A Case Study of Deep Reinforcement Learning for Risk

Luke Pratt Julia Shuieh Robert Kiesler
Georgia Institute of Technology
lpratt30@gatech.edu, jshuieh3@gatech.edu, rkiesler3@gatech.edu
https://github.gatech.edu/rkiesler3/CS7643-DL-Group-Project

Abstract

The existing AI solutions for the game Risk have been inadequate, primarily because they treated it as a zero-sum game. We make the finding that Risk is better modeled as a sequential social dilemma. We outline a multi-agent reinforcement learning (MARL) approach, and use Double DQNs to run preliminary experiments for that model. We've developed a Python-based Risk simulation using OpenAI's Gym API and introduced a novel action-space representation. Our experiments focus on the scalability issues in training, influenced by board size and opponent behavior. This work serves as a groundwork for future research in this direction.

1. Introduction

Chess and Go take a lifetime to master, whereas high levels of Risk skill can be reached in less than 50 hours. How is it then that Risk is unsurmounted by AI?

In 2005, Wolf [10] analyzed the game-tree to be *thousands of orders of magnitude* more expansive than Chess or Go. Most following research used Monte Carlo methods to do a reduced search-space look-ahead. In 2022, Ferrari et al. [3] experimented with a hybrid Monte Carlo based approach. In 2020, Blomqvist [1] applied Alpha-Zero, which uses a hybrid Monte Carlo approach. However, also in 2020, Carr [2] uses a TD λ approach without Monte Carlo. No solution has reached human level.

What prior research neglects is that a given state is only relevant to the particular set of players on the board and how they behave. It also matters how that state was reached, because those dynamics show relationships between players. Further, the popular Monte Carlo method is questionable, because the game-tree is the most intractable aspect.

Risk contains all of the challenges on the edge of the capabilities of reinforcement learning. This is a stochastic asynchronous environment with continuous action values across a multi-phase sequence of actions in a multi-agent mixed reward setting across a large state-space, where the results of actions are only known across longer time horizons. In online play, there is real-time communication.

Our model and approaches center around a new perspective to what the problem of Risk really is and making appropriate simplifications. Broadly speaking, past research did not reach human level due to treating it as a min-max optimization problem. Risk has mixed-cooperation, as players may cooperate to achieve a higher relative finishing position, and is more an extension of research in sequential social dilemmas rather than zero-sum games. [SOURCE/ELABORATE HERE]

1.1. Motivation

The fact that reinforcement learning fails to solve this "simple" problem highlights that there is opportunity for learning. For example, the state-space increases in complexity with board size for a computer, but large boards are not much more complex for a human due to the heuristics they apply. One potential area of exploration is in state-space representation with methods such as Graph Convolutional Networks and partial observability algorithms, or more abstract representations yet with feature construction and/or hierarchical approaches, and finally, attention mechanisms are relevant here.

Techniques that can be drawn from a Risk Agent can extend reinforcement learning to address more challenging cases of the broader paradigm of problems in social dynamics. For example, an approach to Risk could extend to a simulation of humans in sequential mixed competitive cooperative scenario in settings as mundane as working in the same company or as grim as fighting on the same side of a war. While a group of humans may be working together, they still have varying motivations and incentives to protect their own interests.

We found that the MARL algorithm for this set of problem, asynchronous mixed with varying reward functions, had not been developed until this year (2023) and was targeted on cooperative settings in robotics with hierarchical learning. This is the asynchronous extension of MAPPO. A major contribution of this paper could for example be the successful application of this algorithm to this new setting.

There are also aspects that haven't even been considered yet to which Risk is a suitable environment of study. For example, real-time communication would be a feature of a fully developed Risk Agent. As Risk features a multi-phase turn of varying action types, it also has potential as a research environment for planning methods. Next, as this is an environment where varied reward functions are a critical part of the functionality, this is a fertile ground for testing inverse RL methods to determine Agent motivations. Finally, there are interesting possibilities regarding the training techniques needed to train the Agents of varying reward structures, and study as to how the similarity of their reward functions impacts the training convergence.

One potential outcome of this paper is in demonstrating the value of Risk as a research environment, and releasing our version of that environment for public use. Finally, the game of Risk itself has yet to be "solved" by AI. A "solution" would be a bonus to our objectives.

1.2. Limitations

The primary limit is the massive scope of the problem. We do not plan to approach the entirety of this problem, but rather to explore a subset of the problem and to set a foundation for future research. That foundation is to be our best attempt at training a multi-agent set of Risk Agents, and to evaluate not only on a basis of performance but also on a basis of interesting dynamics between Agents.

One limitation is compute. Especially in the multi-Agent setting, we may not have the resources to rapidly train models for various experiments. A further limit is in evaluation. Ideally, we would evaluate against humans. To do this ourselves, we would need to setup a double blind study, and spend some additional weeks building such a system. We have to settle for offline evaluation, which may be done by rendering footage of the Agents playing and characterizing the behaviors, as well as evaluating their performances against hard-coded AIs and each other.

1.3. What is Risk?

In this paper, "Risk" refers to 6-player free-for-all on the classic map with fixed card bonuses. The classic map is Earth with 42 territories. Players start out owning territories across the map with random amounts of troops. A player loses if they have 0 territories.

In a turn, a player has 4 moves. At the start, they earn troops as a function of min(3, territories // 3), plus bonus troops for holding all of any of the seven continents. They may check their hand and trade in some combination of their cards to receive more troops. Next, they place their troops across their territories. They can place across any amount of territories, with 0-100% placement each.

Then, they enter an attack phase. They choose a territory to attack from, select an adjacent un-owned territory, commit 0-100% of troops to attack, and roll dice between attacker and defender. The attacker rolls 3 and the defender rolls 2. The dice are sorted and the higher dice win; defender wins ties. With equal troops, this works out to a modest attacker's advantage per Osborne [6] table 3.

The player can make as many or as few attacks as they wish. If the defender has 0 troops, the attacker takes the territory and transfers 1-all their attacking troops. If they win at least 1 territory in the phase, they are awarded a card at the end of their turn. If they eliminate a player, they win all of that players cards in turn. If they have 5 or more cards, they must trade, even mid-turn, which restarts their turn.

In the fortify phase, the Agent transfers 0-100% of its troops between two of their territories if connected by owned territories. Players can only transfer troops once per fortify phase.

2. Approach

We looked into existing simulations of Risk, but they were not complete or not well suited to interface with Python code. Much prior work was done with a Java SDK known as "Lux Delux". Instead, we simulated the game in Python to generate training data. The territories and players are represented as objects and the board is represented as a bidirectional graph of territory objects.

First, we observed that the players of Risk are the most important aspect of the environment. Every player has different motivations for playing the game and they have varying skill levels. Higher ranked Risk players are not the best calculators but the best at reading their opponents and playing around them. A human plays more for strategic positioning with respect to the *set of players they are playing against*. Prior Risk AI approaches are not well suited for play against humans because they are trained on fixed sets of fixed opponents with identical motivations.

As opposed to Chess and Go, there are not "golden moves" and it is not zero-sum. There may only be a "golden move" with respect to a given set of players after a given set of interactions between them. For this reason, traditional approaches that take advantage of compute have failed, as the compute was used for the wrong premise. An Agent that plays "optimally" may "offend" a human and receive retaliation. In other words, there is some tit-for-tat. Further, a key aspect of the game is that the players sometimes cooperate and sometimes defect. The "vindictiveness" or aggressiveness of a given player is highly variable. We have not observed any existing Risk AI, hard-coded or not, that has considered this factor. Our thesis is that inclusion of considering the nature of the other players on the board will be a primary driver of performance.

How is that done? When playing the game, humans evaluate the motivations and skill levels of other players. Since we don't have real human data, we have the requirement that human behavior must be simulated. So to approach the problem of Risk, we have the interesting sub-problem of simulating a dynamic human interaction. Here, we take the strategy of treating this as a multi-Agent system under reinforcement learning.

What followed from this model was a literature review in existing MARL algorithms. MARL generally assumes Agents act synchronously, while in Risk the Agents take turns. During review, we found that an algorithm for asynchronous MARL has not been created until 2023, and even this specific algorithm may not be suitable. Much of the research on MARL is on cooperative behavior in robotics, while Risk is a mixed reward system where Agents have incentive to defect as much as to cooperate. Hierarchical approaches from cooperative robotics therefore may not be suitable as Risk Agents don't have the motivation to maximize a joint action value. While the asynchronous nature of Risk may very well suit it better to independent RL techniques, this does not address the non-staitionarity of the environment.

Our initial plan was to train multiple Agents of different "personalities". The idea was to define reward structures that emulated the archetypes of real human players, and to train in sets of pseudo-random compositions of these Agents. The Agent would make heuristic observations of the way other Agents behave and learn to infer how to best play around a given player type. For example, an aggression and passivity index could be maintained. Being that the Agent would start out not having an observation of any of the players, it would also have to learn to make decisions based on limited information while it observes other players.

There are some possibilities of using Agents with similar reward functions to act as pre-trained initializations of other variations. Also, our case of mixed rewards for multi-Agent is not unique. Some existing multi-agent algorithms can handle mixed rewards structures, such as MADDPG and MAPPO as reviewed by Lee et al. [4]. It is also worth noting that MARL is not mandatory; bots could be hard-coded to have such behaviors, which a single Agent can train on.

We previously claimed that it is important how a given iteration is reached; tracking heuristic observations of interactions between players may allow the Agent to better honor the Markov property. Although we cannot create a perfect Markovian representation by doing so, prior work has neglected the variations in the behavior of players, and they trained for the least Markovian case because they don't observe one of the most important aspects of the environment.

Most Markov Decision Processes in truth are Partially-Observable MDPs, because it is unlikely that a representation of a state we supply an Agent is a full observation of the ground truth. Spaan [8] reviews POMDPs and gives the example of imperfect sensors in robots. The imperfect sensor may lead the Agent to believe it is in one state, when in truth it is in another, and to select the wrong action. If we give our Agent an imperfect heuristic, it may believe it is playing against one type of player, when it is playing another.

Per Spaan, in some cases this partial observability can be ignored, and in others it deteriorates performance. They highlight techniques for dealing with POMDPs such as maintaining a probabilistic belief state. Here, we posit that the better we can design the heuristics for the Agent to observe, the less we violate the Markov property. Or, the less valuable the heuristics we provide, the more the Agent may learn to ignore our heuristics by receiving negative rewards.

The Risk environment was more complex than expected to program and debug, with 2100 lines of code. We have not yet had time to train the multi-Agent case. Instead, we train DDQN agents and focus our analysis on the impact of board size and AI opponent type on training. We use it to explore the challenge areas as a basis for our future work with MARL.

2.1. Evaluation

The most clear metric is average finishing rank against human players. As there are 6 players, this is a value between 1 and 6. An ideal scenario to measure this is anonymous play online. The difficulty is that without another system to automate our Agent to interact with Risk online, or without collaboration with game developers, we do not have the means to do so.

The data collection process, without being able to automate online play, would be intensive and time consuming, requiring many humans and a double blind study. Therefore, we are forced to evaluate offline. We use the same Agents we developed for simulating group dynamics to measure the effectiveness of the Agent. However, we note that this is not a true measure of success, as our simulation will not be as accurate as evaluating on human play.

Rank against humans would itself also not be enough to declare success. In play against humans, there are strategies that do not involve engaging in dynamic or interesting behaviors to achieve a performance higher than that of the average human. For example, a player can play as passively as possible; taking a card and passing every turn. They won not by greater skill but by valuing "winning" in too absolute of a sense and not valuing their time. Such a player is implicitly playing a different game.

Therefore, we also render gameplay from the Agent and observe the behaviors it engages in qualitatively as a measure of success. We setup a few quantitative measures for this purpose as well, such as average attacks made per turn and the standard deviation of attacks made per turn. The purpose is that either the average or the standard deviation would likely be higher in a more dynamic play-style.

2.2. Environment

We used networkx to create the graph representation. We first implemented essential functions to simulate the game and tested that we can run a game with random moves. Then, we translated this into a Gym environment. We use Gym as it the standard API for reinforcement learning environments and it provides flexibility to use external libraries, but we don't make use of that.

We represent Risk as a problem of troop distribution. For example, in the placement phase, the Agent selects territories TO place a positive amount of troops on. In the attack phase, the Agent selects a territory to attack FROM and TO, with a stochastic exchange of troops occurring between FROM and TO. The Agent may also choose to end the attack turn. In the fortify phase, the Agent selects a territory to remove X troops FROM, and a contiguously connected territory TO place those positive X troops. Therefore, the Agent has a number of actions equal to the number of territories, with an additional skip action. To avoid a continuous action space, we apply the heuristic that the Agent always moves 100% of its troops.

We then had to decide what to include in the state space. We needed a Markovian state while minimizing complexity. We include the binary toggles of the current phase, the normalized troop count in all territories, and the previous action. The previous action is critical because the Agent must know what it selected in the FROM phase if it is in a TO phase. The normalized troop count is a positive number for territories owned by the agent and a negative number for territories owned by any other player.

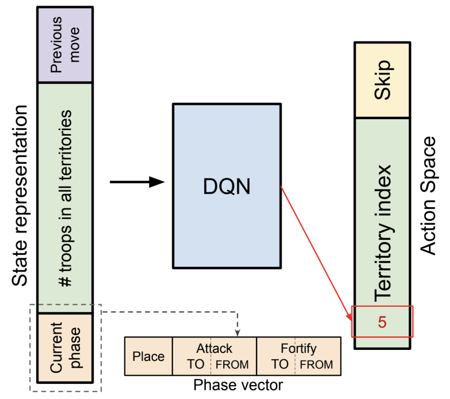

Figure 1. State and action representation used to train DDQN.

We made the choice to hard-code the logic of when the Agent decides to trade cards, as it didn't fit into the TO/FROM model. The agent would only trade when it was forced to, and it trades for the highest possible number of troops. We didn't give the Agent the opportunity to "stack the deck" with cards as a strategy. In "Risk: Global Domination", which is the aim of human-play for this project, the popular dice rolling mode is "balanced-blitz" which is an involved calculation and essentially has the effect of reduced kurtosis. Here, we use no-frills dice rolls.

We found that the original classic map was difficult for the agent. We created smaller map sizes to verify we can train with our design. The smallest map included only four territories, the medium map included all territories in North America, and the larger map included all territories in North and South America. The small map should be very easy for the agent to learn how to win, while increasing the size increases the complexity of the problem and should give insight on the agent's generalizability and learning capabilities. Figure 2 shows an initialization of the large map.

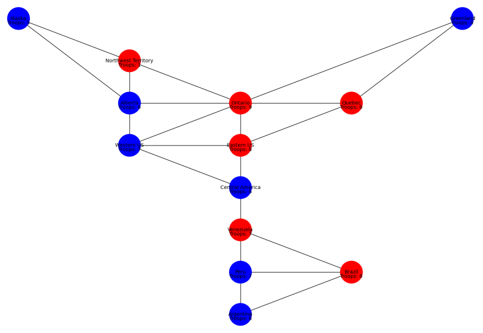

Figure 2. Graph representation of large map used in experiments.

We implemented two bots for the agent to play against to observe how the agent learns. The neutral bot remains passive through the game, with it only placing troops and never attacking. Our expectation was that the agent would just need to learn how to make legal actions in the environment. The pseudo-random bot also attacks and fortifies. Since the bot attacks, the Agent must learn that there is an attacker's advantage in Risk and to use early-game aggression, because the aggressive player will win in a 1v1 scenario. It should be more challenging, but not that difficult.

2.3. Agent

We model our Agents as a DDQN. In DQNs, a neural network works acts as a function approximator to estimate the Q-value. The Q-Value is the expected outcome for taking an action in a given state. The network estimates the value of a state with one step TD- λ bootstrapping updates, which are weighted by γ. γ can interpreted as the trade between how much we value reward now vs later, or it can be argued to be some confidence for how much the Agent believes its policy will perform as expected long-term. The policy is to follow the maximum estimated action.

DDQNs maintain a copy of the network to use as a target value for the update step. It updates that copy by fraction τ on some frequency such as every 500 steps. This is shown to benefit training stability by Van et al. [9] in 2016. We also make use of a replay buffer. Because the Agent gets experience in a correlated manner with the policy it has, it can fall into a local-optima. Random sampling from a replay buffer better honors iid, shown by Mnih et al. [5] in using a DQN to play Atari.

2.4. Reward Function

Because 1vs1 is zero-sum, the Agent is rewarded a small bonus multiplied by how many more territories it holds than the bot. If the Agent selects an illegal move, it is given a penalty and does not advance its phase. The Agent gets a large reward for winning and that same penalty for losing. To encourage a game to end, if a certain number of turns have passed, we start applying a negative reward that scales by how many turns have passed. As we shall discuss, the time penalty was problematic when combined with one of our training parameters.

2.5. Training

In each experiment we used a simple neural network with 5 fully connected layers and ReLU activation. The random bot experiment used hidden layer sizes of 512, 512, 256, 128, 128. The neutral bot experiment used 512, 256, 128, 128, 128. We optimize on an interval that is a modulus of batch size to actions, and then do X optimizations. We tuned hyperparameters for the largest map with manual tuning, and then used the same values for the smaller maps. Shown below are the hyperparameters we use in section 3.1 and 3.2.

	Neutral	Random
α	0.005	5e-5
γ	0.95	.99
ϵ decay	Linear	Sinusoidal
ϵ range	0.1 - 1	0 - 1
ϵ Oscillations	0	20
τ	0.05	0.001
τ update interval	100	1000
Loss	MSE	MSE
Max actions	2000	1200
Episodes	3500	3500
Batch Size	64	64
Optimizations/modulus	2	64
Replay Buffer	100,000	50,000

Table 1. Hyperparameters for sections 3.1 and 3.2

Sinusoidal epsilon explores more variety for each skill level of the Agent, while linear spends more time learning at each epsilon per skill level. The effect of their difference depends on the nature of the learning environment which we can only arbitrarily qualify. Because Random does more optimizations per step, it also uses lower α, τ, and update frequency. One possible update for the sinusoidal parameters is for the memory size to equal wavelength.

We track many metrics: Loss (average/episode), average reward (per action), cumulative reward (per episode), the ratio of illegal moves to legal moves, how many turns an episodes lasts, and the time it takes for an episode to complete. We also log min, median, and max action counts. For results, we focus on the cumulative reward, the illegal move ratio, and how often the Agent wins in validation. We report the highest cumulative reward checkpoint.

3. Experiments and Results

3.1. Neutral Bot

	Small	Medium	Large
Total reward	-2186.44	-6894.45	-9050.20
% Illegal Moves	19.82%	52.13%	76.09%
Win ratio	78%	4%	0%

Table 2. Evaluation results of agent trained against Neutral bot averaged over 50 trials.

As previously mentioned, the Neutral bot will only place troops down, and will not attack or fortify. However, we observed that the agent struggled to grasp the concept of legal and illegal moves.

During our experiment with the small map, the agent showed relatively promising results, displaying an ability to learn effectively and win the game. However, the agent became stuck in a loop of repeatedly making illegal moves in certain configurations where the agent's attacks fail and all troop counts are reduced to only one, rendering it incapable of launching any attacks. Note that the loss is noisy, but the episodes are much shorter than the larger map we tuned for, and so it is training on less data per episode.

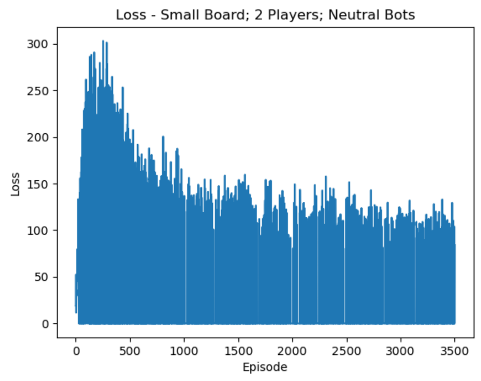

Figure 3. Agent loss on small map with Neutral bot.

The challenges further intensified for our experiments with the medium and large map sizes. As the map size increased, the number of possible game states expanded exponentially. The agent's performance deteriorated, with a marked increase in the frequency of engaging in illegal moves. We observed that the ratio of illegal moves increased over training for both the medium and large map. Out of all evaluation configurations, only a few will lead to the agent winning. We believe this behavior comes from the agent learning that attacking is the best strategy, but fails to learn that it must skip when it cannot attack anymore.

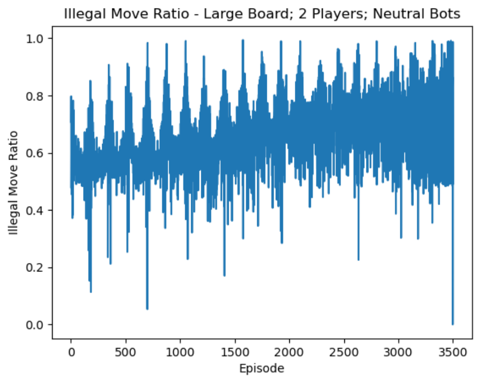

Figure 4. Ratio of illegal moves by agent on large map with Neutral bot.

We attempted to resolve this issue by increasing the illegal move penalty, trying geometric and oscillating epsilon decay, and modifying the model capacity of the DDQN. However, despite any changes, the agent could not learn to not perform any illegal moves. The agent continued to get stuck and repeat illegal moves for all map sizes.

3.2. Random Bot

	Small	Medium	Large	Large*
Total reward	316.12	-604.78	-652.8	-3270.0
% Illegal Moves	4.7%	19.4%	15.3%	66.1%
Win ratio	58%	0%	0%	0%

Table 3. Evaluation results of agent trained against Random bot averaged over 50 trials.

While we expected the pseudo-random bot to be more challenging, it had benefits in training. Because 1 vs 1 is a zero sum setting, the Agent needs to learn early-game aggression. If the bot isn't lethal at all (neutral), it may build up many troops and make it difficult for the Agent to find the complex chain of actions required to take the board.

While programming the "random" bot, we added a few features to make it a little more skilled. The placement phase is truly random; it selects a territory it owns and adds all of its troops. However, as we know this territory has troops in it, this one is selected to be the attacking territory. That attacking territory makes a list of neighboring enemy territories and sorts by troop count. It takes a look at the least defended neighbor, and if it has troop advantage and enough troops to roll 3 dice, it makes the attack. That process continues between a random integer minimum of 0 and maximum of 3 attacks, until the bot loses an attack or doesn't find a troop advantage. Finally, for the fortify phase, it looks for its neighboring owned territory that has the most troops, and then concentrates its troops into itself.

This bot's logic works well early game when the board is randomly divided, but the logic of troop concentration starts to fail late game. To compensate, the number of troops generated is increased to 1:1 with the number of territories owned. For example, if the Agent turtles in a single territory in a 9 territory map, the bot will generate 8x troops. Looking back, a more effective troop generation scheme with this bot would be non-linearly increasing with territories, because with the random placement of troops and a 1:1 relationship of generation and territories, a single Agent territory can defend. While the bot generates 8x the troops, it distributes randomly over 8x the territories.

The first attempt (not shown) had a loss divergence. This was remedied by using gradient clipping and lowering the soft update frequency. Gradient clipping as a technique is outlined by Schaul et al. in 2015 [7]. After updating our parameters to table 1 we get the loss curve in Figure 5.

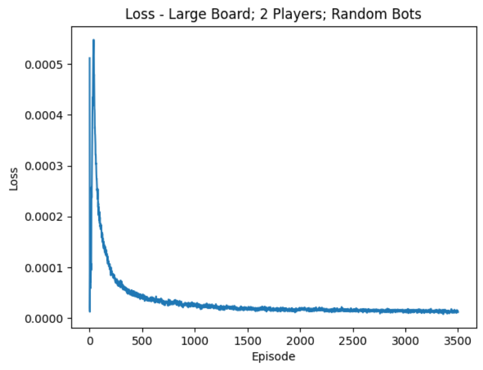

Figure 5. Loss Curve vs Random Bot on Large Map. The majority of learning happens in 2000 episodes.

One challenge of the exploration-exploitation process is that it is very frequently favorable for the Agent to just skip its move instead of try a move that may be illegal, because there are more illegal moves than legal, and we apply a penalty when it makes an illegal move. As shown in Figure 6, the Agent does learn quite well to not make illegal moves. Unfortunately, this is at the cost of over-fitting to skip its turns.

This is amplified by the penalty for taking too long to end an episode. If the Agent is many turns into an episode, it will find that the penalty of illegal moves is lower than making a legal move. One idea we had that we didn't have time to implement was to limit the amount of skips an Agent was allowed to having during an episode, "vetoing" by thereafter selecting the 2nd best estimated move. Also, the oscillating epsilon we used for these trials may encourage it to be more conservative; more experiments are needed with the behavior of epsilon. Finally, the start of the (1 vs 1) game is the most relevant, and real games of this size take far less actions, so we could try more episodes of shorter max actions so it spends less time in unrealistic states.

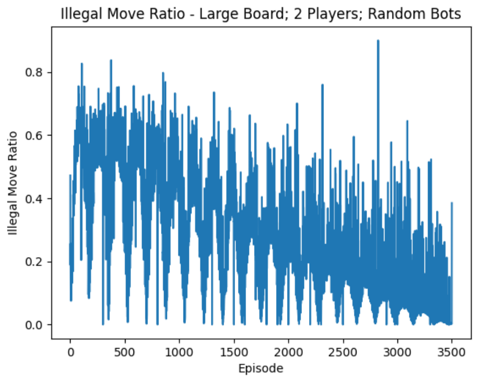

Figure 6. Ratio of illegal moves by agent on large map with Random bot.

Our default scheme is for the Agent to be placed into the same phase repeatedly until it makes a legal move, while giving it negative rewards for each illegal move. Another trial consisted of treating illegal moves by the Agent as skip-moves, and to have this only work against the favor of the Agent. The reward structure is updated to be purely zero-sum with no illegal or time penalty. If the Agent skips a phase by making an illegal move, it only puts the Agent at a disadvantageous position. For example, in the placement phase, the original rules of Risk do not allow skipping. We say, why not, if you don't want to place your troops legally, no problem, your turn is skipped and your placeable troops are set to zero. The rewards received as shown below indicates the Agent may have more to learn under this structure.

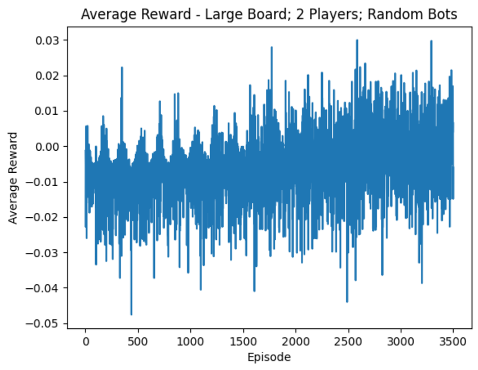

Figure 7. Average reward per action on large map with Random bot, using modified reward structure.

The evaluation results are reported as "Large*" in the table at the top of this section, where the rewards however are evaluated under the original structure. Note that it makes many more illegal moves, as we would expect.

All of the above results are shown for the large environment. The medium environment had similar loss curves. This is because we do not allow an episode to last for more than 1200 actions and both hit that always. So, the amount the Agent may learn is rate limited. Interestingly, as shown in the table, the medium environment made more illegal moves and had a higher positive reward than the large environment. This indicates the Agent had less incentive to exploit the reward function due to the smaller environment.

The results of the small environment show the Agent rapidly learning to play the game. These trials are much faster to train because the Agent needs nowhere near 1200 actions to complete an episode, it is closer to the order of 50. So, the curves are much noisier due to training on less data per episode. We see the same oscillations due to sinusoidal epsilon. Because there are few illegal moves, and because the game ends quickly, it doesn't learn to exploit the reward function to perpetually skip its turn.

4. Discussion and Future Work

We faced challenges because of illegal moves and the Agent over-fitting to skip them. If we don't penalize them, the Agent takes much longer to learn to not do them. If we penalize too harshly, the Agent overfits to skip its turn. The last experiment in section 3.2 showed that alternative reward structures may solve this. We will also explore alternative action-space representations, or placing limits on the amount of moves the Agent may skip in an episode so that it doesn't only gain experience of skipping.

As noted in the training sessions against the bots, slight differences in opponent behavior dramatically alters the learning process. We may benefit from self-play training, or some form of curricula such as training for simpler settings and progressively increasing complexity.

We also plan on doing more work with the state-space representation. Larger boards are complex because the Agent understands them literally, but they are not more complex for humans because the conceptual meaning is the same. Initial plans are discretizing continuous values and trying the graph convolutional network used by Carr [2].

We will continue with the original thesis of multi-Agent handling of Risk. As noted in the approach section, Yu et al. [11] have developed an asynchronous version of MAPPO that may be suitable for this use case. If this is too computationally demanding, we can restrict it to smaller board sizes such as we used in this work. There exist smaller boards in online play, which would be sufficient to train on and then evaluate against humans. Although, too small and more complex player interactions can't be modeled.

Finally, we can improve on the efficiency of our environment and model training. Our longest training sessions were 8 hours on personal desktops, and the training times would only increase with more complex algorithms.

References

[1] Erik Blomqvist. Playing the game of risk with an alphazero agent, 2020. 1

[2] Jamie Carr. Using graph convolutional networks and td lambda to play the game of risk. *arXiv preprint arXiv:2009.06355*, 2020. 1, 8

[3] René G Ferrari and Joaquim VC Assunção. Towards playing risk with a hybrid monte carlo based agent. In *Anais Estendidos do XXI Simpósio Brasileiro de Jogos e Entretenimento Digital*, pages 301–306. SBC, 2022. 1

[4] Ken Ming Lee, Sriram Ganapathi Subramanian, and Mark Crowley. Investigation of independent reinforcement learning algorithms in multi-agent environments. *Frontiers in Artificial Intelligence*, page 211, 2022. 3

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 5

[6] Jason A Osborne. Markov chains for the risk board game revisited. *Mathematics magazine*, 76(2):129–135, 2003. 2

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 6

[8] Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement learning: State-of-the-art*, pages 387–414. Springer, 2012. 3

[9] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 5

[10] Michael Wolf. An intelligent artificial player for the game of risk. *Unpublished doctoral dissertation). TU Darmstadt, Knowledge Engineering Group, Darmstadt Germany. http://www. ke. tu-darmstadt. de/bibtex/topics/single/33*, 2005. 1

[11] Chao Yu, Xinyi Yang, Jiaxuan Gao, Jiayu Chen, Yunfei Li, Jijia Liu, Yunfei Xiang, Ruixin Huang, Huazhong Yang, Yi Wu, et al. Asynchronous multi-agent reinforcement learning for efficient real-time multi-robot cooperative exploration. *arXiv preprint arXiv:2301.03398*, 2023. 8

Student Name	Contributed Aspects	Details
Luke Pratt	Environment, Training, and Analysis	Built the object-oriented representation of the board. Trained the Agent against Random bot, resolved issues with loss divergence with gradient clipping and lower soft update frequency. Designed the "From/To" representation and the reward function. Updated the Agent's memory from a Deque into an array for O(1) sampling and replacement. Provided domain knowledge as a master ranked player. Updated the DDQN to implement tau soft update on an interval. Implemented and tested many of the primitives including vectorized dice rolling for 2.5x speedup. Worked on environment testing and debugging. Implemented the Random and Neutral bots. Inspired by Julia's notebook, created a notebook to test and showcase bot functionality. Modularized the environment from a monolithic function to handler functions. Formalized the approach, analyzed Random bot, and did literature review regarding MARL, POMDPs and prior Risk AI solutions. Code: atomic_actions.py (80%); board.py (75%); actors.py (100%); simulation.py (33%); env.py (30%); train.py (15%); test_bots.ipynb (100%); test_env.ipynb (15%);
Julia Shuieh	Environment, Training, Analysis	Implemented troop delegation, territory cards, placement phase, and fortify phase for simulation. Adapted Risk and implemented Gym environment to use for training agent. Created notebook to test environment before training. Created boards for different sizes to use in experiments. Adapted training pipeline to use configuration file for easier experimentation with different hyperparameter values. Added output graphs and finished evaluation. Trained DDQN agent against Neutral bot and investigated issues with learning legal moves. Described and analyzed work in report, particularly for the environment and Neutral bot experiments. Code: atomic_actions.py (20%); board.py (25%); env.py (40%); simulation.py (33%); test_env.ipynb (85%); train.py (70%)
Robert Kiesler	Environment, Implementation, and Analysis	Explored and analyzed Lux (a similar scope to our project) to glean ideas and spent a considerable amount of time trying to (unsuccessfully) get it running. Helped start the simulation and training pieces of the code. Helped establish a DQN. Experimented with a human vs. bot off-spin of the project. Code: simulation.py (33%); env.py (30%); train.py (15%);
Ashish Panchal	Reinforcement Learning Theory	Dropped for personal reasons. Popped in ocassionally and offered interesting discussion around RL theory.

Table 4. Contributions of team members.