The Use of Mathematics and Matlab for Calculating and Analyzing Strategy of the Game of Risk

Sara Koochagian

November 18, 2022

1 Abstract

The purpose of this paper is to analyze the best strategies to use in the game of World Domination Risk, also known as Risk. This paper begins with the background of the game and how it is played. Then the paper explains the process of finding the probabilities of each result, and creating the Matlab program to simulate a game with different number of armies for the attack and defense. The application of the simulation will be presented with explanations of each individual result. The paper will then end with discussion the conclusions about which strategy will work best for each scenario for the attack and defense side.

2 Introduction

The game of Risk is a game that has strategy in order to play well. Many things can change in the initial scenarios that can change what the best strategy is to use in the game. Based on how the game of Risk works, different scenarios may have different strategies for the defense side.

3 Background

The game of Risk is a game that requires a lot of decision making. There are two sides to the game; the attacker and the defender. Before each roll, each side has to choose how many armies they want to send to battle. The attack side can send up to three armies, while the defense side can send up to two armies. Once both sides have declared the amount of armies they intend to send, both sides will role the number of dice in accordance to the amount of armies they send out. Both players role their dice at the same time, and the battles are then decided.

Between the two players, there are certain outcomes from the battles. The max of each side will go against each other, and if each side has two or more dice the second highest dice will go against each other. The player with the lower dice will lose an army, while the player with the higher dice will keep their army. For example, if the attack rolls a six and the defense rolls a four, the defense will lose their army. In the event of a tie, the attack loses their army.

4 Methodology

In order to create the code for the Matlab program, the first task is to find the probabilities of attackers and defenders for each type of game. There are six potential outcomes depending on how

many armies each side sends out. The possible outcome for each dice roll, or trial, in the game include:

1. attack loses 2 armies, defense loses 0 armies
2. attack loses 1 army, defense loses 0 armies
3. attack loses 1 army, defense loses 1 army
4. attack loses 0 armies, defense loses 1 army
5. attack loses 0 armies, defense loses 2 armies

Not all of these outcomes happen within each type of game scenario. For example, if each side sends out one army each it is not possible for defense to lose two armies. For scenarios like this, the probability would be 0. In order to find the other ones, an excel spreadsheet was created to map out every potential roll that the attack and defense could roll based on the number of armies each side sent out. After calculating each of the possible dice rolls, the total amount of armies lost for each side were calculated, and creating the probabilities for each potential scenario.

These probabilities created the probability matrix used for the Matlab code that will be explained later in this paper. The matrix was set up by having rows be the different combinations of how many armies each side sent, and the columns being the possible outcomes of how many armies can be lost by the attack or the defense. This created a six rows 5 column matrix which can be seen here:

Outcomes	def lose 2	both lose 1	atk loses 2	def loses 1	atk loses 1
3 atk v. 2 def	$\frac{2275}{7776}$	$\frac{2611}{7776}$	$\frac{2890}{7776}$	0	0
3 atk v. 1 def	0	0	0	$\frac{441}{1296}$	$\frac{855}{1296}$
2 atk v. 2 def	$\frac{581}{1296}$	$\frac{420}{1296}$	$\frac{295}{1296}$	0	0
2 atk v. 1 def	0	0	0	$\frac{91}{216}$	$\frac{125}{216}$
1 atk v. 2 def	0	0	0	$\frac{161}{216}$	$\frac{55}{216}$
1 atk v. 1 def	0	0	0	$\frac{21}{36}$	$\frac{15}{36}$

After finding the probability matrix, it was used as the matrix to use in the Matlab code. The first Matlab program code is the scenario of the attack sending three armies and the defense sending two armies. The code uses if and while loops, so there need to be initial conditions created before the while loops.

The first defined element is the initial state. This state is the scenario of how many armies each side sends out based on the location of the matrix. According to the matrix, since the 3 attack vs. 2 defense is in the first row, the initial state is 1. For the other code, for 3 attack vs. 1 defense, the initial state would have the initial state of 2. By setting the initial state to the corresponding row, we then set the state to equal the initial state to start. Once the loops in the program are ran, the state will change based on the result of the trial.

The next defined elements are the number of total armies for each side. It is defined in the beginning, and through each trial the number will change. After the number of armies, the events are defined and calculated in a table per trial.

The while loop is the loop for how to take away armies from one or both sides. This loop is set to apply when both attack and defense have more than one army. The loop takes in a randomly generated number from 0 to 1. Looking at the randomly generated number, the value will fall into a interval in the rows. This is where the cumulative probability row equation comes into the while loop. The cumulative probability row of the elements for the row of states adds up the elements in the row, which for probabilities will equal 1. The while loop takes the random variable and sees what interval the probability falls in between in the cumulative probability row. The index equation in the while loop puts the random variable in an section of the probability row, and once the index is found the program is able to perform an action.

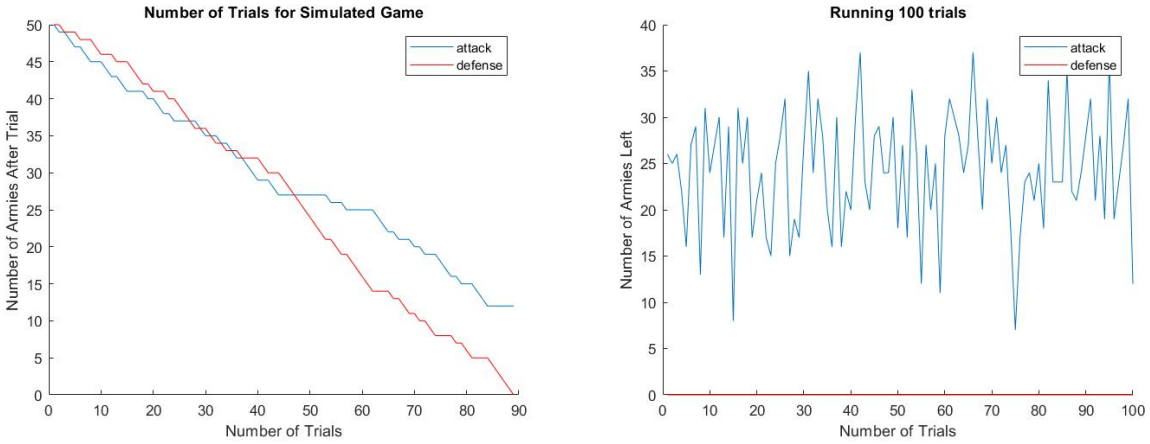
Each action is split up into five different cases, where the number of armies for either side are reduced. The if loops are used to determine which state should be used for the next trial. The if loops observe how many armies each side has, and based on the number a certain state used. This helps for when there are not enough armies on one side to where a state cannot be used. Each of these trials are recorded, and the loop will end when one side goes to 0.

After the simulation is complete, the program creates two graphs. The two graphs are then created, where the first figure shows the simulation of one game, and the second figure shows the armies of 100 trials. The Matlab coding can be found in the appendix of this paper. The code for the 3 attack versus 1 defense is very similar, but has different states since the defense can never send out more than one army at a time.

5 Findings

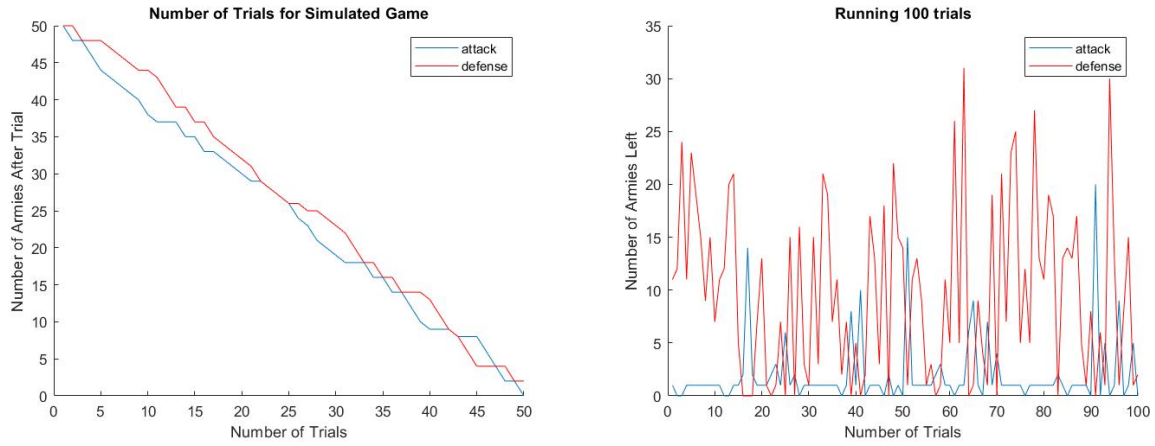
After running both simulations, the graphs were created in order to observe the simulation of a full game and the simulation of running 100 trials. Each starting number of armies will have four associated graphs, two for three attack vs. one defense and two defense. The average results were found for each scenario with the different initial number of armies and the amount of armies sent.

5.1 50 attack armies vs. 50 defense armies



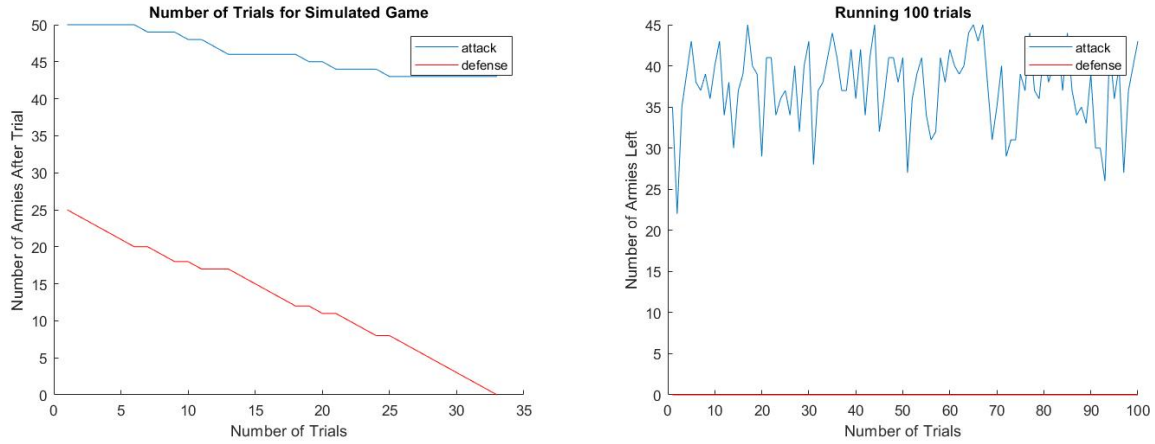
These graphs are for the attack using three armies and defense using one. The particular simulated game that played to the end had the attack winning in 90 trials. When going through the 100 trials, the result average was 24.24 for the attack and 0 for the defense. This means that

out of 100 games, the defense did not win a single game when they started off with same number of armies as the attack.

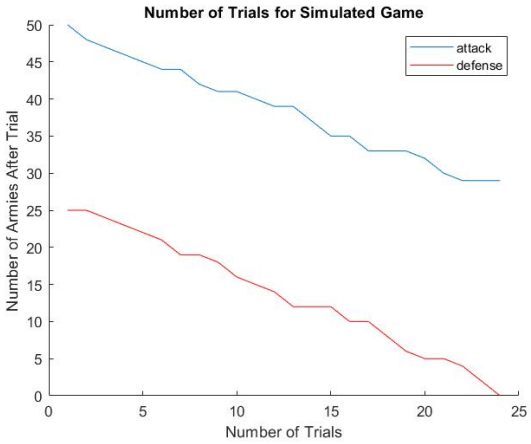


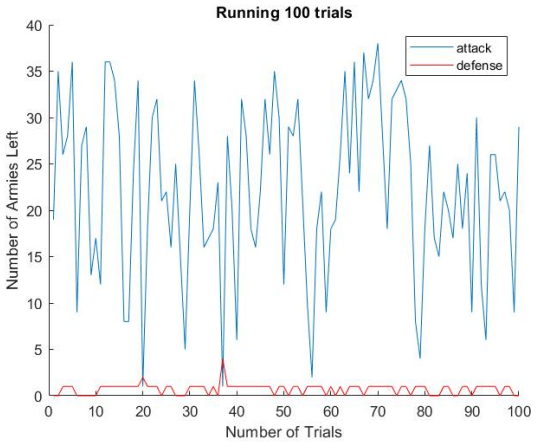
These graphs represent the simulation with the attack using three armies and the defense using two armies. The attack won the single simulation in 50 trials. For the 100 trials, the average results are 1.9700 for the attack and 9.5700 for the defense. This means that out of 100 games, the defense was more probable in winning a game than the attack, but it was not a guaranteed win.

5.2 50 attack armies vs. 25 defense armies



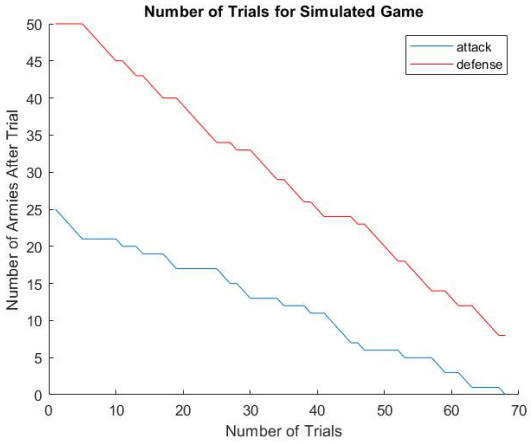
These graphs are for the attack using three armies and defense using one. The particular simulated game that played to the end had the attack winning in roughly 35 trials. When going through the 100 trials, the result average was 37.46 for the attack and 0 for the defense. This means for the 100 games, the defense did no win a single game when the attack started with more armies.

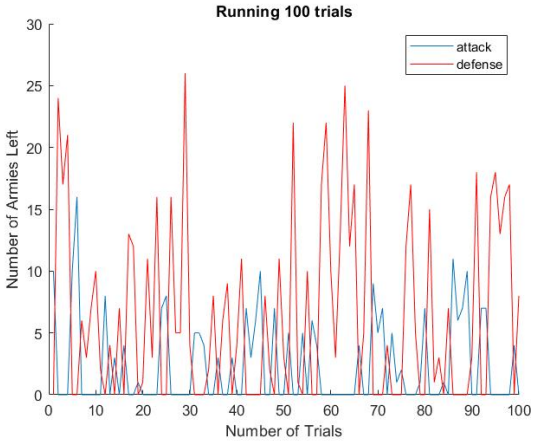




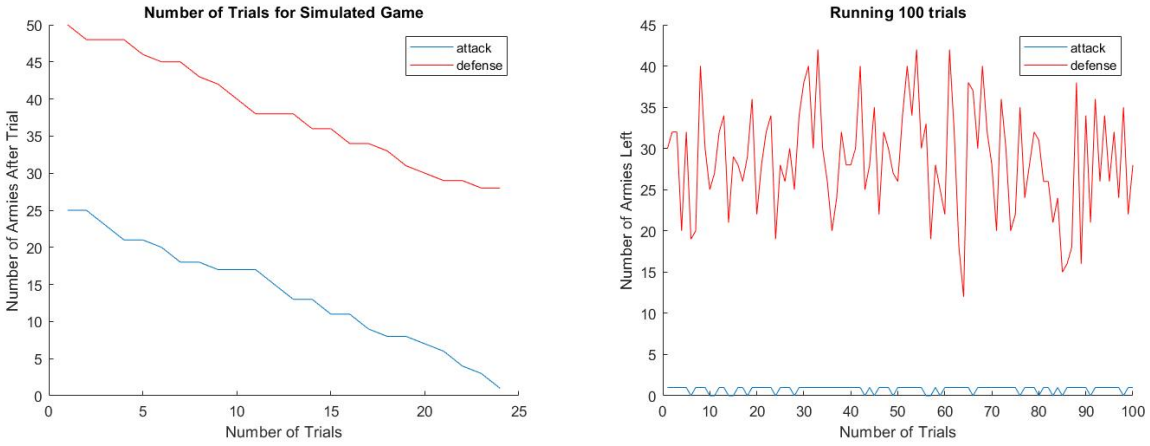
These graphs are for the attack using three armies and defense using two. The particular simulated game that played to the end had the attack winning in roughly 25 trials. When going through the 100 trials, the result average was 22.23 for the attack and 0.7300 for the defense. This means that in most cases the defense lost the round against the attack. However, according to the graph there were a few cases where the defense did beat out the attack, but the attack was still more probable to win.

5.3 25 attack armies vs. 50 defense armies





These graphs are for the attack using three armies and defense using one. The particular simulated game that played to the end had the attack winning in roughly 70 trials. When going through the 100 trials, the result average was 2.3400 for the attack and 6.300 for the defense. According to the graph, no team significantly took more wins than the other. However, with the averages, the defense was able to end with more armies at the end of the game as opposed to the attack.



These graphs are for the attack using three armies and defense using two. The particular simulated game that played to the end had the attack winning in roughly 25 trials. When going through the 100 trials, the result average was 0.7900 for the attack and 28.6500 for the defense. This means that the attack was not able to win a single game when the attack started with less armies than the defense.

6 Conclusion

After these simulations, the average results can show what strategy the defense should take in order to defeat the attack. For the starting number of armies, the best strategy for the defense is the same. Even though these have the same results, the reason for the strategy differs based on the attack average result.

In the scenario of 50 attack versus 50 defense, the best strategy for the defense is to send out two armies. Based on the average results of the 100 games, the defense had the lower average for sending out one army, and was never able to win a game. By sending out two armies, the average result for the defense was greater than the average result for the attack. This signifies that by sending out two armies when both sides start with 50 armies, the defense has a better chance of winning the game.

In the scenario of 50 attack versus 25 defense, the best strategy for the defense is to also send out two armies, but for a different reason than the previous scenario. No matter how many armies the defense sends out, the average result for the defense is always less than the average result for the attack. However, since the average result for the defense sending out two armies is not zero, this is the best strategy since there is a small chance that the defense is able to win a game against the attack.

Finally, the best strategy for the defense in the case of 25 attack versus 50 defense is to send out two armies. Based on the result averages, the defense always has the higher result average compared to the attack. However, the difference between averages when the defense sends out two armies is significantly higher. Since this difference is greater, this shows the defense is more probable to win the game sending out two armies compared to sending out one army.

7 Appendix

```
prob = [2275/7776, 2611/7776, 2890/7776, 0, 0;
0, 0, 0, 441/1296, 855/1296 ;
581/1296 , 420/1296, 295/1296, 0, 0;
0, 0, 0, 91/216, 125/216;
0, 0, 0, 161/216, 55/216;
0, 0, 0, 21/36, 15/36];

for k=1:100

initial_state=1;
state=initial_state;
atk = 50;
def = 25;
count = 1;
event = [];
event(count,:)=[atk,def];

while (atk > 1 && def > 1)
cum_prob_row = cumsum(prob(state,:));
x = rand;
index = find(x<cum_prob_row,1,'first');
action = index;

switch action
    case 1
        def = def - 2;
    case 2
        def = def -1;
        atk = atk -1;
    case 3
        atk = atk -2;
    case 4
        atk = atk -1;
    case 5
        def = def - 1;
end
if atk >= 3 && def >= 2
     state = 1;
end
if atk >= 3 && def == 1
    state = 2;
end
if atk == 2 && def >= 2
    state = 3;
end
if atk == 2 && def == 1
```

```
    state = 4;
end
if atk == 1 && def >= 2
    state = 5;
end
if atk == 1 && def == 1
    state = 6;
end
count = count+1;
event(count,:)=[atk,def];
end
event
result(k,:) = [atk, def];
end

figure(1);
clf;
hold on
plot(event(:,1))
plot(event(:,2),'r');
title('Number of Trials for Simulated Game')
xlabel('Number of Trials')
ylabel('Number of Armies After Trial')
hold off

figure(2);
clf;
hold on
plot(result(:,1));
plot(result(:,2),'r');
title('Running 100 trials')
xlabel('Number of Trials')
ylabel('Number of Armies Left')
hold off
```