World Domination with Artificial Intelligence: Applying Deep Reinforcement Learning to Play the *Risk* Board Game

Oliver Krengel, Henry Chen, Edward Terry
Robotics Institute, MRSD Program
{okrengel,zihanc,eterry}@andrew.cmu.edu

1 Introduction

Implementing an AI agent to play the *Risk* board game is the goal of this project. The game is characterized by a high branching factor, multiple players and a substantial stochastic element. The game board consists of a connected graph of "territories" (nodes) that are occupied by players, as shown in Figure 1. The high-level gameplay sequence is depicted in Figure 2.

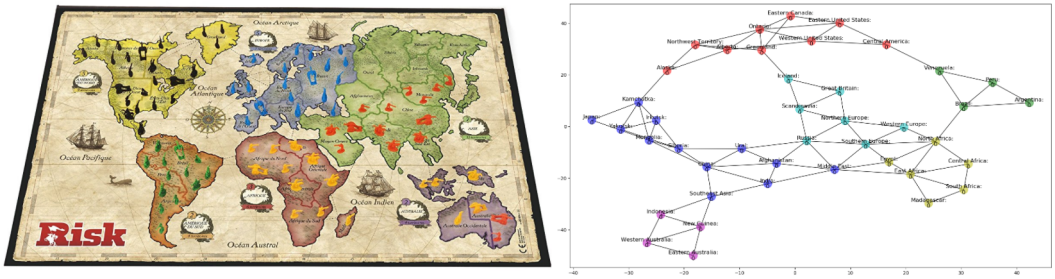

Figure 1: Physical Board Layout and Equivalent Graph Representation

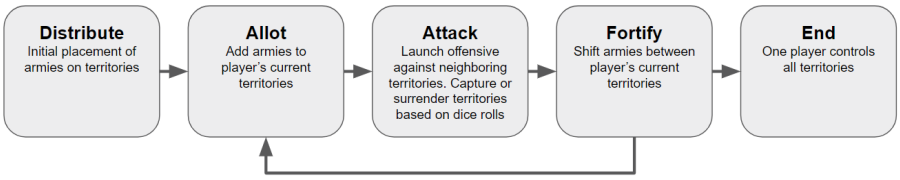

Figure 2: High-level gameplay for *Risk*

2 Prior Work

The prior work on this topic can be classified into two broad categories: accurately characterizing low-level game behavior and attempting to build complete game-playing agents.

Regarding the first category, multiple sources (Osborne, 2003, Georgiou, 2004, Lee, 2012 and Robinson, n.d.) have employed Markov Chain Analysis and Monte Carlo simulations to evaluate the expected outcomes of individual battles. Figure 3 depicts a heat map of the probabilities of attacker victory for a range of attacker and defender combinations, where red is 100% and blue is 0%. This work is a crucial contribution because these engagements are the crux of the game's stochastic behavior.

In the second category, researchers such as Olsson (2005), Wolf (2007) and Luetolf (2013) have attempted to build an AI agent to play *Risk* and in each case rely on hand-engineered features and heuristics to either to encourage the agent to adopt a particular strategy or to correct wayward behavior. For example, Olsson (2005) uses a group of agents with preferences for owning chains of territories or occupying countries with fewer borders to defend. These authors can be forgiven for taking this approach as they were working in a pre-Deep Learning era, where substantial human intervention was needed to produce acceptable behavior. Conducting our literature review leads us to believe that this is the first time that Deep Reinforcement Learning techniques are being applied to *Risk*.

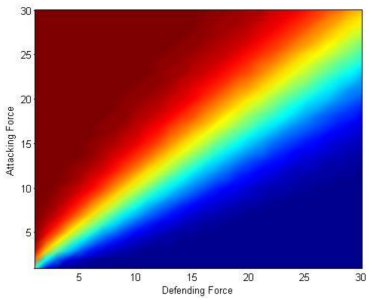

Figure 3: Attacker victory probability heat map for different combinations of attacker and defender force sizes (Robinson, n.d.)

3 Methods

3.1 Game Environment

The project was built in Python 3.5 using Tensorflow 1.6. The project has 4 central classes, governing the Player, Board, Game, and Environment. We have created a basic game environment and once we have verified its functionality we plan to implement the mechanics of the army bonus offered by holding entire continents, as this is an important strategy in the game. We do not plan to implement the dynamics of holding and trading cards, which offer a similar army bonus, but with much more unpredictable and potentially destabilizing effect.

3.2 Formulating a Markov Decision Process (MDP)

The Environment class is of particular importance. It will interpret between the game, which functions as a state machine, and a reinforcement learning agent, which sees the world as an MDP. These agents will be implemented as Q-functions. From game to agent, the environment will know which player's turn it is and query that agent for an action. Once receiving the Q-function, the Environment will apply an ϵ-*greedy-valid* policy to it, choosing the action with the maximum value that is valid in-game. It will then send that action to the game. A similar process must be applied when using TD(0) learning to update the Q-function with respect to its target. The portion of the update equation must only be applied over valid actions in s':

$$\max_{a \in A} Q(s', a')$$

3.3 Network Architecture and Training Approach

Given that there are three action phases per turn (allot, attack and fortify), each with a different action space, we will ultimately train three networks separately, with the state and action spaces defined in Table 1. We will use dueling fully-connected deep neural networks as function approximators with online TD(0) learning as specified by Wiering, et al. (2007). It will not be practical to introduce expert human gameplay as each game takes such a long time, so a self-play training model will be adopted. The approach of Keppler and Choi (2000) to reward definition is simple: provide a reward of +1 for winning the entire game and -1 for losing. Subject to experimentation, we hope to implement this approach but will also consider intermediate reward shaping. There is a fairly rich source of rewards available at each turn; one obvious candidate is the total number of armies under a player's control. While there is some precedent to using hand-crafted features, we will attempt to avoid this.

Table 1: Size of state and action vectors, T = number of territories, E = number of edges. *extra action corresponds to passing turn. **extra action corresponds to no fortification

Action	State-Space	Action-Space
Attack	T	E+1*
Allot	T	T
Fortify	T	E+1**

3.4 Reducing Complexity

The game of *Risk* presents a huge number of choices to the players, some of which are unnecessary. For instance, the work done by Robinson (n.d.) shows that for any engagement between two territories, it is always optimal to roll the maximum number of dice allowed. Since we are in the business of *winning* the game of *Risk*, we use this strategy in all cases. This allows for the action space of the attack action to be reduced to the number of edges.

3.5 Simple Gameplay Proof of Concept

We have begun the the task of learning *Risk* by training on a dramatically simplified board with only a single action type/policy in place: the attack policy. Of the three action types we will be optimizing, we have tackled this one first for three reasons: its expected behavior is well-defined; it does not rely on having access to the full game environment; and it is the most important element in the overall gameplay.

This game is similar to the full game in the following ways:

1. There are two territories, one with agent armies, one with enemy armies
2. One player acts at a time
3. The player has the option to attack or pass in each state, unless there is only 1 army remaining, when it must attack
4. When the turn begins, unless a player has the maximum number of armies, that player receives an additional army
5. Attack probabilities are dependent upon army sizes and identical to Risk

We have 3 "Risk-aware" policies built for the game of *Risk* that we will deploy here. These are used because they are policies we will use for the full game:

1. Random Attack - randomly decides to attack or pass.
2. Army Difference - always attacks when equal or greater armies than opposing player.
3. Max Difference - always attacks when expected value of armies remaining is better for player than opponent.

We can use game theory to analyze the game and determine the optimal policy. Since this is a zero-sum game, a win and loss can be represented by +1 and -1, respectively. First, we determine the values of the states if they are only allowed one action: attack for all possible attack states and pass when armies = 1. Running ten thousand trials beginning at each of the states, we have determined these in Figure 5(a). Next, since the game is symmetric, we can set the value of the state to the maximum of the attack value in Figure 5(a) and $-(passvalue)$, where pass value is the value of the state that the opponent will be in if the turn is passed. We also enforce the constraint that, since the game is symmetric, the attack value at (4,4) is negative, and this state passes the turn to an opponent in the same state, the value at this state must be 0. Finally, we show the optimal action in each state corresponding to whether the attack or pass action had higher value. This can be seen in Figure 5.

We use this optimal policy to validate the ability of an agent to learn a good attack strategy for a single engagement. We make the environment into an MDP with T defined by attack probabilities and the pass action. We represent the state as a two-element vector with -(opponent armies) in one state and (player armies) in the other. We set the reward at +1 for a win, -1 for a loss, 0 for all other states. Results can be seen in the preliminary results section.

4 Preliminary Results

4.1 Learning a linear-optimal policy

Using a linear Q-learning network, we trained the agent against an optimal policy and a random policy, with hyperparameters: $\epsilon = 0.2$, $\alpha = 0.001$, games $= 1000$, $\gamma = 0.95$. The policy that the agent learns is the best linear approximation that can be learned for the game board, and is dominant to all three of the "Risk-aware" policies. Policy results for each matchup can be seen in Table 2. Note that the hyperparameters used here are not stable and did not consistently achieve the linear-optimal policy when training.

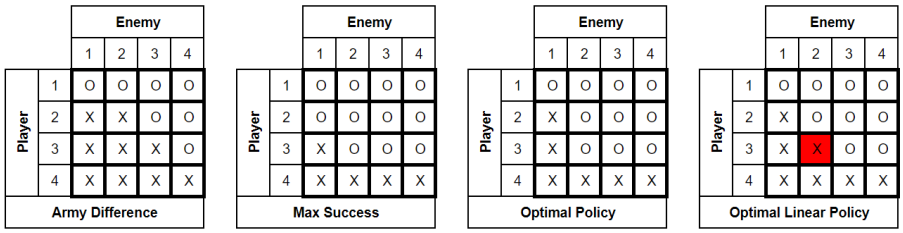

Figure 4: Policies of Different Strategies: X = attack, O = pass

```
>> simplest_game_values
Attack state values are:
   -0.2628   -0.6966   -0.8930   -0.8950
    0.2554   -0.4520   -0.7718   -0.8604
    0.6828   -0.0068   -0.3838   -0.6502
    0.8920    0.4792    0.1268   -0.2312

state values are:
   -0.2554   -0.6828   -0.8920   -0.8920
    0.2554   -0.1268   -0.4792   -0.4792
    0.6828    0.1268   -0.1268   -0.1268
    0.8920    0.4792    0.1268         0

Optimal actions are:
     0     0     0     0
     1     0     0     0
     1     0     0     0
     1     1     1     0
```

Figure 5: top-a) Value of state without pass option; middle-b) State value, maximum of attack value and pass value; bottom-c) Optimal action in each state; 1=attack, 0=pass

Table 2: A trained agent performs better than the "Risk-aware" strategies.

Agent win percentage over 1000 games

		Agent				
		Learned linear-optimal	Optimal	Max Success	Army Difference	Random
Opponent	Optimal	**48.50%**	50.00%			
	Max Success	**54.50%**	54.70%	50.00%		
	Army Difference	**54.90%**	58.60%	52.10%	50.00%	
	Random	**65.00%**	67.30%	62.90%	54.60%	50.00%

References

[1] Georgiou, H. (2004, January 2). RISK Board Game - Battle Outcome Analysis. Retrieved from c4i.gr/xgeorgio/docs/RISK-board-game_rev-3.pdf

[2] Keppler, D., & Choi, E. (2000). CS 473 - An Intelligent Agent For Risk. Retrieved from http://www.cs.cornell.edu/boom/2001sp/choi/473repo.html

[3] Lee, J. D. (2012). The Comparison of Strategies used in the game of RISK via Markovian Analysis and Monte-Carlo Simulation (Unpublished master's thesis). Air Force Institute of Technology. Retrieved from www.dtic.mil/dtic/tr/fulltext/u2/a564006.pdf

[4] Luetolf, M. (2013). A Learning AI for the game Risk using the TD(λ)-Algorithm (Unpublished master's thesis). Universitat Basel. Retrieved from http://ai.cs.unibas.ch/papers/theses/luetolf-bachelor-13.pdf

[5] Olsson, F. (2005). A Multi-Agent System for playing the board game Risk (Unpublished master's thesis). Blekinge Institute of Technology. Retrieved from https://www.diva-portal.org/smash/get/diva2:831092/FULLTEXT01.pdf

[5] Osborne, Jason A. "Markov chains for the risk board game revisited." Mathematics Magazine (2003): 129-135.

[6] Robinson, G. (n.d.). The Strategy of Risk. Retrieved from web.mit.edu/sp.268/www/risk.pdf

[7] Wiering, M. A., Patist, J., & Mannen, H. (2007, May 31). Learning to Play Board Games using Temporal Difference Methods. Retrieved from http://www.cs.uu.nl/research/techreps/repo/CS-2005/2005-048.pdf

[8] Wolf, M. (2005). An Intelligent Artificial Player for the Game of Risk (Unpublished master's thesis). Darmstadt University of Technology. Retrieved from www.ke.tu-darmstadt.de/lehre/arbeiten/diplom/2005/Wolf_Michael.pdf