Reinforcement Learning in the Game of RISK

Lars Quaedvlieg LCPM.QUAEDVLIEG@STUDENT.MAASTRICHTUNIVERSITY.NL
Ramon Reszat R.RESZAT@STUDENT.MAASTRICHTUNIVERSITY.NL
Joan Botzev J.BOTZEV@STUDENT.MAASTRICHTUNIVERSITY.NL
Daniel Roder DANIEL.RODER@STUDENT.MAASTRICHTUNIVERSITY.NL
Michael Balzer M.BALZER@STUDENT.MAASTRICHTUNIVERSITY.NL
Sree Kotala S.KOTALA@STUDENT.MAASTRICHTUNIVERSITY.NL
Department of Data Science and Knowledge Engineering, Maastricht University

Abstract

In this paper artificially intelligent players have been developed capable of playing the game of RISK. Multiple algorithms have been implemented using Temporal-Difference and Q-Learning. Both of these algorithms have been trained by self-play. This study serves as a comparison between linear and non-linear evaluation functions. In the final experiments the non-linear approach performed better than both the random baseline as well as the linear function with the selected weights.

Keywords: RISK, Temporal Difference Learning, Deep Q-Learning, Self-Play, Linearity

1. Introduction

The field of Reinforcement Learning and games have gone a long way together. On the one hand, games allow for many meaningful testing applications in the field of Reinforcement Learning [1]. On the other hand, games can make use of these algorithms to improve the overall experience of players.

Multiple aspects of the games can differ, including but not limited to the environment and transitions within it, actions and the goal of the game. RISK is considered difficult to master in that sense. This is due to partially observable environments, stochastic state transitions and the game being multi-player. It has an infinite state-space complexity and a game-tree complexity from 10^{2350} to 10^{5945} [2].

RISK is a strategy game; The goal is to own all countries in the world by defeating opponents. The game, which can be played with 2 to 6 players, consists of two game phases which will be referred to as the *distribution* phase and the *battle* phase. In the distribution phase, players will choose on which countries to initially place their troops until they have no more to place. Then, the battle phase starts. Each turn in the battle phase consists of three sub-phases. These are called the *placement*, *attack* and *fortifying* phases. They are consecutively meant for placing new troops, attempting to invade other countries and moving troops between countries.

RISK is a niche game for benchmarking reinforcement learning algorithms. However its probabilistic elements have been evaluated in detail in papers such as [3] and [4]. Furthermore heuristic strategies have been proposed by [5] and [6] to construct valuable features that describe the current state of the game board.

The goal of this paper is to argue whether it is possible to implement a bot that can play the game of Risk. Furthermore, it is interesting to compare the performance of linear versus non-linear function approximators for (action-)value function approximation.

This paper extends the idea of using temporal-difference learning [7] with Q-Learning to find an approximation of the optimal policy in the game of RISK.

In section 3.1 features that describe the state of the game are discussed, heuristic algorithms for the distribution- and placement phase are proposed and finally the Reinforcement Learning approaches for the battle phase will be considered. Finally, experiments will be performed and the results will be discussed.

2. Graphical User Interface

The UI is built using JavaFX and CSS. The GUI consists of main menu, rules page, player selection screen with 6 options, pause menu and the actual game map itself. The game map consists of SVGs and coupled with in-game features provide a delightful user experience to aid in playing the game. These are discussed in more detail at appendix A.

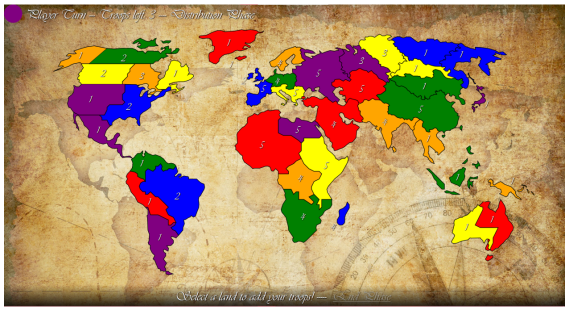
Player Turn – Troops left: 3 — Distribution Phase
Select a land to add your troops! — End Phase

Figure 1: RISK board-map with 6 players, showcases the SVG paths and features described.

3. Methods

3.1 Features

Both bots use five distinct game features to determine the value of each state. These features are heavily influenced by the papers of [5] and [6]. The features chosen are the following:

- **Armies feature**: Proportion of army strength in comparison to all other armies on the board.
- **Territories feature**: Proportion of territories controlled by the player
- **Enemy Reinforcement feature**: Sum of expected reinforcement of enemy players
- **Best Enemy feature**: Negative strength measure for the strongest enemy player
- **Hinterland feature**: Proportion of countries owned that are not adjacent to enemy countries

3.2 Heuristic Algorithms

Three heuristic algorithms have been implemented. One for the distribution phase, one for the placement phase and one for the fortifying phase. All algorithms depend on the difference in troops between each country and its attacking neighbors. The heuristic algorithm for the fortifying phase is based on the idea of reinforcing those borders with the highest presence of enemy troops. For the distribution heuristic, a table is created with priorities based on the number of players. The table can be found back in the appendix 11.

3.3 Temporal Difference Learning

Temporal Difference Learning ($TD(\lambda)$) has been chosen as one of the bots in order to not having to rely on given data sets [8]. Since TD-Learning relies on the difference between the current estimate of a state value and its actual state value for a given state, predefined data can be replaced by some sort of evaluation function, hence the name "Temporal Difference". For the specific environment of the implanted RISK board game, the evaluation function has been chosen to be linear, in order to use the Temporal Difference formulae to evaluate a given state based on its features. For the approach of this paper, a feature space of five distinct features, mentioned above, has been chosen. The Temporal Difference bot will only engage in an actual attack phase, the rest is solved by heuristics described in 2.2. The features are collected in a linear function, presented by Sutton.[7]
.

$$w_{i,t+1} = w_{i,t} + \alpha \times \big(F(x_{t+1}) - F(x_t)\big) \times \sum_{k=1}^{t} \lambda^{t-k} f_i(x_k) \tag{1}$$

The formula seen above is used during the TD-Learning approach to adjust the weights inside the linear function to evaluate future states based on predicted features. The calculation involves subtracting the current state value from an estimated future state. This is then multiplied with the sum of all values a specific feature took in the past steps, multiplied by a regularization factor λ, taken to the power $t-k$, where $t-k$ counts the rounds that were taken up to time t, backwards. The distinct feature is the one matching to the weight that is to be updated. The resulting product is multiplied with a learning parameter alpha and added to the current weight. This step is executed for all weights simultaneously to ensure that all states are calculated with most recent weights.

In this iteration of the TD-Bot, it has been decided to update the weights of the linear evaluation function after each attack move allowing for a faster learning experience while on the other hand risking inhomogeneous learning steps. [5]

3.3.1 Finding the future state value

In the Battle phase each attack independent of its outcome will lead the agent to a new state therefore implying a new calculation of the state value. Depending on the current board situation the agent might encounter a large number of possible attack moves. To decide which of these moves or rather which of the state values resulting from these moves $\mathrm{F(x_{t+1})}$ shall be used for the TD-learning algorithm we introduced a method which allows the agent to estimate the future state values for all its possible attack moves of which he will subsequently choose the one with the highest expected state value. These values are calculated by estimates like the expected troop loss and result in a simulated state value which is in turn weighted by the estimated win-chance.

3.3.2 Deciding to attack

While the previously mentioned method allows the TD-Bot to choose an attack target it lacks the ability of actually allowing to let the bot decide whether to attack or not. For this reason an additional heuristic which decides based on the win chance if the bot should start an attack or not was introduced. For this reason two hyper- parameters where introduced which allow manual adjustments to the threshold set for the win chance and a hyper-parameter defining the probability that the bot will deviate from this threshold and engage in a less favorable fight.

3.3.3 Normalization

The results suggest a similar behaviour of the TD-Algorithm as was indicated by [5]. Thus games with a large number of steps tend to reach a point where the weights suddenly jump resulting in an ever rising state value. This effect is amplified by the fact that in the current iteration the bot learns after each attack move. To counteract this problem the following normalization method is introduced, also indicated by [5].

$$w_{i,t+1} = w_{i,t} + \alpha \times (F(x_{t+1}) - F(x_t)) \times \frac{\sum_{k=1}^{t} \lambda^{t-k} f_i(x_k)}{||f(x_t)|| \times ||\sum_{k=1}^{t} \lambda^{t-k} f(x_k)||} \tag{2}$$

Here $\mathrm{f(x_t)}$ is the vector of all features for the current state. Therefore the denominator is based on the values of all features while the numerator is only based on the feature corresponding to the weight that is currently updated. Furthermore $||x||$ is used to denote the euclidean norm. As it can be seen by graphs in the appendix (C), normalized weights will prevent a divergence of said values, such that their impact will be kept moderate over the course of the training which would otherwise lead to massively increasing and decreasing state values.

3.4 Deep Reinforcement Learning

Deep Reinforcement Learning is a natural addition to Temporal Difference learning, which uses a non-linear evaluation function to approximate state-action (Q) values. The task involves taking a binary attack decision that maximizes the expected future rewards. Hence the RISK bot is choosing actions in the battle phase that increase the number of countries owned by it while avoiding loosing troops over the course of the game to eventually achieve its goal of winning.

3.4.1 Deep Q-Network

We estimate the discounted action-state values and the expected future reward $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ of the state after taking an action using a multi-layered deep Q network [9].

3.4.2 Deep Q-Learning

Stochastic gradient descent is not good for single samples in online training. Thus using non-linear approximation for the Q-values will almost never converge. Deep Q-Learning aims to address this. In order to make policy iteration more stable, two improvements have been made. These go by the names of *Experience Replay* [10] and *Fixed Q-Targets*, as proposed in [9]. As said before, standard Q-Learning uses experience it obtains only once. This is a waste, since re-iterating over this experience might stabilize the policy evaluation process. The idea of experience replay is to re-use previous experience. At each time-step t, the information $S_t, A_t, R_{t+1}, S_{t+1}$ is stored in replay experience D. This can later be sampled from to re-iterate over previous experience. The loss function at iteration i which aims to be minimized becomes

$$L_i(w_i) = \mathbb{E}_{s,a,r,s'\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right] \quad (3)$$

This can be done using stochastic gradient descent on the constantly re-sampled mini-batches from D. The idea behind fixed Q-targets is to save old network weights w_i^- for a number of n steps. This means that for these n steps, the network is updated towards the Q-target of the old network. This also stabilizes training, since the network is not being changed after every turn. After n steps, the parameters w_i^- of this target network are overwritten with the current parameters w_i. The algorithm that was used by us can be found at B.2.1. Rewards are determined in the following manner: For every country the bot takes over, a reward of $R^a_{ss'} = 2$ is accumulated. However, for every troop lost, a reward of $R^a_{ss'} = -1$ gets added to the cumulative reward.

4. Experiments

4.1 Abstractions

This paper assumes certain abstractions and simplifications to the RISK game environment.
One simplification used in the approach of this paper is limiting the amount of troops to be stationed in one country to 10. This is done to keep the bot from placing all its troops into a single country, as this could provoke unwanted behaviour such as the bot focusing on keeping one of its owned countries alive. This would lead to an endless game of reinforcing one country and making it unbeatable, thus, preserving a fixed status within the game.

4.2 Setup Experiment

The experiments regarding the TD-learning algorithm focus on tuning the hyper-parameters. For this reason, four separate experiments were conducted each assessing the impact of increments on one hyper-parameter. For each experiment, the hyper-parameters were incremented from zero to one in a series of twenty steps. In a 1 versus 1 game, one bot was adjusted to use the incremented value while the second one kept using standard values. To account for the probabilistic nature of RISK, 500 games per increment were played and the results of each increment were averaged. Each graphic depicts the relation between the incremented hyper-parameter and the average number of turns needed for the adjusted bot to win (Turns to win) and the average percentage of games won for each increment.

4.2.1 Alpha

Two major peaks in the Win Rate at alpha values of 0.2 and 0.6, they coincide with relatively low 'Turns to Win' values which suggests that the optimum alpha value lies at either of these points.

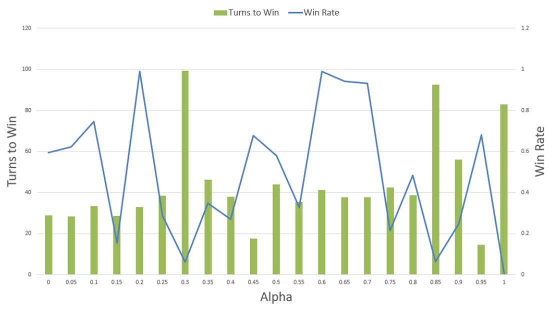

Figure 2: Win Rate and Turns to Win as Alpha Changes

As the alpha values of 0.2 and 0.6 both have the same 'Win Rate', the 'Turns to Win' values that is used to decide that an alpha value of 0.6 is optimal due to a high peak and low bar of all the potential candidates. Although 0.6 seems to be a high alpha value with a risk of overshooting the optimum, this high value allows for the bot to converge to an optimum quick enough to dominate the game.

4.2.2 Lambda

Peaks are observed at lambda values of 0.25, 0.45 and 0.85. Observing their 'Turns to Win' helps to determine which value to classify as optimal.

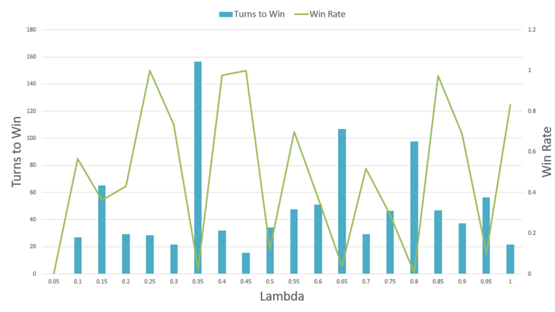

Figure 3: Win Rate and Turns to Win as Lambda Changes

From the figure above, it can be seen that the peak for lambda at 0.45 is most desirable, although the peak at 0.25 has an equal 'Win Rate', the 'Turns to Win' at lambda = 0.45 is significantly lower compared to lambda = 0.25. The lambda of 0.45 suggests that the bot does perform better when taking into consideration that previous states affect a future state. However, the value may not be higher due to the stochastic nature of Risk. Valuing intermediate states too much could be punished by bad 'dice rolls'.

4.2.3 Win-Chance Threshold

The win-chance threshold is used to determine whether the bot should start an attack against an adjacent country. A higher value therefore indicates a more defensive play-style, as the bot will only attack if the predicted win-chance is above the threshold.

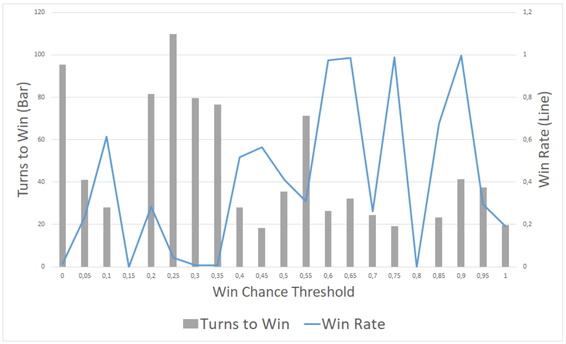

Figure 4: Win Rate and Turns to Win as Win Chance Threshold Changes

Based on these results a more defensive strategy, i.e. first collecting troops in conquered territories and attacking only when the predicted win-chance is high enough, seems to yield a higher overall win rate while also resulting in a lower number of turns needed to win. A value of 0.6 - 0.65 seems optimal as it results in a win-rate close to 1, while keeping the turns needed to win fairly low. Values above 0.65 show more fluctuation which could be due to the random-chance threshold, which also helps in deciding whether to start an attack. In general lower values for the win-chance threshold seem to result in a lower win-rate in conjunction with a higher number of turns needed to win. This could be explained by the bot starting a lot of risky attacks early and conquering additional territories which he is then not able to protect later on.

4.2.4 Random-Chance Threshold

Another parameter created for a RISK playing bot specifically in this paper is the Random-Chance threshold. This threshold was designed to introduce the tendency of the bot to take a rather risky and not well evaluated move in the turn for the case that the bot cannot find any desirable state based on observations made about the Win-Chance threshold. An increase in the Random-Chance threshold equals an increase in the bots willingness to take on an attack despite knowing that the chance of winning the according battle is low compared to the Win-Chance threshold.

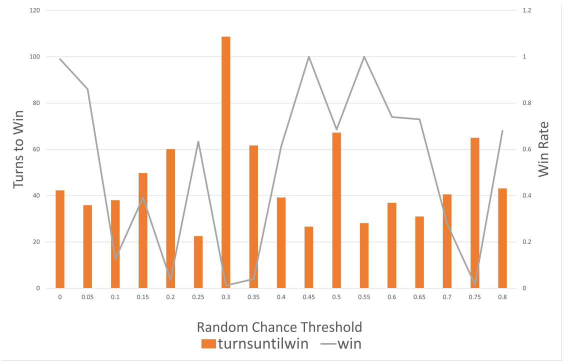

Figure 5: Win Rate and Turns to Win as Random Chance Threshold Changes

As can be seen in the figure above, with the current standard values for the other hyperparameters, it seems that the bot achieves a highest win-chance when setting the Random-Chance threshold to 0.45 or 0.55. It can be assumed that this is the margin within which the experiment bot plays as best opponent against the standard bot as in any other case, the bot takes too much or not enough risk to attack its opponent, resulting in taking more turns or not winning a lot of games overall.

4.3 Deep-Q Learning Approach

The main goals of a Q-Bot is to minimize the number of troops lost in battle while maximizing the amount of controlled territory. Executing well in that regard also results in an increased number of bonus troops at the distribution phase of the following turn.

4.3.1 Self-play Training Procedure

The loss, as defined at (3), must be minimized by the network. Instead of using the simple features, this experiment makes use of the features described in 3.1.

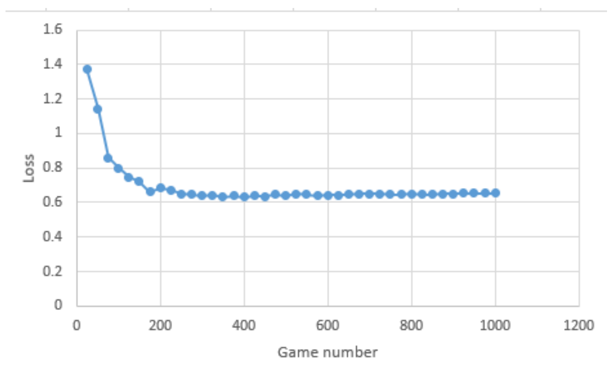

Figure 6: Loss over episodes of training

It can be observed that the average loss converges as the number of episodes increase. An average is calculated every 25 games.

4.3.2 Simple Attack Policy

Two approaches for describing a state were implemented. The first one, which will be referred to as simple features, describe a state as a 2-tuple (a, b). Here, a is the number of troops in the country to attack from and b for the country to attack to. The intuition was that by giving it different reward signals, it can decide on a decision rule for attacking.

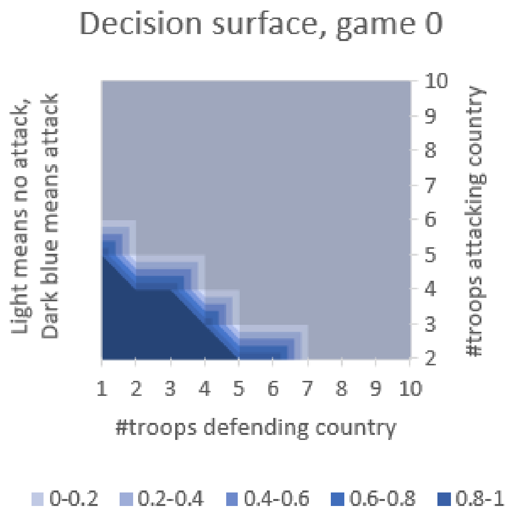

Figure 7: Initial policy of the DQN

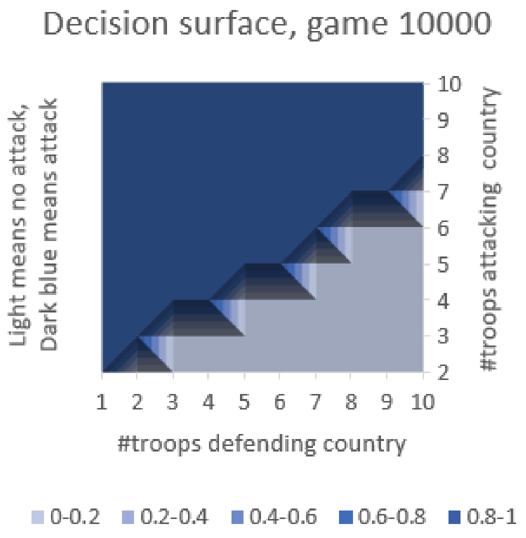

Figure 8: Converged DQN policy

Starting with initial decision space as depicted in Figure 7, control is performed using Deep Q-Learning. After 10000 games, the policy in Figure 8 is obtained.

4.3.3 Winning Player in Self-play

To track the progress made towards winning the game, the number of troops available for each player after the end of every turn for all games played during the training procedure is collected.

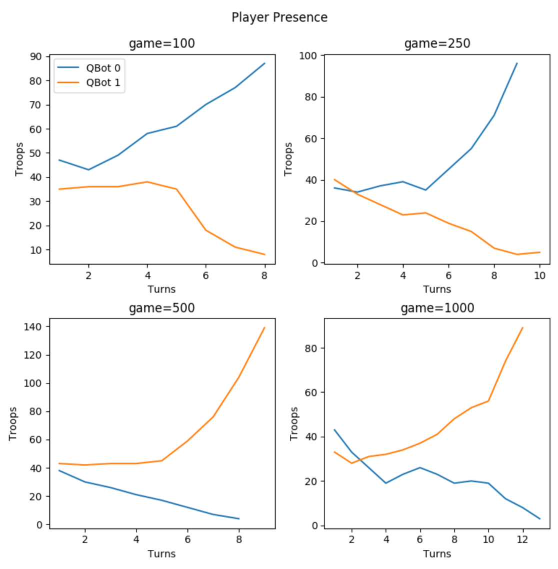

Figure 9: Number of troops in Self-Play

From the sample games displayed in Figure 9 we see that it is common for one of the Q-bots to dominate the game after 8 to 12 turns. Further it can be observed that for self-play as soon as a significant difference in the army strengths is reached, the player with the highest presence prevails and wins the game eventually.

4.4 Comparison of the methods

In order to compare the bots to a certain standard, a random bot was implemented. This bot makes random moves in the *attack* phase. However, in all other phases, it behaves exactly like other bots.

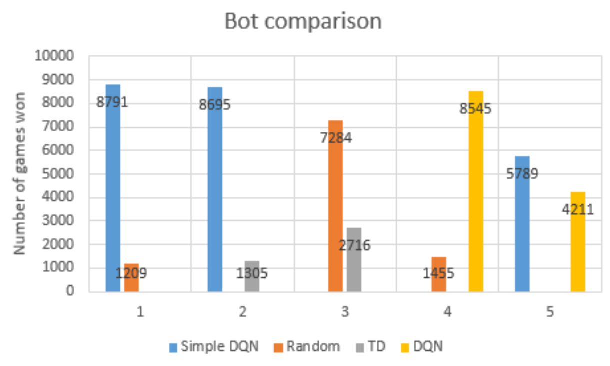

Figure 10: Comparison of the different bots

This way it is possible measure performance of attacking algorithms on a large number of games. In this experiment, each pair was trained for 10000 games. This largely reduces the variance. Simple Q-learning is the overall winner, defeating every other bot. Using the normal features for the DQN comes close, but fails to win the most games for the pair. The Temporal-Difference bot wins less than 30% of its games against the random bot.

5. Discussion

Multiple approaches were implemented in order to create a bot that can play the game of Risk. These approaches both made use of Temporal-Difference learning. The first aim of this research was to find out whether it is possible to create a bot which has the ability to play the game of Risk with reasonable performance.

For the Deep Q-Learning approach, using simple features to describe the state space resulted in a well-performing bot already. After training, it was able to defeat a random playing bot and TD-bot around 87% of the time. In a game like Risk, this is an excellent performance. As seen in figure 9, the player that creates the higher presence will eventually win the game. Since the game is very stochastic, there are some cases where the random player obtains this presence and will thus win the game.

The same idea holds for the approach when using more complex features (as described in section 3.1). However, there is something interesting to notice here. The complex features do not perform nearly as well as the simple features. There could be many causes for this. It is suspected that using a deep neural network with the simple features already creates a complex decision space. The more complex features may not represent the state as well as this space.

For the TD-Learning algorithm, we have shown that tuning certain hyper-parameters does have a significant influence on the resulting play-style of the bot. From these results, a more defensive approach seems to yield higher win-chances while keeping the number of turns needed fairly low. Looking at the results from the comparison experiment from section 4.4 we see a distinct difference when comparing the achieved win-chance to those achieved during self-play. While a high fluctuation in win-percentages could also be observed during the hyper-parameter tuning experiments the results from the comparison experiment seem to suggest a far worse performance. Part of the reason for this drop in performance could be the use of unoptimized hyper-parameters as the default values were used to represent a baseline for the TD-Algorithm. In addition, the TD-bot seems to follow a more defensive strategy overall. Here he first builds up significant army strength in its territories and then tries to win with large attacks. The random bot could intervene with the TD-bot's ability to build up his army strength in the first place by attacking early therefore not allowing the TD-bot to fully prepare its strategy resulting in a worse performance.

There are many ways in which these current approaches can be improved. Right now, a model-free approach has been used. This means the bot does not make use of knowledge about the game dynamics and rules. Implementing a simulation-based planning algorithm like Temporal-Difference Search for planning could increase performance drastically.

Alternatively, the bot could learn a model of the game; for instance by generating the maximum likelihood Markov model based on observed experience. These ideas can all be combined together to presumably improve performance.

The current algorithms can also be extended to handle the other remaining phases of the game. Then, reward can be tuned for the whole game, instead of just the battle phase.

6. Conclusion

It was shown that using Q-Learning with a simple multi-layered neural network [10], it is possible to extract a policy based on the win chances [4] of the battle phase, which is suitable to win games against a human player. Using simple features to describe the state space already results in a high performance against other approaches.

When comparing linear function approximations against non-linear ones, it can be noted that non-linear performs a lot better. This is due to the more complex decision space it can create. This leads to a powerful combination of the features, which has more flexibility.

TD-Lambda learning has shown to be a stable and slow pace bot that focuses on maintaining an optimum state for as long as possible, rather than playing with the game dynamics. This is due to the "Temporal Difference" and the linear evaluation function used, giving the bot only limited possibility to predict future outcomes and situations the bot might find itself in. This, combined with the less complex approximation of states due to the linear evaluation function gives the bot a less detailed understanding of the game, making it easier for human players or more advanced bots to out-think strategies used in simple TD-Lambda and leverage that predictability to win against the TD-bot rather easily. However, the slow pace strategies of Temporal Difference, mostly involving just building up a high troop count in owned territories and thus increasing the chance for an eventual attack to be successful, still makes the bot one possible solution to counter the problem of letting an AI master the game of RISK and win a reasonable amount of games.

References

[1] M. Wiering and M. Van Otterlo, *Reinforcement learning*, vol. 12. Springer, 2012.

[2] H. J. Van Den Herik, J. W. Uiterwijk, and J. Van Rijswijck, "Games solved: Now and in the future," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 277–311, 2002.

[3] B. Tan, "Markov chains and the risk board game," *Mathematics Magazine*, vol. 70, 12 1997.

[4] J. A. Osborne, "Markov chains for the risk board game revisited," *Mathematics Magazine*, vol. 76, no. 2, pp. 129–135, 2003.

[5] M. Wolf, "An intelligent artificial player for the game of risk," *Unpublished doctoral dissertation)*, 2005.

[6] F. Hahn, "Evaluating heuristics in the game risk an aritifical intelligence perspective," *Unpublished doctoral dissertation)*, 2010.

[7] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[8] J. Fürnkranz, "Machine learning in games: A survey," *Machines that learn to play games*, pp. 11–59, 2001.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[10] L.-J. Lin, "Reinforcement learning for robots using neural networks," tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[11] U. BoardGames, "The standard continents of risk," 2020.

Appendix A. GUI Details

A.1 In-Game Features

The game is presented as a world map consisting of SVGs representing each country. For more intuitive user interaction we have added labels indicating the number of troops placed in each country. Moreover on each player's turn we have a colored flag indicator showing the current player, a number of troops in inventory notification as well as text about the current phase of the game: distribution; placement; attack; fortify; win. A more concrete text indicating what is now required of the player is displayed at the bottom of the game window. To provide a more intuitive nature for human versus bot games, we have included a screen that displays the land(s) a bot has conquered after each turn the bot has played; so that the user can follow along with how the bot is playing with more ease. Finally the implementation of a skip button allows the user to fast forward the AI's turn and a conquer screen indicates after the turn which countries have been conquered by that AI.

A.2 SVG

The use of SVGs was chosen due to their versatility. We are able to scale the SVGs relative to the display that the game is being shown on. Other features such as the colours changing to the players colour when they become the owner of it; either by distributing troops or conquering in the activate phases. As the country being hovered over is also brought forward and make slightly larger, in addition, a shadow is cast on the country to draw the users attention to it. The colours available are all specifically chosen to be contrasting and vivid to draw interest from users of all ages, a secondary benefit is that it more clearly indicates a territory's owner.

Appendix B. Tabular information

B.1 Distribution Priorities

Player count	Priority 1	2	3	4	5	6
2	North America	Australia	South America	Africa	Europe	Asia
3	Australia	North America	South America	Africa	Europe	Asia
4	Australia	South America	Africa	North America	Asia	Europe
5	Australia	South America	Africa	North America	Europe	Asia
6	Australia	South America	Africa	North America	Europe	Asia

Figure 11: Priorities on countries for distribution [11]

B.2 Methods

B.2.1 Deep Q-Learning Algorithm

Algorithm 1 Deep Q-Learning Algorithm

```
Require: $\pi \Leftarrow \epsilon$-greedy($Q_w$), $n$ (batch-size), $lag$ (copy weights interval)
  $i \Leftarrow 0$
  while $\pi$ not converged do
    $A_t, R_{t+1}, S_{t+1} \sim \pi(S_t)$
    Append $(S_t, A_t, R_{t+1}, S_{t+1})$ to $D$
    for $i = 0, ..., n-1$ do
      $(S_d, A_d, R_d, S'_d) \sim D$
      $target \Leftarrow R_d + \gamma \max_{a'} Q_{w^-}(S'_d, a')$
      $err \Leftarrow target - Q_w(S_d, A_d; w)$
      $\nabla_w L(w) \Leftarrow (err \nabla_w Q(S_d, A_d, w)$
      $w \Leftarrow w - \alpha \nabla_w L(w)$
    end for
    if $i \mod lag = 0$ then
      $w^- = w$
    end if
    $i \Leftarrow i + 1$
  end while
```

B.3 Experiment Data

	1	2	3	4	5	6	7	8	9	10
2	1	1	1	1	1	1	0	0	0	0
3	1	1	1	1	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Figure 12: Data for decision surface at game 0, as presented at 7

	1	2	3	4	5	6	7	8	9	10
2	1	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	1	1	1	0	0	0	0	0	0
5	1	1	1	1	1	1	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0
7	1	1	1	1	1	1	1	1	1	0
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1

Figure 13: Data for decision surface at game 10000, as presented at 8

Game	Loss
25	1.373675118
50	1.142622606
75	0.859688564
100	0.799261594
125	0.746382538
150	0.72024093
175	0.663491657
200	0.683843653
225	0.671489613
250	0.650646243
275	0.648120404
300	0.64036958
325	0.643992871
350	0.632119079
375	0.63804814
400	0.630119071
425	0.639257048
450	0.636795815
475	0.645470185
500	0.637449925
525	0.652163947
550	0.647730151
575	0.639209795
600	0.643148434
625	0.64384299
650	0.645585634
675	0.649491162
700	0.650643593
725	0.649308528
750	0.646770803
775	0.646518211
800	0.648671937
825	0.64778524
850	0.647318807
875	0.649503873
900	0.650567805
925	0.653255578
950	0.654029195
975	0.655243877
1000	0.655459844

Figure 14: Data for loss of the DQN

Bot wins	1	2	3	4	5
Simple DQN	8791	8695			5789
Random	1209		7284	1455	
TD		1305	2716		
DQN				8545	4211

Figure 15: Data for the bot comparison

Appendix C. TD Normalization

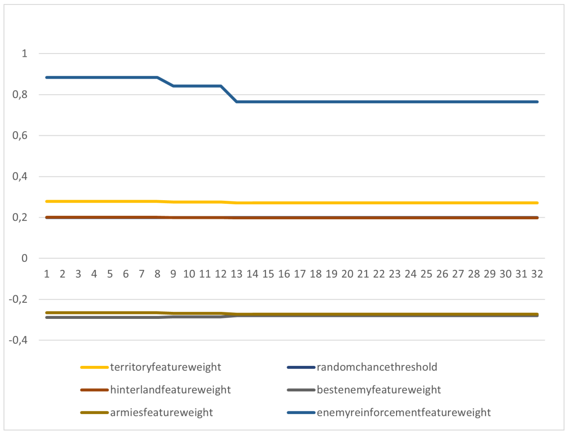

Figure 16: Normalized Feature Weights, no divergence

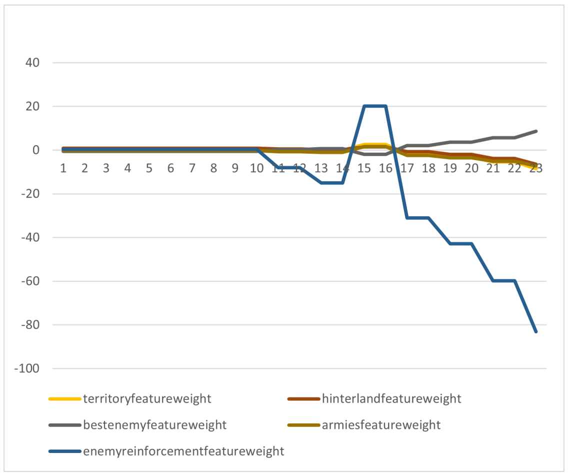

Figure 17: Non-normalized Feature Weights, massive divergence after indistinct amount of turns