Modeling the Attack Phase in Risk: The Board Game Using Graph Theory

Aditya Mittal[†] Stanford University

Yi Chai[‡] Stanford University

Tiyu Wang[*] Stanford University

Elie Yared[ſ] Stanford University

Qian Wang[ǂ] Stanford University

ABSTRACT

We model the attack phase in RISK: The Board Game using graph theoretic concepts and algorithms learnt in our Discrete Mathematics and Algorithms class. In RISK, the objective is world domination via one-by-one conquest of adjacent countries represented as nodes on a graph according to a complex set of rules. We use an extremely efficient version of dynamic programming (twice as fast as the regular dynamic programming setup) to compute the attacker's probabilities of winning or losing and the expected number of armies left after an attack. We also define a value function for each node based on properties of the graph which gives a metric for evaluating the worth of a certain territory at each stage of the game. We use these probabilities and the value function to define the feasibility of an attack, which we use to devise an attack phase strategy based on a greedy algorithm. This strategy is accompanied by some suggested modifications to the game to make it more interesting. The modified version of the game is then modeled and solved as a variation of the maximum flow problem.

Introduction

We observe that the board of Risk: The Board Game is a graph which we define as $G(V, E, W, C, R)$ where V is the set of vertices (the territories), the set E of edges (the connections between the territories), the set W of weights (the number of armies in a give territory), the set C of colors (representing player colors), and the regions R which is a set of subgraphs of the graph itself (the continents). In the game after each attack the graph G is updated. Each *turn* T of the game is played in 5 stages:

Stage 1: Trade Cards (optional stage, required if the player has 5 cards)

Stage 2: Compute and add reinforcements

Stage 3: Attack Stage

Stage 4: Tactical Stage: Move armies between territories

Stage 5: Take a card if a territory was conquered

Ideally, we want to define an optimal strategy for each of the 5 stages, however, in this report we will only design the optimal strategy for the attack phase. Much of the work from the attack phase can be used to compute optimal strategies for the other phases, especially the computing and adding reinforcement stage and the tactical stage. We skip the strategy for cards because it is more or less independent of the graph theoretical aspects and is simplistic. We observe that no strategy can be perfect because the attack stage has a probabilistic element to it. We define the iteration parameter for a turn T as i, for an attack A as

Aditya Mittal, Yi Chai, Tiyu Wang, Elie Yared, and Qian Wang are M.S. students in Institute of Computational and Mathematical Engineering at Stanford University.
The model for the attack phase in RISK: the Board Game was constructed for the CME305 class project offered by Amin Saberi during Winter 2008.
Contact: admittal@stanford.edu [†] yichai@stanford.edu [‡] tiyuwang@stanford.edu [*] eyared@stanford.edu [ſ] ccicci@stanford.edu [ǂ]

j, and for a player K as k so that for each iterate we can index the graph as $G_{i_{j_k}}$. So, now we go ahead and describe a greedy strategy for the attack stage, which is the simplest possibility. We iterate upon our value functions after every die roll to determine the next best option. We wrote some dynamic programming Matlab code for evaluating our value functions and probabilities of winning in attacks. In the subsequent sections we will discuss how we can strategize by looking beyond the immediate possibilities in terms of network flow and discover that some minor modifications can make the game much more interesting and finally we will also discuss some generalizations of our formulation to other applications. Appendix 1 describes the notation used throughout the paper, and Appendix 2 is the Matlab code.

Greedy Strategy for Attack Phase

So here is the general strategy we iterate upon our value functions after every die roll to determine the next best option:

- Enumerate all territories that can be attacked.
- Determine value of attacking a certain territory ,

  We define F to be the feasibility of attacking a territory d from a territory a.

  First, define $\Delta Val_c^v = Value(\{V_c + v\}) - Value(\{V_c\})$
  Where c is a player and V_C is the set of nodes that the player owns.
  So $F = \Delta Val_A^{(d)} * \Pr(a,d) + \Delta Val_D^{(d)} * \Pr(a,d) - \Delta Val_A^{(a)} * (1 - \Pr(a,d))$

  A is the offensive player, and a is the region that the attack is being launched from. D is the defensive player, d is the region being attacked.

  $Pr(a,d)$ is the probability of success of the attack and $E(w_a|A\ wins)$ is the expected number of armies that are left over given that the attack succeeds, and $E(w_d|A\ loses)$ is the expected number of armies that are left over given that the attack fails. Our Matlab program employing dynamic programming computes both the probability of success in the attack and the expected value corresponding to the winner amongst A and D.

  Calculate this feasibility for each pair of possible attacking and defending regions and attack the region d from the region a with the highest feasibility.

- Terminate the iteration if no attacks can be made legally or if F takes on a negative value.

There are two parts to the function F, a value function measuring the worth of each of the territories that a color possesses which would be used to compute the ΔVal_C^v function and $Pr(a,d)$, the probability of success of the attack. Now we define the Value function as follow:

$$Value(V_c) = PositionalValue(V_c) + ReinforcmentValue(V_c)$$
$$= StaticValue(V_c) + DynamicValue(V_c) + ReinforcementValue(V_c)$$

The $StaticValue(V_c)$ is the "intrinsic" value of the territories V_c without regard to the current setup of the armies in the territories W_{V_c}. Border territories are in this measure more valuable than inland territories. To compute the $StaticValue(V_c)$ we consider the value of the territories of color c in terms of their positions on the board. We identify that there is a special value associated with a territory that is on the "border" of a region, that is it is a node connected to two or

more regions. Recognizing that the value of these border territories is a function of the reinforcement value of the region and the number of borders the region has we compute the value of these border regions as follows:

$$StaticValue(Border\ v) = \sum_{r} Reinforcement\big(N(r,v)\big) + \frac{Reinforcement(r_v)}{|B(r)|}$$

where $N(r,v)$ defines the regions neighboring the territory v, not including the region which includes v itself. This precisely defines the distinction between "border" territories and "internal" territories because if $N(r,v) = \emptyset$ then v is an "internal" territory, otherwise it is a "border" territory. We will let this set of borders of a region be $B(r)$. Also r_v is simply the region in which the territory v lies.

After computing a value for the border territories we compute the value for "internal" territories, which are not a border by creating a continuum of values relative to the distances of the "internal" territories from the border states whose values we already computed (similar to creating a gradient on a mesh based on a boundary value problem). This is a discrete problem so we compute the value for any internal territory by determining the distance from the borders of the region and computing a weighted average based on the value of the border using this computed distance. So, we define the static value of an internal territory as:

$$StaticValue(Internal\ v) = \frac{\sum_{v_i \in B(r)} StaticValue(Border\ v_i)/Distance(v, v_i)}{|B(r)|}$$

The distance function is simply a computation of the shortest path between the internal territory and the ith border. This is easily computed using some shortest path algorithm such as Dijkstra's algorithm.

The $DynamicValue(V_c)$ is the value of a certain territory taking into account the current state of the game with the current distribution of weights and other factors. For example, a territory will be made more valuable under this measure to the player if it "completes" a region thus generating regional reinforcements for the next turn. To compute the $DynamicValue(V_c)$ we consider the fractional measure of the reinforcements over all regions in 2 ways, in terms of the number of territories controlled within a given region, and in terms of the proportion of armies controlled by the color c within the given region:

$$DynamicValue(V_c) = \sum_{r} \frac{|V_c \cap r|}{|r|} Reinforcement(r) + \sum_{r} |W_c \cap r|$$

where $|V_c \cap R_c|$ is the number of territories owned in the region and $|W_c \cap r|$ is the number of armies with a given region r. This is essentially a measure of how many reinforcements we will be able to get if we conquer the regions corresponding to the armies that exist in the given region.

Next, we want to compute the $ReinforcementValue$ which is simply the expected reinforcements the player will get at the beginning of the next turn. This one is easily computed because it is dictated by the rules of the game based on the territories controlled.

$$\begin{aligned} ReinforcmentValue(V_c) &= Reinforcment(R_c) + Reinforcment(V_c) + Reinforcment(Y_c) \\ &= Reinforcment(R_c) + \max\{\left\lfloor \frac{|V_C|}{3} \right\rfloor, 3\} + Reinforcment(Y_c) \end{aligned}$$

Here the $Reinforcement$ function is overloaded for various types of parameters such as regions, territories, or cards, defined by the game rules. Now that we know how to find the $Value$ function, we just need to be able to find the probability functions for winning attacks in order to determine the feasibility of the attacks as defined earlier. We do this using dynamic programming in the following section.

Probability of Winning an Attack and Expected Value for the Armies Remaining: Dynamic Programming

Using dynamic programming we construct a Matlab function *masterWin*, which calls a sub-function *probWin* to compute the probability of winning an attack, $\Pr(w_a, w_d)$ given the number of armies in the attacking territory, w_a, and the number of armies in the defending territory, w_d. This function also outputs the expected number of armies that will remain for the attacker if the attack succeeds, w_a', and the expected number of armies that will remain for the defender if the attack fails, w_d'. That is we can compute the $\Pr(w_a, w_d)$ and also that $E(w_a'|A\ wins)$, and also $E(w_d'|A\ loses)$.

The probabilities for winning one dice roll, i.e. one instance of iterate j, given the number of dice the attacker and defender roll are known and are given in the following table which was computed by Daniel C. Taflin in *The Probability Distribution of Risk Battles*:

			Attacker		
Defender			One die	Two Dice	Three Dice
	One Die	Attacker Wins	15/36	125/216	855/1296
		Defender Wins	21/36	91/216	441/1296
	Two Dice	Attacker Wins	55/216	295/1296	2890/7776
		Defender Wins	161/216	581/1296	2275/7776
		Both Win One	n/a	420/1296	2611/7776

Starting from the inputs w_a and w_d we compute the number of attacking and defending dice. Out assumption is that both sides uses maximum number of dices they are allowed, three for the attacker and two for the defender, if they have sufficient number of armies. Then we use the probabilities from the above table to determine the possible branches of the next state and the consequential values of w_a and w_d in the next state. We keep a matrix which is updated with each state of w_a and w_d, so we never have to re-compute a state we have already computed, which saves a lot of computation as opposed to a brute-force algorithm and sum-product over the entire probability tree structure; this gives us an efficient output which was not the case before we used dynamic programming. For example, the matrix storing $\Pr(w_a, w_d)$ is given below. We are ultimately interested in the last entry of the matrices.

$$\begin{bmatrix} \Pr(1,0) & \Pr(1,1) & \cdots & \Pr(1,w_d) \\ \Pr(2,0) & \Pr(2,1) & \cdots & \Pr(2,w_d) \\ \vdots & \vdots & \ddots & \vdots \\ \Pr(w_a,0) & \Pr(w_a,1) & \cdots & \Pr(w_a,w_d) \end{bmatrix}$$

For a naïve dynamic programming algorithm, such a matrix is built from the boundary instances fixing $w_a = 1$ or $w_d = 0$ and constructing all other entries by incrementing the indices. This will eventually lead to the solution of the entry of interest, but it is the upper bound on the performance of dynamic programming, since all entries up to the point of interest are computed. In our case, we use a recursive function that computes only the required subsequent states, starting at the point of interest. It improves performance because only the truly necessary states are computed. Looking at the dice roll table, if we assume that w_a and w_d are sufficiently larger than three, then a total of two armies must be lost after every round of dice roll. Therefore, a

state will be computed somewhere along the probability tree only if the total change in w_a and w_d compared to the initial value is even. Thus, ignoring the states where w_a and w_d are less than three, our recursive dynamic program only calculates approximately $\frac{1}{2}$ of the total number of entries in the matrices. As a result, the performance if our algorithm is doubled from that of a naïve dynamic programming algorithm.

So, now we have all the pieces we need to compute the feasibility of an attack based on the greedy algorithm described at the beginning. So now we proceed to consider some strategy by looking beyond the immediate possibilities in terms of network flow and discover that some minor modifications can make the game much more interesting, as promised earlier.

Some properties of the RISK Game and their implications:

1) The board is dense, by dense we mean well connected.
2) Infinite movement of armies is allowed between territories.
3) Infinite armies may sit in any one territory.
4) The amount of reinforcements gained by controlling a region is much greater than the amount of reinforcements gained by controlling a large group of territories.

After implementing the dynamic programming part of the project to compute the various expected values and probabilities of winning during an attack, we consider the application of network flow algorithms and path finding algorithms in defining the strategy on the risk game board. We determine based on the properties of the RISK Game mentioned above that the strategy that occurs in terms of a network flow is actually a triviality because the weights on the edges is infinite and all of the flow can be pushed through.

Consider the following scenario typical to a RISK Game: suppose there is a large army in Greenland, which is our attacking node a and there is a large army in Mexico, which is the defending node d. Assume that all nodes in between nodes a and d are controlled by 3rd parties and have arbitrary armies in them. Player A now wants to find the best way to get his armies to the defending node. The optimal strategy then is just to find the path with the least armies in between nodes a and d which can be done by a simple search because of properties 2) and 3) above. Therefore, in order to make the game more interesting we propose two modifications to the game. The first modification will limit the amount of armies that can be moved from one territory to the next (perhaps by adding terrains and limiting army motion across terrains like mountains and so forth), and the second modification will limit the number of armies that can be kept in any one territory at a time (perhaps by having population limits based on size of territory).

With these modifications in place along with the expected value and probability computations from earlier, the game becomes much more interesting because it is not immediately obvious how to do the movements between territories during the attack phase without solving a maximum flow problem. Otherwise, the armies can get stuck in intermediate territories with a high probability. We will discuss the details of solving this modified maximum flow problem in the next section. It is more complicated than a simple maximum flow problem because there are maximum capacities not only for edges, but also for nodes, and furthermore, there are probabilities and expected values for how the flow actually occurs. However, before discussing the details of the modified maximum flow problem, let us consider the implication of these modifications. Essentially,

these modifications force players to split up their armies within multiple territories rather than stocking up all the armies in a single border territory, providing for a much more interesting game.

Often the game stalls for a long time in the middle game because players are just sitting around stocking up all the armies in a single border territory, and these modifications prevent this stalling from happening. On that note, we observe that property 4) is also a contributor to the stalling that happens in the middle of the game, because players are waiting to build up a huge army that can in one turn take a region and protect it because the tradeoff between taking a bunch of territories to gain armies is weak compared to the loss of value in splitting all the armies. Of course, with our modifications there is more value in splitting the armies therefore, there is less stall in the game.

This also raises the value of solving another interesting problem within the game, which is to find Hamiltonian paths within subregions and not only for regions. Of course, we know that finding the Hamiltonian path is NP-complete but for a small graph like the RISK gameboard it can be found. This extends to larger graphs of similar nature because we can find smaller subregions on which we are able to quickly calculate the Hamiltonian path. There are other applications such as in biology and finance to which the general theory we develop applies which have been discussed in another section, which is why we discuss the case of larger graphs.

The Modified Maximum Flow problem

So, now we are going to briefly discuss the details of the flow problem that arises in this modified maximum flow problem assuming that we already know how to do the standard flow problem. We are going to approach this by doing a multistep attacking phase. After each attack we will redraw a flow on the graph and look at the residual graph to determine the next step in the attack in order to deal with the expected values and probabilities. Keeping in mind the situation described in the previous section between Greenland and Mexico we now define the two countries as the source and sink for our flow problem. The nodes have maximum capacities and so do the edges.

The complication that arises in this technique is when two countries can attack a single country along the path that is computed as the path for maximum flow. Since, once a country is conquered by the attacker moving along to get to the sink, it cannot be reconquered from another node and therefore, the maximum flow computed using the standard max flow algorithms like Ford Fulkerson can never be achieved except in the special case that the graph from the source to the sink is a tree with all the leaves of the tree connecting to the sink. So, now let's try to put a bound on the maximum achievable flow for this modified situation in comparison to the maximum flow. If the graph happens to be the special case of a tree, there is no problem as maximum flow itself can be achieved. Otherwise we use an edge removal process to find a subtree of the graph.

The edge removal process is such that we remove the worst edges so we keep the best possible flow. To be precise the edge removal process is to iterate over the set of nodes and find the nodes with more than one incoming edge. For any node with multiple incoming edges, remove all edges except the one that carries the highest flow through it as defined by a Ford Fulkerson flow. The bound then is the maximum flow minus the flow through the edges that are removed. The upper bound on the maximum flow is clear from Ford Fulkerson. The lower bound is then the maximum flow through the single path which allows maximum flow amongst all paths from a to d.

So, we have now defined a precise means of computing the flow in this more interesting modification of RISK we have designed during the process of the project, which keeps a futuristic outlook rather than just the greedy method as discussed in the first part of the paper. Still, we have only defined the attack phase strategy, and we leave the other phases of the game for future

discussion. We now discuss some other more general applications that may be modeled using a model similar to the one we built for the game of RISK. Many real life situations occur in biology and finance etc. of this sort.

Applications

More general applications of this principle:

Although RISK is inspired by military campaigns from the Napoleonic era, we can look at this problem form a broader scope and find many interesting similar models where our algorithmic approach applies:

In the economic sphere, this can be seen as a special application of game theory. It applies very well to the situation of competing franchises or companies attached to a territory. Troops will model here financial clout, and corresponding to continents will be economy of scale and monopoly revenues. These situations are observed for instance in the food retail sector with competing supermarkets, like Safeway or Walmart. Our approach allows the firm to adopt a strategy that will focus on growth and eradication of competition on a territory. It also highlights how some territories are more crucial than others, those that will be likened to the border states in RISK. The constraints we added to the rules of the game help define a better model that will take into account local parameters form territories and differences in graph topology that allows more or less easily to permeate from a region to the other.

In biology, we can liken the game modeled by RISK to competing parasites on a complex organism. Countries are then an interpretation for cells and continents for organs. Troops represent copies of the infectious agent which will get reinforced proportionally to the space it covers. Again our new set of rules help define a more precise model, and these parameters can be estimated form real data to represent the biological properties of the different agents competing within the same organism. This can also be a matrix for a model of spread of genetically modified crops in a territory. Then the different species of crop will compete for patches of land whose chemical properties will allow different patterns of reinforcement of the species and where factors of dissemination from one adjacent patch to the other will be factored in by our rule concerning limited troops movements.

From a marketing perspective, continents will represent huge clusters of customers that have similar consuming habits. Sub-territories will define subcategories of customers in the big clusters. Then the players are different exclusive products, like for instance Sony Playstation or Microsoft Xbox. The armies represent the strength of each product in advertising and capitation of the market. We remark that the continent approach is yet again interesting because it highlights the special properties of customer clusters where some regions are more permeable to outer influence than others. Furthermore he rules we added will again better represent spreading model. Different groups have different potential to be a basis for permeating other close sub-groups based on some intrinsic characteristic that can also be estimated from statistical studies.

These three generalizations show that the scope of this study can be broadened to more complex situations of competing players on a global territory. It is crucial to remark that in this model a single territory holds relatively little intrinsic value. The value of a territory is mostly determined by the expansion actions it allows for next moves. Some of the parameters we have added to make the game both more realistic and interesting help define better models in this generalization. These parameters need to be estimated to calibrate the model if we choose to use it to get some insight in one of these situations. Therefore our RISK formulation is an interesting model for quick phenomena that need to spread in a relatively short lifespan.

The strategies followed by players will differ largely depending on the graph topology of the territory. Many vulnerable "continents" will lead to conservative strategies dominated by the local analysis, where moves are determined by more immediate conquest considerations. On the other hand, too large continents will lead also to only one-step conquest where the "continent" premium will only modify the gaming on rare occasions, as a boundary condition. Only where the continents are of the proper size will players alternate frequently between local strategies and bold attempts to seize a whole contiguous territory. It seems that our modification of the game of RISK, albeit a bit complicated to grasp for beginners, is correctly calibrated so that players find themselves in that situation, making games both volatile and fun.

Appendix 1: Variables Used

$$A: attacking\ player$$

$$a: attacking\ territory$$

$$B(r): Borders\ of\ a\ region\ r$$

$$C: set\ of\ colors$$

$$c: color\ representing\ a\ given\ player$$

$$D: defending\ player$$

$$d: defending\ country$$

$$E: Set\ of\ edges$$

$$e: edge, e = \{v_1, v_2\}$$

$$G(V, E, W, C, R): Graph, defined\ at\ any\ time\ by\ G_{i_{j_k}}$$

$$i: iterate\ over\ a\ turn$$

$$j: iterate\ over\ a\ die\ roll\ from\ attacker\ and\ defender$$

$$k: iterate\ over\ a\ each\ player$$

$$N(r, v): defines\ the\ regions\ neighboring\ the\ territory\ v, excluding\ region\ containing\ v$$

$$R: Set\ of\ all\ Region\ or\ Continents$$

$$R_c: The\ regions\ controlled\ by\ color\ c$$

$$|R_c|: The\ number\ of\ regions\ controlled\ by\ color\ c$$

$$r: a\ single\ region\ or\ continent$$

$$r_v: region\ containting\ territory\ v$$

$$T: Turn\ with\ iterate\ i$$

$$V: Set\ of\ Vertices$$

$$V_c: territories\ of\ a\ given\ color$$

$$|V_c|: The\ number\ of\ territories\ controlled\ by\ color\ c$$

$$v: a\ vertex\ or\ territory, v = \{c, w\}$$

$$W: Weights\ or\ armies$$

$$|W_c|: Number\ of\ armies\ of\ color\ c$$

$$Y: Cards$$

Appendix 2: Matlab Code

ProbWin.m

```
%
% function probWin
%
% Yi Chai
% March 27, 2008
%
% for CME305 project 2008
%
% this function recursively computes the entries in the pWin and sArmy
% matrices until the element in the last row and last column is evaluated
%
% this is used as a subroutined called by masterWin function in evaluating
% the battle between an attacking territory and a defending territory in
% the board game Risk
%
% Inputs:
% a      the number of troops in the attacking territory
% b      the number of troops in the defending territory
% pWin   a matrix where the i,j element denote the probability of
%        successful attack given i attackers and j defenders
% sArmy  a matrix where the i,j element denote the expected number of
%        surviving troops after successful attack given i attackers and j
%        defenders
% sDef   a matrix where the i,j element denote the expected number of
%        surviving defender troops after the attack failed, given i
%        attackers and j defenders
%
% Outputs:
% pWin   updated probability matrix
% sArmy  updated expected number of survival troops matrix
% sDef   updated expected number of survival defender troops matrix
%
% Assumptions of rules:
% max allowed number of dices are always rolled when possible for each side
% attacker receives up to 3 dices, defender receives up to 2 dices
% the attacking territory fails if only 1 troop remains
```

```
% the defending territory fails if zero troop remains
function [pWin, sArmy, sDef] = probWin(a, d, pWin, sArmy, sDef)

% if the particular scenario (a, d) is already computed previously
% do not compute again
if (pWin(a, d + 1) >= 0)
    return;
end

% initializes the probability matrices for different dies
% probability table obtained from:
% http://en.wikipedia.org/wiki/Risk_game
A = zeros(3, 2);
B = zeros(3, 2);
C = zeros(3, 2);
%Attacker Wins
A(1,1)=15/36;
A(1,2)=55/216;
A(2,1)=125/216;
A(2,2)=295/1296;
A(3,1)=855/1296;
A(3,2)=2890/7776;
%Defender Wins
B(1,1)=21/36;
B(1,2)=161/216;
B(2,1)=91/216;
B(2,2)=581/1296;
B(3,1)=441/1296;
B(3,2)=2275/7776;
%Both Win One
C(1,1)=0;
C(1,2)=0;
C(2,1)=0;
C(2,2)=420/1296;
C(3,1)=0;
C(3,2)=2611/7776;

% compute how many dices attacker is rolling
if (a > 3)
    aDice = 3;
elseif (a == 3)
    aDice = 2;
elseif (a == 2)
    aDice = 1;
else
    % in this case, attacker loses with only 1 troop remaining
    pWin(a, d + 1) = 0;
    sArmy(a, d + 1) = 1;
    sDef(a, d + 1) = d;
    return;
end

% compute how many dices defender is rolling
if (d > 1)
    dDice = 2;
elseif (d == 1)
    dDice = 1;
```

```
else
    % in this case, defender loses with zero troop remaining
    pWin(a, d + 1) = 1;
    sArmy(a, d + 1) = a;
    sDef(a, d + 1) = d;
    return;
end

% compute 3 sub-components of the probability tree, corresponding to the 3
% cases where attcker wins dice round, defender wins dice round, or each
% side loses one army
% update the appropriate elements in the matrices recursively
[pWin, sArmy, sDef] = probWin(a, d - dDice, pWin, sArmy, sDef);
[pWin, sArmy, sDef] = probWin(max(1, a - dDice), d, pWin, sArmy, sDef);
[pWin, sArmy, sDef] = probWin(a - 1, d - 1, pWin, sArmy, sDef);

% update the matrix entries of this particular instance as a weighted
% average of the 3 sub-components, with the attacker winning probability of
% each scenario as the weights
pWin(a, d + 1) = A(aDice, dDice)*pWin(a, d - dDice + 1)...
    + B(aDice, dDice)*pWin(max(1, a - dDice), d + 1)...
    + C(aDice, dDice)*pWin(a - 1, d);
sArmy(a, d + 1) = A(aDice, dDice)*sArmy(a, d - dDice + 1)...
    + B(aDice, dDice)*sArmy(max(1, a - dDice), d + 1)...
    + C(aDice, dDice)*sArmy(a - 1, d);
sDef(a, d + 1) = A(aDice, dDice)*sDef(a, d - dDice + 1)...
    + B(aDice, dDice)*sDef(max(1, a - dDice), d + 1)...
    + C(aDice, dDice)*sDef(a - 1, d);
```

MasterWin.m

```
%
% function masterWin
%
% Yi Chai
% March 27, 2008
%
% for CME305 project 2008
%
% this function evaluates the battle between an attacking territory and a
% defending territory in the board game Risk
%
% Inputs:
% a   the number of troops in the attacking territory
% b   the number of troops in the defending territory
%
% Outputs:
% pWinA   the probability that the attack succeeds
% sArmyA  the expected number of surviving troops after successful attack
% sArmyD the expected number of surviving defence troops after failed attack
%
% Assumptions of rules:
% max allowed number of dices are always rolled when possible for each side
% attacker receives up to 3 dices, defender receives up to 2 dices
% the attacking territory fails if only 1 troop remains
% the defending territory fails if zero troop remains
function [pWinA, sArmyA, sArmyD] = masterWin(a, d)
```

```
% initializes the matrices to contain negative elements
pWin = zeros(a, d + 1) - 1;
sArmy = zeros(a, d + 1) - 1;
sDef = zeros(a, d + 1) - 1;

% give the actual work to probWin function
[pWin, sArmy, sDef] = probWin(a, d, pWin, sArmy, sDef);

% outputs the element in the last row and last column of the matrices
pWinA = pWin(a, d + 1);
sArmyA = sArmy(a, d + 1);
sArmyD = sDef(a, d + 1);
```

References:

[1] Kleinberg, Jon, and Éva Tardos. Algorithm Design. United States: Addison Wesley, 2005.

[2] "Risk (Game)." Risk (Game) Wikipedia. 30 Mar. 2008. 4 Apr. 2008 http://en.wikipedia.org/wiki/Risk_game.

[3] "RISK Strategy and Rules." Hasbro. 2007. 4 Apr. 2008 http://www.hasbro.com/risk/default.cfm?page=strategy.

[4] Taflin, Daniel C. "The Probability Distribution of Risk Battles." The Probability Distribution of Risk Battles. 25 Nov. 2001. 4 Apr. 2008 http://www.recreationalmath.com/Risk/RiskPaper.doc.

Acknowledgements

Given that we get an A+, we heartily thank Professor Amin Saberi and our TA's Adam Guetz and Chen Gu for their gracious support and teaching us all these awesome algorithms.