$ and $P(L|3v2) = \frac{2275}{7776} \approx 0.293$.

Hence, $P(T|2v2) = 1 - \left(\frac{2890}{776} + \frac{2275}{7776}\right) = \frac{2611}{7776} \approx 0.336$.

Reflection:

When initially attempting to calculate the probabilities enumerated previously, I wanted to use a smarter method to find the probabilities. Counting was my last option. Unfortunately, I was unable to come up with a faster method, so I ended up resorting to counting. This was due to the fact that, especially *3v2* and *2v2* rounds, I could not recognize any patterns in how the number of combinations increased with the defender values. The results were the same, however, as an avid programmer who strives to solve problems as efficiently as possible, I was not satisfied with this brute-force approach.

Method 2: simulation

As another approach, I coded up a program (Appendix A) that would roll *a* attacking dice and *d* defending dice, for various sample sizes *n* and calculated the ratio of wins, losses, and ties. The results are shown in the table below:

Table 7

Trials (n)	3v2			2v2			1v2		3v1		2v1		1v1	
	W	L	T	W	L	T	W	L	W	L	W	L	W	L
10	0.5	0.2	0.3	0.1	0.4	0.5	0.3	0.7	0.5	0.5	0.6	0.4	0.5	0.5
100	0.41	0.27	0.32	0.21	0.42	0.37	0.30	0.70	0.62	0.38	0.59	0.41	0.43	0.57
1000	0.378	0.268	0.354	0.200	0.456	0.344	0.240	0.760	0.632	0.368	0.570	0.430	0.382	0.618
10000	0.371	0.293	0.336	0.227	0.447	0.326	0.245	0.754	0.662	0.338	0.579	0.421	0.410	0.590
100000	0.371	0.292	0.336	0.226	0.449	0.325	0.256	0.744	0.658	0.342	0.580	0.420	0.415	0.585
1000000	0.371	0.293	0.336	0.228	0.448	0.324	0.255	0.745	0.660	0.440	0.580	0.420	0.416	0.584

The values for $n = 1000000$ are close to the probabilities previously calculated.

Hence, the probability of winning/losing/to tie are shown in Table 8:

		No. Attacking dice (Table 8)		
		1	2	3
No. Defending dice	1	W: 0.417 L: 0.583	W: 0.579 L: 0.421	W: 0.579 L: 0.293
	2	W: 0.255 L: 0.745	W: 0.228 L: 0.448 T: 0.324	W: 0.372 L: 0.293 T: 0.336

This table makes intuitive sense, as if more attacking dice are rolled, the greater the chance there is for rolling a higher number than rolled by the defender. Conversely, if more defending dice are rolled, the greater the chance there is to roll a higher number than of the attacker.

Probabilities of Battles

Accounting for multiple rounds

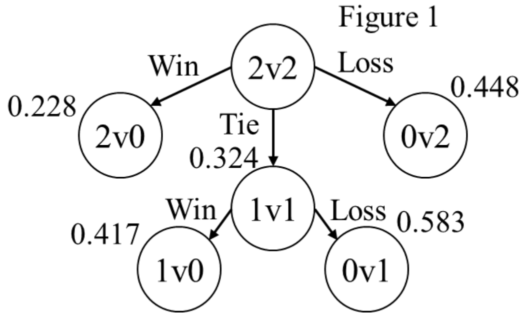
The aforementioned probabilities only apply to one round. Let d_0 be the initial number of defenders, and a_0 be the initial number of attackers – a_0 is one less than the total number of attackers. However, any battle with more than one troop on either side can have more than one round. For instance, let the attacker have *3* troops and the defender have *2*. There are two ways in which the attacker can win (let *W* = win, *T* = tie, *L* = loss).

- *2v2 -> 2v0*: as attacker defeats both troops.
  - Probability = $P(W|a = 2, d = 2) = 0.228$
- *2v2 -> 1v1 -> 1v0*: tie in the first round and winning the second round.
  - Probability = $P(T|a = 2, d = 2) \times P(W|a = 1, d = 1) = 0.324 \times 0.417 = 0.135$

Therefore, the probability of winning a battle of *3v2* is 0.363, which is higher than a single dice roll of *2v2* (0.228).

This battle can be represented as a tree diagram as in Figure 1:

Figure 1



A battle is hence a series of rounds. A round is defined by the number of attackers and defenders in it. It does not matter how the round was arrived at. A *2v1* round is the same irrespective of whether it was arrived at from a *3v2* tie, or a *3v1* loss. Additionally, once a round is arrived at, no succeeding round will be equal to it, as the number of attackers or defenders will only decrease in each following round. The exception to this rule is when either a_0 or d_0 are 0. Here, the battle is over and there will be no more succeeding rounds.

More generally, a battle is effectively a Markov chain. A Markov chain is a chain of states – or rounds – where succeeding states are independent of past states given the current state i.e. $X_{n+1} \perp\!\!\!\perp (X_0, X_1, \ldots, X_{n-2}, X_{n-1}) | X_n$, where X_n represents the current state of the battle, X_0 represents the initial state of the battle, and X_{n+1} represents the next state. This applies to battles as once a round is reached, future rounds only depend on the round, and not how the round was arrived at.

Rounds can be categorized into transient and absorbent rounds. Transient rounds are ones that cannot be visited more than once. Rounds where a_0 and d_0 are greater than 1 are transient as the number of attacker or defenders will only decrease in each successive round. Absorbing rounds are ones that once visited, cannot be left. Absorbing states in battles are achieved when either $a_0 = 0$ or $d_0 = 0$. These states represent a win or loss as the battle cannot proceed from there. If the absorbing state has $a > 0$ and $d = 0$, the battle is won by the attacker. Conversely, if the state has $a = 0$ and $d > 0$, the battle is won by the defender.

Setup

For a_0 and d_0 initial number of usable attacking and defending troops respectively, the total number of possible transient states will be $a_0 \times d_0$. Let the transient states be enumerated in the list A_0 (a, d):

$$A_0 = \{(1,1), (1,2), \ldots, (1, d_0), (2,1), (2,2), \ldots, (2, d_0), \ldots, (3, d_0), \ldots, (a_0, d_0)\}$$

To translate an index i, where i refers to the i^{th} element, to its respective round, I developed two formulas:

$$a = \lceil \frac{i}{d_0} \rceil$$

$$d = \big((i - 1)\ \%\ d_0\big) + 1$$

The formula for *a* works as the value for *a* increases every d_0 indices in the array. If d_0 were *3*, and *i* were *5*, there would be two attackers. However, since $\frac{5}{3} = 1.667$, I rounded up all values for *i* between *4* and *6* (inclusive).

Upon developing the second equation, I found it counter-intuitive here that it did not depend on a_0. This was because since the first used d_0 it made intuitive sense that the second would use a_0. Consequently, I initially incorporated a_0 into the equation. However, I recognized that since it is *d* that increases in each consecutive index (unless *a* increases), I only needed to use *d*. Therefore, my initial equation was $d = i\ \%\ d_0$. However, I recognized that for values where *i* is a multiple of d_0, this would return *0*, rather than returning d_0. For instance, if d_0 were *3*, and *i* were *3*, *d* should equal *3* as well. However, since $3\ \%\ 3 = 0$, this equation was wrong. I solved this problem by subtracting 1 from *i*, and adding *1* after the modulus. While this seemed unnecessary, it allowed values of *i* that are multiples of d_0 to be first reduced by *1*, returning $d_0 - 1$, and adding *1* to that value, returning d_0.

There will be $a_0 + d_0$ absorbing states. They are enumerated in the list A_1:

$$A_1 = \{(0,1), (0,2), \dots, (0, d_0), (1,0), (2,0), \dots, (a_0, 0)\}$$

The first d_0 states represent a loss for the attacker as the number of usable attackers is *0*, while the next a_0 states represent a win as the number of usable defenders is *0*.

The values for *a* and *d* can be calculated given two conditions:

If $i \leq d_0$: $a = 0, d = i$

If $i > d_0$: $d = 0, a = i - d_0$

A₀ and *A₁* can be concatenated to form a $(a_0 \cdot d_0) + (a_0 + d_0)$ length vector *A* with all the possible states, with the first $(a_0 \cdot d_0)$ indices being transient states and the indices from $(a_0 \cdot d_0) + 1$ to $(a_0 \cdot d_0) + (a_0 + d_0)$ being absorbing states.

For example, let $a_0 = 3$ and $d_0 = 3$. The lists would be:

$$A_0 = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)\}$$

$$A_1 = \{(0,1), (0,2), (0,3), (1,0), (2,0), (3,0)\}$$

$$A = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3), (0,1), (0,2), (0,3), (1,0), (2,0), (3,0)\}$$

Reflection:

Finding an algorithm to convert the index to the corresponding number of attackers and defenders is something I believe will be useful when calculating probabilities for larger values of a_0 and d_0. This investigation will attempt to find probabilities up to $a_0, d_0 = 40$. A would be a length of $(40 \cdot 40) + (40 + 40) = 1680$. Hence, writing all the possible combinations manually will be too tedious and impractical. Instead, I plan to code a program that will automate calculating probabilities for large values of a_0 and d_0, which can effectively use these algorithms.

Using matrices

A matrix is a 2-dimensional vector. It is a table with *n* rows and *m* columns, where (i, j) refers to the element in the *i*th row and *j*th column.

In Markov chains, matrices can be used to represent the probabilities of transitioning between states i.e. ‘transition matrices’. Each row in the transition matrix represents a current state and each column represents a state in the next turn. The element in *i*th row and *j*th column represents the probability of the state with index *i* transitioning to the state with index *j*.

For $a_0 = 1$ and $d_0 = 1$, the number of states would be $(a_0 \cdot d_0) + (a_0 + d_0) = (1 \cdot 1) + (1 + 1) = 3$. Therefore, the transition matrix would have three rows and columns and would be as follows:

$$P = \begin{bmatrix} P_{(1,1)\to(1,1)} & P_{(1,1)\to(0,1)} & P_{(1,1)\to(1,0)} \\ P_{(0,1)\to(1,1)} & P_{(0,1)\to(0,1)} & P_{(0,1)\to(1,0)} \\ P_{(1,0)\to(1,1)} & P_{(1,0)\to(0,1)} & P_{(1,0)\to(1,0)} \end{bmatrix}$$

Using the values from Table 8, the probabilities for this matrix can be substituted:

$$P = \begin{bmatrix} 0 & 0.583 & 0.417 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transition matrix for $a_0 = 2$ and $d_0 = 2$ is shown below:

$$P = \begin{bmatrix} P_{(1,1)\to(1,1)} & P_{(1,1)\to(1,2)} & P_{(1,1)\to(2,1)} & P_{(1,1)\to(2,2)} & P_{(1,1)\to(0,1)} & P_{(1,1)\to(0,2)} & P_{(1,1)\to(1,0)} & P_{(1,1)\to(2,0)} \\ P_{(1,2)\to(1,1)} & P_{(1,2)\to(1,2)} & P_{(1,2)\to(2,1)} & P_{(1,2)\to(2,2)} & P_{(1,2)\to(0,1)} & P_{(1,2)\to(0,2)} & P_{(1,2)\to(1,0)} & P_{(1,2)\to(2,0)} \\ P_{(2,1)\to(1,1)} & P_{(2,1)\to(1,2)} & P_{(2,1)\to(2,1)} & P_{(2,1)\to(2,2)} & P_{(2,1)\to(0,1)} & P_{(2,1)\to(0,2)} & P_{(2,1)\to(1,0)} & P_{(2,1)\to(2,0)} \\ P_{(2,2)\to(1,1)} & P_{(2,2)\to(1,2)} & P_{(2,2)\to(2,1)} & P_{(2,2)\to(2,2)} & P_{(2,2)\to(0,1)} & P_{(2,2)\to(0,2)} & P_{(2,2)\to(1,0)} & P_{(2,2)\to(2,0)} \\ P_{(0,1)\to(1,1)} & P_{(0,1)\to(1,2)} & P_{(0,1)\to(2,1)} & P_{(0,1)\to(2,2)} & P_{(0,1)\to(0,1)} & P_{(0,1)\to(0,2)} & P_{(0,1)\to(1,0)} & P_{(0,1)\to(2,0)} \\ P_{(0,2)\to(1,1)} & P_{(0,2)\to(1,2)} & P_{(0,2)\to(2,1)} & P_{(0,2)\to(2,2)} & P_{(0,2)\to(0,1)} & P_{(0,2)\to(0,2)} & P_{(0,2)\to(1,0)} & P_{(0,2)\to(2,0)} \\ P_{(1,0)\to(1,1)} & P_{(1,0)\to(1,2)} & P_{(1,0)\to(2,1)} & P_{(1,0)\to(2,2)} & P_{(1,0)\to(0,1)} & P_{(1,0)\to(0,2)} & P_{(1,0)\to(1,0)} & P_{(1,0)\to(2,0)} \\ P_{(2,0)\to(1,1)} & P_{(2,0)\to(1,2)} & P_{(2,0)\to(2,1)} & P_{(2,0)\to(2,2)} & P_{(2,0)\to(0,1)} & P_{(2,0)\to(0,2)} & P_{(2,0)\to(1,0)} & P_{(2,0)\to(2,0)} \end{bmatrix}$$

Plugging in the probabilities from Table 8:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.583 & 0 & 0.417 & 0 \\ 0.255 & 0 & 0 & 0 & 0 & 0.745 & 0 & 0 \\ 0.421 & 0 & 0 & 0 & 0 & 0 & 0 & 0.579 \\ 0.324 & 0 & 0 & 0 & 0 & 0.448 & 0 & 0.228 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The sum of probabilities in each row is *1*, which makes sense as it represents all the possible states the current state can transition to in one turn.

Given *P*, and a row vector *s₀* representing the initial state, the probabilities of transitioning to each state across one turn, *s₁* can be calculated by finding their dot product i.e. $s_1 = s_0 \cdot P$

The dot product between an arbitrary row vector and matrix are shown below:

$$[a \quad b] \cdot \begin{bmatrix} m & n \\ o & p \end{bmatrix} = [am + bo \quad an + bp]$$

Applying this formula, to the *2v2* matrix, where *v* represents the current state being (2, 2), the transition probabilities across one turn can be calculated as follows:

$$s_1 = s_0 \cdot P = [0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0]$$

$$\cdot \begin{bmatrix} 0 & 0 & 0 & 0 & 0.583 & 0 & 0.417 & 0 \\ 0.255 & 0 & 0 & 0 & 0 & 0.745 & 0 & 0 \\ 0.421 & 0 & 0 & 0 & 0 & 0 & 0 & 0.579 \\ 0.324 & 0 & 0 & 0 & 0 & 0.448 & 0 & 0.228 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [0.324 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0.448 \quad 0 \quad 0.228]$$

s_1 represents the following probabilities:

$$[P_{(2,2)\to(1,1)}\ \ P_{(2,2)\to(1,2)}\ \ P_{(2,2)\to(2,1)}\ \ P_{(2,2)\to(2,2)}\ \ P_{(2,2)\to(0,1)}\ \ P_{(2,2)\to(0,2)}\ \ P_{(2,2)\to(1,0)}\ \ P_{(2,2)\to(2,0)}]$$

To find s_2, s_1 can be multiplied by P [Bazett]. However, since $s_1 = s_0 \cdot P$, s_2 can be written in terms of s_0 and P:

$$s_2 = s_0 \cdot P \cdot P = s_0 \cdot P^2$$

This involves matrix-matrix multiplication (when calculating P^2). The dot product between two arbitrary matrices are shown below:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{bmatrix}$$

More generally, given matrices *M₁* and *M₂*, where the dimensions are $a \times b$ and $b \times c$ respectively, the cell (i, j) in $M_1 \cdot M_2$ would be $\sum_{n=1}^{b} M_1(i,n) \times M_2(n,j)$.

s_2 for $a_0 = 1$ and $d_0 = 1$ can be calculated as follows (let s_0 represent $(1, 1)$):

$$s_2 = s_0 \cdot P^2 = s_0 \cdot \begin{bmatrix} 0 & 0.583 & 0.417 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0.583 & 0.417 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= s_0 \cdot \begin{bmatrix} 0 \times 0 + 0.583 \times 0 + 0.417 \times 0 & 0 \times 0.583 + 0.583 \times 1 + 0.417 \times 0 & 0 \times .417 + 0.583 \times 0 + 0.417 \times 1 \\ 0 \times 0 + 1 \times 0 + 0 \times 0 & 0 \times 0.583 + 1 \times 1 + 0 \times 0 & 0 \times 0.417 + 1 \times 0 + 0 \times 1 \\ 0 \times 0 + 0 \times 0 + 1 \times 0 & 0 \times 0.583 + 0 \times 1 + 1 \times 0 & 0 \times 0.417 + 0 \times 0 + 1 \times 1 \end{bmatrix}$$

$$= s_0 \cdot \begin{bmatrix} 0 & 0.583 & 0.417 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0.583 & 0.417 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.583 & 0.417 \end{bmatrix}$$

An interesting observation I noticed here was that $P^2 = P$, entailing that $P^n = P, n \epsilon \mathbb{N}$. This makes sense as if $a_0, d_0 = 1$, only one round can happen until either a win or loss i.e. one round for a transient state to become an absorbing state. Hence, $s_1 = s_2 = s_n$.

Applying this to find the probabilities across two turns for $a_0, d_0 = 2$, for an initial state of *(2,2)* would yield the following matrix:

$$s_2 = s_0 \cdot P^2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & 0.583 & 0 & 0.417 & 0 \\ 0 & 0 & 0 & 0 & 0.149 & 0.745 & 0.106 & 0 \\ 0 & 0 & 0 & 0 & 0.245 & 0 & 0.179 & 0.579 \\ 0 & 0 & 0 & 0 & 0.189 & 0.448 & 0.135 & 0.228 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0.189 & 0.448 & 0.135 & 0.228 \end{bmatrix}$$

Since the maximum number of turns for $a_0, d_0 = 2$ is two (refer to figure 1), $s_n = s_2, n \geq 2$.

To summarize, $s_n = s_0 \cdot P^n$, where s_n is the *n*th state, s_0 is the initial state and *P* is the transition matrix.

Organizing matrices

A transition matrix can be split into four quadrants depending on the type of transition [“Absorbing Markov Chains”]:

$$P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix}$$

Q: probabilities of transitioning between transient states. Since there are $a_0 \times d_0$ transient states, the dimensions of this quadrant would be $(a_0 \times d_0) \times (a_0 \times d_0)$

R: probabilities of transitioning from a transient to an absorbing state. There are $a_0 \times d_0$ transient and $a_0 + d_0$ absorbing states. So, the dimensions of *R* would be $(a_0 \times d_0) \times (a_0 + d_0)$

0: probabilities of transitioning from an absorbing to a transient state. The dimensions of this quadrant would be $(a_0 + d_0) \times (a_0 \times d_0)$. Since absorbing states cannot transition to any other state, this quadrant would only consist of zeros.

I: probabilities of transitioning between absorbing states. This quadrant is $(a_0 + d_0) \times (a_0 + d_0)$. Since absorbing states can only transition to themselves, all probabilities – except for the ones along the top-left to bottom-right diagonal which will be *1* – will be *0*. *I* is hence an 'Identity matrix', which is a matrix when multiplied with, does not change the value of the original matrix. Identity matrices consist of a diagonal of *1*'s from the top-left to the bottom-right corner, with the rest being filled with *0*'s. They always have the same number of rows and columns. They are analogous to multiplying a number by *1*:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot I = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1a + 0b & 0a + 1b \\ 1c + 0d & 0c + 1d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Representing the transition matrix for $a_0, d_0 = 2$ in this manner yields the following:

Table 9

(a, d)	(1, 1)	(1, 2)	(2, 1)	(2, 2)	(0, 1)	(0, 2)	(1, 0)	(2, 0)
(1, 1)	0	0	0	0	0.583	0	0.417	0
(1, 2)	0.255	0	0	0	0	0.745	0	0
(2, 1)	0.421	0	0	0	0	0	0	0.579
(2, 2)	0.324	0	0	0	0	0.448	0	0.228
(0, 1)	0	0	0	0	1	0	0	0
(0, 2)	0	0	0	0	0	1	0	0
(1, 0)	0	0	0	0	0	0	1	0
(2, 0)	0	0	0	0	0	0	0	1
			Q	R	0	I		

Hence, given *Q* and *R*, *0* and *I* can be formulated as their values are independent of *Q* and *R*.

Calculation

Let $F(n)$ be a $(a_0 \cdot d_0) \times (a_0 + d_0)$ matrix, where the *(i, j)th*...

If an absorbing state is reached in *n* turns exactly, the first $n - 1$ turns must be between transient states. The probabilities of transition across these turns would hence be Q^{n-1}. Only the last turn would be a transition between a transient and absorbing state. These probabilities are stored in *R*, as *R* consists of probabilities of transition between transient and absorbing states.

Hence, $F(n)$ would equal $Q^{n-1}R$. Q^{n-1} has dimensions $(a_0 \cdot d_0) \times (a_0 \cdot d_0)$, and *R* has dimensions $(a_0 \cdot d_0) \times (a + d_0)$. Therefore, the dot product of these two matrices would have dimensions $(a_0 \cdot d_0) \times (a_0 + d_0)$.

Given that a battle can have an infinite number of steps, even though after a certain state all states in succeeding turns will be identical, the probabilities must be a sum of all $F(n)$, where *n* is a non-negative integer:

$$F = \sum_{n=0}^{\infty} F(n) = \sum_{n=0}^{\infty} Q^{n-1}R = R\sum_{n=0}^{\infty} Q^{n-1}$$

Simplifying this equation is similar to that of a geometric series:

1) $\sum_{n=0}^{\infty} Q^{n-1} = Q^0 + Q^1 + \cdots + Q^{\infty}$
2) $Q \times \sum_{n=0}^{\infty} Q^{n-1} = Q^1 + Q^2 + \cdots + Q^{\infty}$

(Subtract equation 2 from equation 1; Let *I* be a $(a_0 \cdot d_0) \times (a_0 \cdot d_0)$ identity matrix)

$$\sum_{n=0}^{\infty} Q^{n-1} - Q \times \sum_{n=0}^{\infty} Q^{n-1} = Q^0$$

($Q^0 = I$; Factorize $\sum_{n=0}^{\infty} Q^{n-1}$)

$$\sum_{n=0}^{\infty} Q^{n-1}\,(I-Q) = I$$

$$\sum_{n=0}^{\infty} Q^{n-1} = \frac{I}{I-Q}$$

$$R\sum_{n=0}^{\infty} Q^{n-1} = \frac{R \times I}{(I-Q)} = (I-Q)^{-1}R$$

Therefore,

$$F = (I-Q)^{-1}R$$

I noticed that this formula resulted in the inversion of a matrix. If *M* is a matrix with an equal number of rows and columns, and *M'* is its inverse, $M \times M' = I$ ["Inverse of a Matrix"]. Since calculating an inverse of a matrix manually is challenging, I decided to use Python's *NumPy* library for calculating it and all other matrix operations.

Applying this formula to a battle with $a_0, d_0 = 2$ yields the following:

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.255 & 0 & 0 & 0 \\ 0.421 & 0 & 0 & 0 \\ 0.324 & 0 & 0 & 0 \end{bmatrix}, R = \begin{bmatrix} 0.583 & 0 & 0.417 & 0 \\ 0 & 0.745 & 0 & 0 \\ 0 & 0 & 0 & 0.579 \\ 0 & 0.448 & 0 & 0.228 \end{bmatrix}$$

$$I - Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.255 & 0 & 0 & 0 \\ 0.421 & 0 & 0 & 0 \\ 0.324 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.255 & 1 & 0 & 0 \\ -0.421 & 0 & 1 & 0 \\ -0.324 & 0 & 0 & 1 \end{bmatrix}$$

$$(I-Q)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.255 & 1 & 0 & 0 \\ -0.421 & 0 & 1 & 0 \\ -0.324 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.255 & 1 & 0 & 0 \\ 0.421 & 0 & 1 & 0 \\ 0.324 & 0 & 0 & 1 \end{bmatrix}$$

$$(I-Q)^{-1}R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.255 & 1 & 0 & 0 \\ 0.421 & 0 & 1 & 0 \\ 0.324 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.583 & 0 & 0.417 & 0 \\ 0 & 0.745 & 0 & 0 \\ 0 & 0 & 0 & 0.579 \\ 0 & 0.448 & 0 & 0.228 \end{bmatrix}$$

$$= \begin{bmatrix} 0.583 & 0 & 0.417 & 0 \\ 0.149 & 0.745 & 0.106 & 0 \\ 0.245 & 0 & 0.176 & 0.579 \\ 0.189 & 0.448 & 0.135 & 0.228 \end{bmatrix}$$

To calculate the probability of losing, the first d_0 columns must summed, as the columns and A_1 are indexed the same. Likewise, to calculate the probability of winning, the columns from $\mathrm{d}_0 + 1$ to $\mathrm{d}_0 + \mathrm{a}_0$ must be summed. Both these calculations will result in column vectors of length $\mathrm{a}_0 \cdot \mathrm{d}_0$. Summing these vectors will result in a vector where each element is *1*:

$$\mathrm{P(L)} = \begin{bmatrix} 0.583 + 0 = 0.583 \\ 0.149 + 0.745 = 0.894 \\ 0.245 \\ 0.189 + 0.448 = 0.637 \end{bmatrix}, \mathrm{P(W)} = \begin{bmatrix} 0.417 \\ 0.106 \\ 0.176 + 0.579 = 0.755 \\ 0.135 + 0.228 = 0.363 \end{bmatrix}, \mathrm{P(W)} + \mathrm{P(L)} = \begin{bmatrix} 1.00 \\ 1.00 \\ 1.00 \\ 1.00 \end{bmatrix}$$

The probability of winning given a battle of (*2*, *2*) according to this table is *0.363*. This is equal to the value calculated from Figure 1.

An advantage of this method is that given a_0 and d_0, the vectors $\mathrm{P(W)}$ and $\mathrm{P(L)}$ include the probabilities of all combinations of attackers and defenders up till (a_0, d_0). Therefore, to calculate the $\mathrm{P(W)}$ and $\mathrm{P(L)}$ for all combinations up to $a_0, d_0 = 40$, I will only need to go over this method once for $a_0, d_0 = 40$. However, since the Q and R quadrant of this matrix would be of dimensions $(a_0 \cdot d_0) \times (a_0 \cdot d_0)$ and $(a_0 \cdot d_0) \times (a_0 + d_0)$ respectively, i.e. 1600×1600 and 1600×80, I will not be able to calculate this manually.

Even for matrices where $a_0, d_0 = 3$ – the dimensions become quadratically large and thus have a quartic increase in the number of probabilities (Q would have dimensions 9×9) – manual calculations become tedious. Therefore, I coded a program in Python that uses NumPy to do these calculations and fills Q and R automatically (Appendix B).

Reflection:

Given these calculations, I was particularly content with how I used my knowledge of geometric series and extrapolated it to matrix operations to simplify what would be a near impossible task to one that could be easily computed. I chose to use *40* as the upper bound of troops for two reasons. Firstly, my set comes with *40* infantry troops. Secondly, I attempted to run the program with an upper bound of *60*, causing the program to crash. Considering other types of troops, the total troops one can have is *187*. However, I do not expect this to run without crashing. I plan to improve my program's efficiency to be able to handle larger upper bounds.

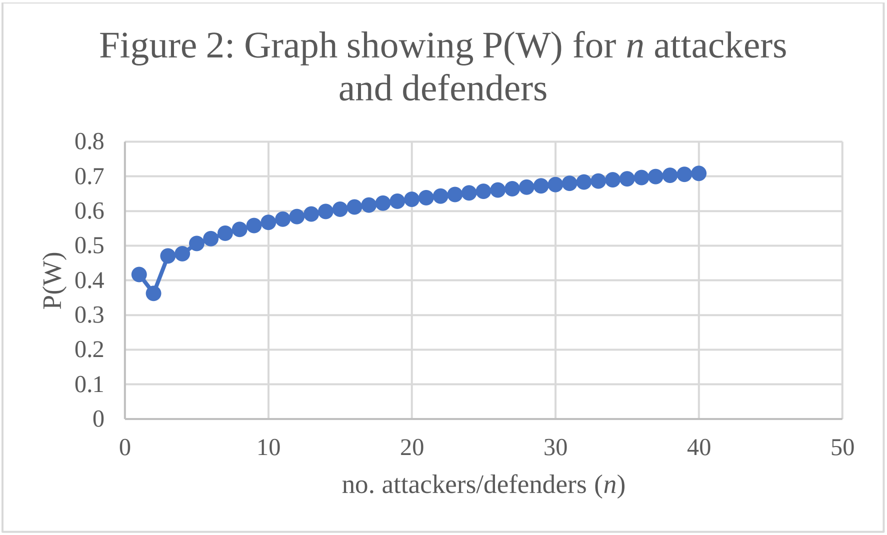
Results

Below is a table showing the probabilities for winning given a attacking troops and d defending troops, where $a = d$:

Table 10

No. Troops	P(W)	No. Troops	P(W)	No. Troops	P(W)	No. Troops	P(W)
1	0.417	11	0.576	21	0.638	31	0.680
2	0.363	12	0.584	22	0.643	32	0.683
3	0.470	13	0.592	23	0.648	33	0.687
4	0.477	14	0.599	24	0.652	34	0.690
5	0.506	15	0.605	25	0.656	35	0.693
6	0.521	16	0.611	26	0.661	36	0.697
7	0.536	17	0.617	27	0.665	37	0.700
8	0.547	18	0.623	28	0.669	38	0.703
9	0.558	19	0.628	29	0.672	39	0.706
10	0.568	20	0.633	30	0.676	40	0.709

The graph below (Figure 2) shows the probability in Table 10:



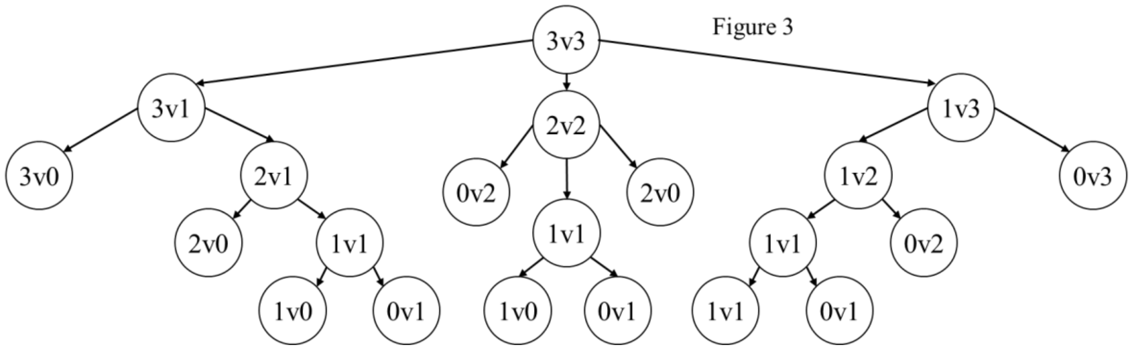
The graph shows that when both sides have 5 or more troops, the attacker has a higher chance of winning. When both sides have 4 or less troops, the defender is more likely to win.

Alternate method:

I also approached this problem through a technique I learned in Computer Science called ‘Dynamic Programming’ [“Dynamic Programming”]. Essentially, the probability for winning given *a₀* and *d₀* i.e. $\mathrm{P(W|a_0, d_0)}$ can be recursively defined by the three possible states it will transition to in one step: $\mathrm{P(W|a_0, d_0) = P(T|a_0 - 1, d_0 - 1) + P(W|a_0, d_0 - 2) + P(L|a_0 - 2, d_0)}$. This equation does not include edge cases i.e. when either a_0 or d_0 are less than 2.

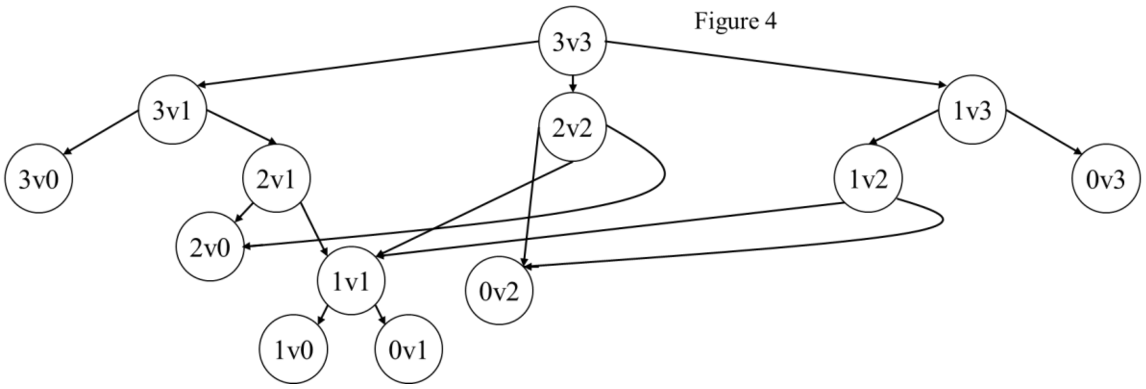
However, computing this would result in an exponential runtime with a base 3 as in each succeeding step, the probability splits three-way. For instance, the graph below shows all possible paths for a *3v3* battle:

Figure 3



In the graph, several states repeat themselves. For instance, *1v1* is present three times, *2v0* and *0v2* are present two times each. Hence, these probabilities are calculated more than once, which is inefficient. I expected that for larger values of a_0 and d_0, the number of repeated calculations would be larger. Therefore, I decided to use 'memoization', a technique in programming that stores calculated probabilities in a table that could be referred to later, rather than having a redundant calculation. For instance, once the $\mathrm{P(W|1,1)}$ has been calculated, it can be stored in the table and used later. Figure 4 shows a simplified version of the graph that uses memoization:

Figure 4



The graph in Figure 4 consists of 13 nodes, while Figure 3 consists of 21 nodes. This difference should increase as a_0 and d_0 increase.

The results for the program were identical to the results of the previous method, and the code can be found in Appendix C, and the data in Appendix D.

Conclusion

The results show a general trend that, given an equal number of troops, as the number of troops increases, the chances for the attacker to win increases as well. This makes intuitive sense, as when more troops are present, the attacker can roll 3 dice a greater fraction of the total number of rounds. Since in a *3v2* round the probability for winning is higher than that of losing, an increase in the portion of *3v2* rounds increases the chances of winning. If both sides have greater than or equal to five troops, the attacker is more likely to win. Hence, it makes sense to start and attack when five troops are present.

Final Reflection

Strengths:

- I was able to calculate the probability to win given more than *20* troops. This is significant as every other website I saw had an upper bound of *20*. Hence, I was able to produce a greater quantity of accurate data through this investigation than most other investigations.
- I used multiple approaches to each problem. For calculating the probabilities of each individual round, I used two methods: counting, a simulation. For calculating the probabilities of battles, I used an approach involving matrices and dynamic programming. This ensured that my data was accurate.

Limitations:

- The investigation does not consider the wider game. If the attacker wins but is likely to only have a few troops remaining after the battle, it does not make sense to attack, as the other player can easily push a counter-offensive next turn and take back the newly gained territory. A more useful metric here would be the expected number of troops remaining after launching an offensive.

- The investigation does not consider the geography of the territory. Choosing to attack is not solely dependent on the number of troops in either territory. Instead, it depends on which continent the other territory is in, and whether surrounding territories are fortified.
- The investigation does not account for when the players do not send the maximum number of troops in each round. When initially starting off the game, me and my brother would not send the maximum number of troops we have. This makes intuitive sense as the defender/attacker can spread out the battle over a greater number of rounds. However, since this is not the optimal strategy, I decided to ignore these cases. Despite this, players may still not send the maximum number of troops. Hence, I would like to account for this.
- When storing the probabilities for winning a single round i.e. the dice probabilities, I used decimals to 6-places rather than fractions. This affected the precision of my data for larger numbers of troops. Using fractions could have eliminated this issue. However, at the time, I did not know how to

Extension

I would like to calculate the expected number of troops remaining after each battle. This should be easy as the data already exists in the matrix *F* as it stores the probability of ending in each absorbing state. Additionally, I plan to use Monte Carlo simulations to find more significant probabilities, such as conquering a continent, or winning the game to deduce which territories may be the best to start off with.

Works cited

“Absorbing Markov Chains.” *Brilliant Math & Science Wiki*, https://brilliant.org/wiki/absorbing-markov-chains/.

Bazett, Trefor. “Markov Chains & Transition Matrices - Youtube.” *YouTube*, https://www.youtube.com/watch?v=1GKtfgwf3ig.

“Dynamic Programming.” *GeeksforGeeks*, https://www.geeksforgeeks.org/dynamic-programming/.

“Inverse of a Matrix.” *Math Is Fun*, https://www.mathsisfun.com/algebra/matrix-inverse.html.

“Risk Game Rules.” *UltraBoardGames*, https://www.ultraboardgames.com/risk/game-rules.php.

Appendix A:

```c
int main (void)
{
    int numWins = 0;
    int numLosses = 0;
    int numTies = 0;
    int a[3];
    int d[n];
    int n = 0;
    int aMax, dMax, aSecond, dSecond;
    printf("enter num trials: ");
    scanf("%d", &n);

    int i;
    for (i = 0; i < n; i++)
    {
        // generate random dice rolls for attacker
        a[0] = ((rand() + 1) % 6) + 1;
        a[1] = ((rand() + 1) % 6) + 1 ;
        a[2] = ((rand() + 1) % 6) + 1;
        // generate random dice rolls for defender
        d[0] = ((rand() + 1) % 6) + 1;
        d[1] = ((rand() + 1) % 6) + 1;
        //sort the dice rolls to find the max and second largest
        bubbleSort(a, 3);
        bubbleSort(d, 2);

        aMax = a[0];
        aSecond = a[1];

        dMax = d[0];
        dSecond = d[0];
        // compare
        if (aMax > dMax)
            numWins++;
        else if (aMax <= dMax)
            numLosses++;
    }
    printf("Wins: %d/%d\n", numWins, i);
    printf("Losses: %d/%d\n", numLosses, i);
    printf("Ties: %d/%d\n", numTies, i);
    printf("\n");
}
```

Appendix B

```
1.  def main():
        a = int(input("Enter a: "))
        d = int(input("Enter d: "))

        qDimension = (a * d + 1)
        rDimension = (a + d + 1)
        q = np.zeros([qDimension, qDimension])
        r = np.zeros([qDimension, rDimension])
        # fill Q using dice data and the algorithms
        fillMat(a, d, q, qDimension, qDimension, 'q')

2.      # fill R using dice data and the algorithms
        fillMat(a, d, r, qDimension, rDimension, 'r')
        newQ = np.delete(q, 0, 1)
        newQ = np.delete(newQ, 0, 0)

        newR = np.delete(r, 0, 1)
        newR = np.delete(newR, 0, 0)
        #print(newQ)

        i = 1
        # make identity matrix
        i = np.identity(qDimension - 1)

3.      # subtract q from i
        diff = np.subtract(i, newQ)

4.      # find inverse of difference
        inverse = np.linalg.inv(diff)

5.      # multiply by R
        final = np.dot(inverse, newR)
        loss = final[:, : d]
        wins = final[:, d: d+a]

        lossVector = np.sum(loss, 1)
        winsVector = np.sum(wins, 1)

        fWin = open('win.csv', 'w')
        for i in range(a*d):
            for j in range(a):
                fWin.write(str(wins[i][j]))
                fWin.write(', ')
            fWin.write('\n')
        fLose = open('lose.csv', 'w')
        for i in range(a*d):
            for j in range(d):
                fLose.write(str(loss[i][j]))
                fLose.write(', ')
            fLose.write('\n')

        fWinV = open('winV.csv', 'w')
        for i in range (a*d):
            fWinV.write(str(winsVector[i]))
            fWinV.write('\n')
        fLoseV = open('LoseV.csv', 'w')
        for i in range(a * d):
            fLoseV.write(str(lossVector[i]))
            fLoseV.write('\n')

        winsEqualAB = []
        for i in range(a*d):
            if (i)%(d+1) == 0:
                winsEqualAB.append(winsVector[i])
        print(len(winsEqualAB))
        fWinEquals = open('winEquals.csv', 'w')
        for i in range(min(a,d)):
            fWinEquals.write(str(winsEqualAB[i]))
            fWinEquals.write('\n')
```

Appendix C

```
float pW(int a, int d)
{
    int usableAttackers = min(a, 3);
    int usableDefenders = min(d, 2);

    float case1 = p[d - min(usableAttackers, usableDefenders)][a];
    if (case1 < -0.5)
        case1 = pW(a, d - min(usableAttackers, usableDefenders));

    float case2 = p[d][a - min(usableAttackers, usableDefenders)];
    if (case2 < -0.5)
        case2 = pW(a - min(usableAttackers, usableDefenders), d);

    float case3 = p[d - 1][a - 1];
    if (case3 < -0.5)
        case3 = pW(a - 1, d - 1);

    float pWin = dice(usableAttackers, usableDefenders, 0)*case1 + dice(usableAttackers, usableDefenders, 1)*case2 + dice(usableAttackers, usableDefenders, 2)*case3;
    // memoization step
    p[d][a] = pWin;

    return pWin;
}
```

Appendix D

def\att	1	2	3	4	5	6	7	8
1	0.417	0.754	0.916	0.972	0.990	0.997	0.999	1.000
2	0.106	0.363	0.656	0.785	0.890	0.934	0.967	0.980
3	0.027	0.206	0.470	0.642	0.769	0.857	0.910	0.947
4	0.007	0.091	0.315	0.477	0.638	0.745	0.834	0.888
5	0.002	0.049	0.206	0.359	0.506	0.638	0.736	0.818
6	0.000	0.021	0.134	0.253	0.397	0.521	0.640	0.730
7	0.000	0.011	0.084	0.181	0.297	0.423	0.536	0.643
8	0.000	0.005	0.054	0.123	0.224	0.329	0.446	0.547
9	0.000	0.003	0.033	0.086	0.162	0.258	0.357	0.464
10	0.000	0.001	0.021	0.057	0.118	0.193	0.287	0.380
11	0.000	0.001	0.013	0.039	0.083	0.147	0.222	0.312
12	0.000	0.000	0.008	0.026	0.059	0.107	0.173	0.247
13	0.000	0.000	0.005	0.017	0.041	0.080	0.130	0.197
14	0.000	0.000	0.003	0.011	0.029	0.057	0.100	0.152
15	0.000	0.000	0.002	0.007	0.019	0.041	0.073	0.119
16	0.000	0.000	0.001	0.005	0.014	0.029	0.055	0.090
17	0.000	0.000	0.001	0.003	0.009	0.021	0.040	0.069
18	0.000	0.000	0.000	0.002	0.006	0.014	0.029	0.051
19	0.000	0.000	0.000	0.001	0.004	0.010	0.021	0.038
20	0.000	0.000	0.000	0.001	0.003	0.007	0.015	0.028
21	0.000	0.000	0.000	0.001	0.002	0.005	0.011	0.021
22	0.000	0.000	0.000	0.000	0.001	0.003	0.008	0.015
23	0.000	0.000	0.000	0.000	0.001	0.002	0.005	0.011
24	0.000	0.000	0.000	0.000	0.001	0.002	0.004	0.008
25	0.000	0.000	0.000	0.000	0.000	0.001	0.003	0.006
26	0.000	0.000	0.000	0.000	0.000	0.001	0.002	0.004
27	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.003
28	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.002
29	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.001
30	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001
31	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001
32	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
33	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
34	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
35	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
36	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
37	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
38	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
39	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
40	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

9	10	11	12	13	14	15	16
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0.990	0.994	0.997	0.998	0.999	1.000	1.000	1.000
0.967	0.981	0.988	0.993	0.996	0.998	0.999	0.999
0.930	0.954	0.972	0.982	0.989	0.993	0.996	0.998
0.873	0.916	0.943	0.964	0.976	0.985	0.990	0.994
0.808	0.861	0.905	0.934	0.956	0.970	0.981	0.987
0.726	0.800	0.852	0.896	0.925	0.949	0.964	0.976
0.646	0.724	0.794	0.845	0.889	0.918	0.943	0.959
0.558	0.650	0.723	0.790	0.839	0.882	0.912	0.938
0.480	0.568	0.654	0.723	0.787	0.835	0.877	0.907
0.400	0.494	0.576	0.658	0.723	0.784	0.831	0.872
0.334	0.417	0.506	0.584	0.661	0.724	0.783	0.828
0.270	0.353	0.433	0.518	0.592	0.665	0.725	0.782
0.219	0.290	0.371	0.448	0.528	0.599	0.669	0.726
0.173	0.240	0.309	0.387	0.461	0.538	0.605	0.672
0.138	0.192	0.259	0.326	0.402	0.473	0.547	0.611
0.106	0.155	0.210	0.276	0.342	0.416	0.484	0.555
0.083	0.122	0.173	0.228	0.293	0.357	0.428	0.494
0.062	0.097	0.137	0.189	0.244	0.308	0.371	0.440
0.048	0.074	0.111	0.152	0.205	0.259	0.323	0.384
0.036	0.058	0.086	0.124	0.167	0.220	0.274	0.336
0.027	0.044	0.069	0.099	0.138	0.181	0.234	0.288
0.020	0.034	0.053	0.079	0.111	0.151	0.195	0.248
0.015	0.025	0.041	0.062	0.090	0.123	0.164	0.208
0.011	0.019	0.031	0.049	0.071	0.101	0.134	0.177
0.008	0.014	0.024	0.038	0.057	0.080	0.111	0.146
0.006	0.011	0.018	0.029	0.044	0.065	0.090	0.122
0.004	0.008	0.014	0.022	0.035	0.051	0.073	0.099
0.003	0.006	0.010	0.017	0.027	0.041	0.058	0.082
0.002	0.004	0.008	0.013	0.021	0.032	0.047	0.066
0.001	0.003	0.006	0.010	0.016	0.025	0.037	0.054
0.001	0.002	0.004	0.007	0.012	0.019	0.030	0.043
0.001	0.002	0.003	0.006	0.009	0.015	0.023	0.034
0.001	0.001	0.002	0.004	0.007	0.011	0.018	0.027
0.000	0.001	0.002	0.003	0.005	0.009	0.014	0.021
0.000	0.001	0.001	0.002	0.004	0.007	0.011	0.017
0.000	0.000	0.001	0.002	0.003	0.005	0.008	0.013
0.000	0.000	0.001	0.001	0.002	0.004	0.006	0.010
0.000	0.000	0.000	0.001	0.002	0.003	0.005	0.008
0.000	0.000	0.000	0.001	0.001	0.002	0.004	0.006

17	18	19	20	21	22	23	24
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0.999	0.999	0.999	1.000	1.000	1.000	1.000	1.000
0.996	0.998	0.999	0.999	0.999	1.000	1.000	1.000
0.992	0.995	0.997	0.998	0.999	0.999	0.999	1.000
0.984	0.989	0.993	0.995	0.997	0.998	0.999	0.999
0.972	0.981	0.987	0.991	0.994	0.996	0.997	0.998
0.955	0.969	0.978	0.985	0.989	0.993	0.995	0.997
0.933	0.950	0.965	0.975	0.983	0.988	0.992	0.994
0.902	0.928	0.946	0.962	0.972	0.980	0.986	0.990
0.869	0.898	0.925	0.943	0.959	0.969	0.978	0.984
0.826	0.865	0.895	0.921	0.940	0.956	0.967	0.976
0.781	0.824	0.863	0.892	0.918	0.937	0.953	0.964
0.728	0.781	0.822	0.860	0.889	0.915	0.934	0.950
0.676	0.729	0.781	0.821	0.858	0.887	0.913	0.931
0.617	0.680	0.731	0.781	0.820	0.857	0.885	0.910
0.563	0.623	0.683	0.733	0.781	0.819	0.855	0.883
0.504	0.570	0.628	0.686	0.735	0.781	0.819	0.854
0.451	0.513	0.577	0.633	0.690	0.737	0.782	0.819
0.396	0.462	0.522	0.584	0.638	0.693	0.738	0.783
0.349	0.408	0.472	0.530	0.591	0.643	0.696	0.740
0.301	0.361	0.419	0.481	0.538	0.597	0.648	0.699
0.261	0.314	0.373	0.429	0.490	0.545	0.602	0.652
0.221	0.274	0.326	0.384	0.439	0.499	0.552	0.608
0.189	0.234	0.286	0.337	0.395	0.449	0.507	0.559
0.158	0.201	0.246	0.297	0.348	0.405	0.458	0.515
0.133	0.169	0.212	0.257	0.309	0.359	0.415	0.467
0.109	0.143	0.180	0.224	0.268	0.320	0.369	0.424
0.091	0.119	0.154	0.191	0.235	0.279	0.330	0.379
0.074	0.099	0.128	0.164	0.201	0.246	0.290	0.340
0.060	0.081	0.108	0.138	0.174	0.212	0.256	0.300
0.048	0.067	0.089	0.117	0.147	0.184	0.222	0.266
0.039	0.054	0.074	0.097	0.126	0.156	0.194	0.232
0.031	0.044	0.060	0.081	0.105	0.134	0.166	0.203
0.025	0.035	0.050	0.067	0.089	0.113	0.143	0.175
0.020	0.029	0.040	0.055	0.073	0.096	0.121	0.152
0.016	0.023	0.033	0.045	0.061	0.080	0.103	0.129
0.012	0.018	0.026	0.037	0.050	0.067	0.086	0.111
0.010	0.014	0.021	0.030	0.041	0.055	0.073	0.093

25	26	27	28	29	30	31	32
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0.999	0.999	1.000	1.000	1.000	1.000	1.000	1.000
0.998	0.999	0.999	0.999	1.000	1.000	1.000	1.000
0.996	0.997	0.998	0.999	0.999	0.999	1.000	1.000
0.993	0.995	0.997	0.998	0.998	0.999	0.999	1.000
0.989	0.992	0.994	0.996	0.997	0.998	0.999	0.999
0.982	0.987	0.991	0.994	0.995	0.997	0.998	0.999
0.974	0.981	0.986	0.990	0.993	0.995	0.996	0.997
0.962	0.972	0.979	0.985	0.989	0.992	0.994	0.996
0.948	0.960	0.970	0.977	0.984	0.988	0.991	0.993
0.929	0.946	0.958	0.969	0.976	0.982	0.987	0.990
0.908	0.927	0.944	0.956	0.967	0.975	0.981	0.986
0.881	0.906	0.925	0.942	0.955	0.965	0.973	0.980
0.853	0.880	0.905	0.924	0.940	0.953	0.964	0.972
0.818	0.852	0.879	0.903	0.922	0.939	0.951	0.963
0.784	0.818	0.852	0.878	0.902	0.921	0.937	0.950
0.742	0.784	0.819	0.851	0.877	0.901	0.919	0.936
0.702	0.744	0.785	0.819	0.851	0.876	0.900	0.918
0.656	0.705	0.746	0.786	0.819	0.851	0.875	0.899
0.613	0.661	0.708	0.748	0.787	0.820	0.850	0.875
0.566	0.619	0.665	0.711	0.750	0.789	0.820	0.850
0.522	0.572	0.624	0.669	0.714	0.752	0.790	0.821
0.475	0.529	0.578	0.629	0.672	0.717	0.754	0.791
0.433	0.483	0.536	0.584	0.633	0.676	0.719	0.756
0.389	0.442	0.491	0.543	0.590	0.638	0.680	0.722
0.350	0.398	0.450	0.498	0.549	0.595	0.642	0.683
0.310	0.360	0.407	0.458	0.505	0.556	0.600	0.647
0.276	0.320	0.369	0.415	0.466	0.512	0.562	0.605
0.242	0.286	0.329	0.378	0.424	0.473	0.519	0.567
0.213	0.251	0.295	0.338	0.386	0.432	0.481	0.526
0.184	0.222	0.260	0.304	0.347	0.395	0.439	0.488
0.160	0.193	0.231	0.270	0.313	0.356	0.403	0.447
0.137	0.169	0.202	0.240	0.279	0.322	0.364	0.411
0.118	0.145	0.177	0.210	0.249	0.287	0.331	0.373

33	34	35	36	37	38	39	40
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0.999	0.999	1.000	1.000	1.000	1.000	1.000	1.000
0.998	0.999	0.999	0.999	1.000	1.000	1.000	1.000
0.997	0.998	0.999	0.999	0.999	1.000	1.000	1.000
0.995	0.997	0.998	0.998	0.999	0.999	0.999	1.000
0.993	0.995	0.996	0.997	0.998	0.999	0.999	0.999
0.990	0.992	0.994	0.996	0.997	0.998	0.998	0.999
0.985	0.989	0.991	0.994	0.995	0.997	0.998	0.998
0.979	0.984	0.988	0.991	0.993	0.995	0.996	0.997
0.971	0.978	0.983	0.987	0.990	0.993	0.995	0.996
0.961	0.969	0.977	0.982	0.986	0.990	0.992	0.994
0.949	0.960	0.968	0.976	0.981	0.986	0.989	0.992
0.935	0.947	0.959	0.967	0.975	0.980	0.985	0.988
0.917	0.934	0.946	0.958	0.966	0.974	0.979	0.984
0.898	0.916	0.933	0.945	0.957	0.965	0.973	0.979
0.875	0.897	0.915	0.932	0.944	0.956	0.964	0.972
0.850	0.874	0.897	0.914	0.931	0.943	0.955	0.963
0.821	0.850	0.874	0.896	0.914	0.930	0.942	0.954
0.792	0.822	0.851	0.874	0.896	0.913	0.929	0.942
0.758	0.793	0.822	0.851	0.874	0.895	0.912	0.928
0.725	0.760	0.795	0.823	0.851	0.873	0.895	0.912
0.687	0.727	0.762	0.796	0.824	0.851	0.873	0.895
0.651	0.690	0.730	0.764	0.797	0.825	0.852	0.873
0.610	0.655	0.693	0.732	0.765	0.798	0.825	0.852
0.573	0.615	0.659	0.697	0.735	0.767	0.800	0.826
0.532	0.578	0.620	0.663	0.700	0.737	0.769	0.801
0.495	0.538	0.584	0.624	0.666	0.703	0.740	0.771
0.454	0.501	0.544	0.589	0.629	0.670	0.706	0.742
0.419	0.461	0.508	0.550	0.594	0.633	0.674	0.709