Final Report

Daniel Yao

Abstract

Risk: The Classic World Domination Game is a military strategy game, where the objective is to control every territory on the map. Combat is determined by throwing dice, and this stochastic element lends itself to the application of Markov Decision Processes (MDPs) and Reinforcement Learning (RL). Previous work has shown that an RL agent can achieve modest improvements over a random agent in a simplified version of Risk. This report outlines the basic theory of MDPs and RL, specifically from the perspective of double deep Q-networks (DDQNs). A simplified version of Risk, which involves more actions than the previous work, is introduced, and a DDQN agent is trained to play the game against a random opponent. The agent achieves a win rate of over 90% against a random opponent. Nevertheless, the agent is still not very good, and it would not be able to compete against a human player. Future directions include designing a better reward function, training the agent against a heuristic opponent, or training the agent against itself.

1. Introduction

Risk: The Classic World Domination Game is a military strategy game that contains a stochastic element, where combat is determined by throwing dice [1]. This stochastic element lends itself to the application of Markov Decision Processes (MDPs) and Reinforcement Learning (RL), which address decision-making in a stochastic environment. RL agents have previously achieved super-human performance in many games, in board games like chess and video games like Atari. However, many highly stochastic board games such as Settlers of Catan or Risk have not been solved by RL [2] [3] [4] [5] [6]. In 2000, Keppler and Choi, two undergraduate students at Cornell University, developed an RL agent to play a simplified version of the game. However, much of the strategy was hard-coded, including all of the positioning decisions. Indeed, the number of actions was limited to only 242 in that version of the game [7].

In this report, the basic theory of MDPs and RL is outlined, specifically from the perspective of double deep Q-networks (DDQNs). An RL agent for a simplified version of the game that extends the work of Keppler and Choi to involve many more actions, including those of positioning, in the decisions determined by the RL agent.

2. Markov Decision Processes

MDPs are used to model decision-making problems in a stochastic environment. First, a definition.

Def. A finite MDP is a five-tuple (S, A, P, R, γ) where

1. S is the finite state space,
2. A is the finite action space,
3. $P : S \times A \times S \to [0, 1]$ is the transition function,
4. $R : S \times A \times S \to \mathbb{R}$ is the reward function, and
5. $\gamma \in [0, 1]$ is the discount factor.

$A(s)$ is the space of actions available in state $s \in S$. $P(s' \mid s, a)$ is the probability that the next state is $s' \in S$ given that the current state is $s \in S$ and the action taken is $a \in A(s)$. $R(s', a, s)$ is the reward received when the current state is $s' \in S$, the action taken was $a \in A(s)$, and the previous state was $s \in S$ [6].

An agent is said to follow a policy π, which determines the distribution of actions taken in each state.

Def. A policy π is a function $\pi : A \times S \to [0, 1]$ where $\pi(a \mid s)$ is the probability that an agent in state $s \in S$ takes action $a \in A(s)$ [8]. A policy is said to be deterministic if $\pi(a \mid s) = 1$ for some action $a \in A(s)$ for each states $s \in S$ [6].

It is clear that an MDP has the Markov property, in that the future is independent of the past, given the present. Indeed,

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t).$$

for all $t \geq 0$. Given an MDP, a Markov chain may be recovered with state space S and transition matrix $\tilde{P}$ from an MDP by summing over $A(s)$ to obtain

$$\tilde{P}(s' \mid s) = \sum_{a \in A(s)} P(s' \mid s, a)\pi(a \mid s)$$

for all $s, s' \in S$ [6].

With this relationship between MDPs and Markov chains, a transition diagram for an MDP may be exhibited. Figure 1 shows an example with three states and actions that move between the states. Notice that the transition function P in this example is deterministic. The next state is completely determined by the action taken, which is not the case in general [6]

The objective of an agent is to take actions that maximize some measure of reward. This idea is formalized by the definition of discounted return G_t, the state value function v_π, the state-action value function q_π, and the corresponding optimality conditions.

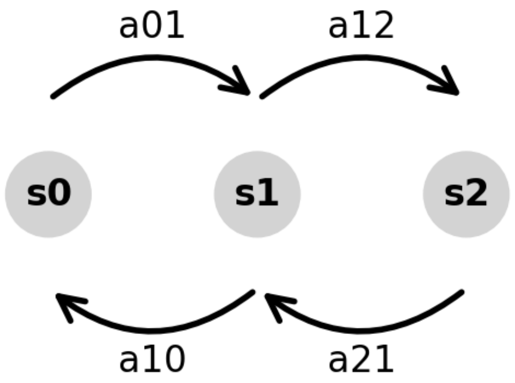

Figure 1: Transition Diagram for MDP. The transition diagram for an MDP with state space $S = \{0, 1, 2\}$ and action space $A = \{a_{01}, a_{12}, a_{21}, a_{10}\}$. The action a_{01} transitions from state 0 to state 1 and likewise for the other actions. The transition function P is deterministic in this example, but it need not be in general.

Def. The discounted return G_t at time t is the sum of all future rewards, discounted by the factor γ. That is,

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where R_{t+k+1} is the reward received at time $t + k + 1$. Notice that G_t and R_{t+k+1} are random variables. If games do not almost surely end, then a choice of $\gamma < 1$ guarantees that the sum always converges [6].

Def. The state-action value function $q_\pi : S \times A \to \mathbb{R}$ gives the expected return of taking action $a \in A(s)$ in state $s \in S$ and following policy π thereafter. That is,

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

where $\mathbb{E}_\pi$ is the expectation with respect to the distribution of states and actions induced by policy π [6].

Def. A policy π is said to be optimal if it maximizes the expected return for all states $s \in S$. That is,

$$\pi^* = \arg\max_\pi q_\pi(s, a)$$

for all $s \in S$ and $a \in A(s)$. An optimal policy need not be unique [6].

Here, it is seen that the optimal policy π^* induces an action distribution $\pi^*(a \mid s)$ for each $s \in S$ that maximizes the expected return. Indeed, once $Q_{\pi^*}(s, a)$ is known, the optimal policy may be determined by a greedy algorithm, where an action $a \in A(s)$ that maximizes $Q_{\pi^*}(s, a)$ is chosen in each state $s \in S$. This is known as a greedy policy. This idea is stated formally in a theorem. A sketch of the proof is provided as well.

Def. A policy π is said to be greedy with respect to the state-action value function Q_{π^*} if

$$\pi(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max Q_{\pi^*}(s, a') \\ 0 & \text{otherwise} \end{cases}$$

for all $s \in S$. It is clear that a greedy policy is deterministic [6].

Lem. The state-action value function q_π may be expressed recursively in the Bellman expectation equation. A quick manipulation shows that

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi [G_t \mid S_t = S, A_t = a] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi [R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \text{ [6].} \end{aligned}$$

Lem. The Bellman optimality equation characterizes the optimal state-action value function q^*. It states that

$$q^*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a_{t+1}} q_{\pi^*}(S_{t+1}, a_{t+1}) \mid S_t = s, A_t = a\right],$$

which follows immediately from the Bellman expectation equation [6].

Thm. Any greedy policy with respect to a state-action value function q^* that satisfies the Bellman optimality equation is an optimal policy [9].

Pf. (Sketch) Let $T : \mathbb{R}^{S \times A} \to \mathbb{R}^{S \times A}$ be the Bellman operator defined by

$$T(q(s, a)) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a_{t+1}} q(S_{t+1}, a_{t+1}) \mid S_t = s, A_t = a\right].$$

T is a contraction mapping with respect to the infinity norm, so it admits an unique fixed point q^* by the Banach fixed point theorem, and moreover, q^* is optimal by the Bellman optimality equation. Any greedy policy π^* induced by q^* is optimal [9].

3. Reinforcement Learning

From these statements, the motivation of RL is evident. T is a contraction mapping, so by repeatedly applying T to an initial state-action value function Q_0, the fixed point q^* is approached. (Here, Q denotes an approximation of q_π.) The agent learns the state-action value function Q by interacting with the environment and observing the rewards. It then uses the learned Q to approximate the optimal policy π^* by approximating the Bellman operator [9].

This idea is captured in Q-learning. Q-learning is a model-free RL algorithm, which means that it does not require knowledge of the transition function P or the reward function R. This has the advantage in that P may be very hard to describe analytically [6].

Tabular Q-learning maintains a table of state-action values $Q(s, a)$ for all states $s \in S$ and actions $a \in A(s)$. This approach is limited, however, in that a large state-action space may require a large table and that not all state-action pairs may be seen. Risk is a game with a large state-action space, so the method used is instead deep Q-learning, which uses a neural network to approximate the state-action value function Q [4].

In particular, DDQN is used, which involves two neural networks: a target network Q_{θ^-} and an online network Q_θ with parameters θ^- and θ, respectively. The target network Q_{θ^-} is used to compute the target value for the current state-action pair, while the online network Q_θ is used to compute the policy. By maintaining two networks, DDQN mitigates the overestimation bias that often results when evaluation and selection are performed with the same network [10].

Given state $s_t \in S$, an action a_t is computed by the online network with an epsilon-greedy policy π_ε that chooses the greedy action with respect to Q_θ with probability $1 - \varepsilon$ and chooses an action uniformly at random with probability ε. At each step, ε is decayed exponentially with decay rate β from an initial value ϵ_i to a final value ϵ_f, where β, ϵ_i, and ϵ_f are hyperparameters. The purpose of the epsilon-greedy policy is to balance exploration and exploitation. In early training, the agent explores the state-action space by taking random actions with high probability. As training progresses, the agent exploits the learned state-action value function by taking greedy actions with high probability [4].

A replay buffer is maintained as a queue of experiences (s_t, a_t, r_t, s_{t+1}), where s_t is the current state, a_t is the action taken, r_t is the reward received, and s_{t+1} is the next state. The replay buffer is of length N, where N is a hyperparameter. At each step, a minibatch of size M is sampled uniformly at random from the replay buffer [4].

For each sample of the minibatch, the temporal-difference error is computed as

$$\delta_t = R_t + \gamma\, Q_{\theta^-}(s_{t+1}, a^*_{t+1}) - Q_\theta(s_t, a_t),$$

where a^*_{t+1} is the greedy action with respect to Q_{θ^-} in state s_{t+1}. The TD error is the difference between the θ-predicted state-action value and the θ^- target state-action value.

The update rule for the online network is

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\delta_t)$$

where $\alpha > 0$ is the learning rate and $L : \mathbb{R} \to \mathbb{R}$ is the loss function. The online network is copied to the target network every C steps, where C is a hyperparameter [10]. In this way, the agent learns to approximate the optimal state-action value function q^*, and therefore the optimal policy π^*, by interacting with the environment and observing the rewards received.

4. Simplified Risk

A simplified version of Risk is now introduced [1].

The map consists of eight territories, numbered $0, \ldots, 7$, which are divided into three continents. Continent A consists of territories $0, 1, 2$, continent B consists of territories $3, 4$, and continent C consists of territories $5, 6, 7$. The territories are connected by undirected edges. This map is meant to represent the Old World, where A is Asia, B is Europe, and C is Africa. A graphical representation of the map is shown in Figure 2.

Let a_1, a_2 denote the number of armies owned by player 1 and player 2, respectively. Let t_1, t_2 denote the number of territories owned by player 1 and player 2, respectively. Let c_1, c_2 denote the continent bonuses of player 1 and player 2, respectively. Controlling every territory in continent A yields a continent bonus of $+2$, controlling every territory in continent B yields a continent bonus of $+1$, and controlling every territory in continent C yields a continent bonus of $+1$.

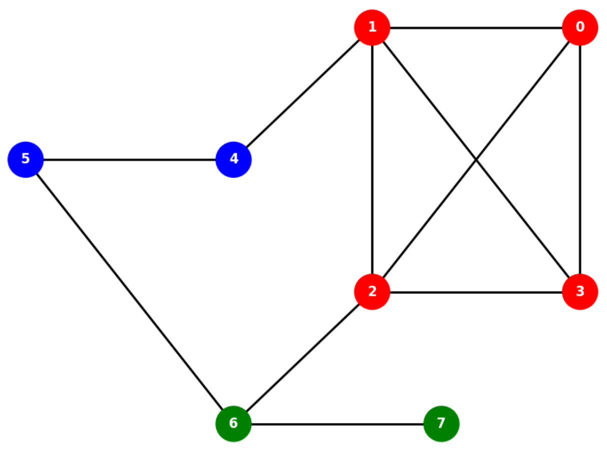

Figure 2: Map of Simplified Risk. Continent A (red) yields a continent bonus of $+2$, continent B (blue) yields a continent bonus of $+1$, and continent C (green) yields a continent bonus of $+1$.

The game begins with player 1 controlling one army in each of territories $0, 2, 4, 6$ and player 2 controlling one army in each of territories $1, 3, 5, 7$. Each territory must have at least one army at all times (so it must be owned by a player at all times), and a player may never control more than 40 armies.

Gameplay takes the form of turns, where each turn consists of three phases: Reinforce, Attack, and Fortify. Each of these phases are explained from the perspective of player 1, and of course, these phases are likewise from the perspective of player 2.

In Reinforce, player 1 places r_1 armies in a single territory that he controls, where

$$r_1 = \max\left(\left\lfloor \frac{a_1}{3} \right\rfloor + c_1, 1\right).$$

Player 1 may only reinforce up to his maximum of 40 armies.

In Attack, player 1 has the option to move as many armies as he wants from one territory that he controls to one adjacent territory that his opponent controls. He must leave at least one army in the source territory. Suppose that player 1 attacks a territory with m_1 armies, and that player 2 defends the territory with m_2 armies. Player 1 throws $\min(m_1, 3)$ dice, and player 2 throws $\min(m_2, 2)$ dice. The highest die of each player is compared, and the second highest die of each player is compared (if possible). Player 2 wins ties. The number of armies of each player is decremented by the number of dice lost, and the process is repeated until one player has no armies left. Control of the territory is transferred to the winning player. An intermediate reward is given that equals the difference in the number of armies between the two players, normalized to $[-1, 1]$.

In Fortify, player 1 may move as many armies as he wants from one territory that he controls to one adjacent territory that he controls. He must leave at least one army in the source territory.

The game ends when one player controls every territory or when 40 turns have passed (20 turns for each player). A final reward is given, which equals $+1$ for a victory, -1 for a loss, and the difference in the number of armies between the two players, normalized to $[-1, 1]$, if 40 turns have passed without a winner.

The stochastic nature of the game is demonstrated in Figure 3

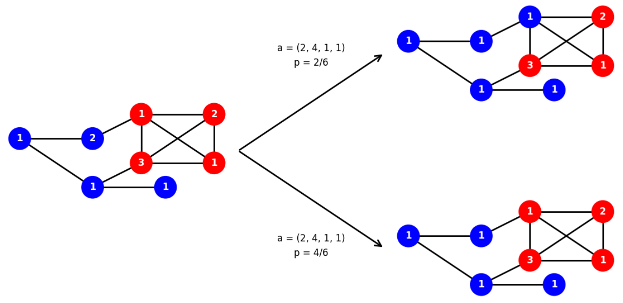

Figure 3: Transition Diagram for Attack. The action $a = (2, 4, 1, 1)$ is interpreted as Attack (phase 2), source territory 4, destination territory 1, and 1 army moved. The outcome of this action is determined by the throw of a die. It is probability $2/6$ that the attack is successful, and it is probability $4/6$ that the attack fails.

Simplifications in this version include that the game is played on a small map with only two players, that the game ends after 40 turns, that the game begins with a fixed board state, that only one territory may be reinforced, that only one territory may attack or be attacked, and that only one territory may be fortified. Victory cards are also not included. [1].

Nevertheless, even with this simplified version, the number of states and actions is still large. Indeed, the state space is encoded as

$$S = \{1, 2, 3\} \times \{1, 2\}^8 \times \{1, \dots 40\}^8.$$

The first coordinate specifies which phase it is. The other coordinates specify who controls each territory and how many armies are in each territory. It can be seen that $|S| = 3 \times 2^8 \times 40^8$, which is an astronomically large number.

The action space is encoded as

$$A = \{1, 2, 3\} \times \{0, \dots 8\} \times \{0, \dots, 8\} \times \{0, \dots 40\}.$$

The first coordinate specifies which phase it is. The second coordinate specifies the source territory of the action. The third coordinate specifies the destination territory of the action. The fourth coordinate specifies the number of armies in the source territory that are moved to the destination territory. An action vector is interpreted according to the current state to produce a state transition. It can be seen that $|A| = 3 \times 8 \times 8 \times 40$, which is also a large number. The state-action space $S \times A$, is therefore intractable for tabular Q-learning, which is why deep Q-learning is used instead of tabular Q-learning.

α	M	β^{-1}	γ
10^{-8}	64	500	0.9
10^{-7}	128	1000	0.99
10^{-6}	256	2000	1

Table 1: Hyperparameter Values. α is the learning rate, M is the batch size, β is the decay rate, and γ is the discount factor. All other hyperparameters are fixed constant.

5. Methods

The simplified version of Risk was implemented with Gymnasium [11]. A DDQN with two hidden layers of 128 nodes each was implemented with Tensorflow and Keras [12] [13]. The agent was trained with an epsilon-greedy policy against a random agent, which chooses actions uniformly at random. The agent was evaluated with a greedy policy against the same random agent.

A Huber loss function was used with $\delta = 1$ to mitigate the effects of outliers. The replay buffer size $N = 10{,}000$ was fixed, the target update period $C = 2$ was fixed, the initial $\varepsilon_i = 1$ was fixed, and the final $\varepsilon_f = 0.1$ was fixed. The hyperparameters swept where the learning rate α, the batch size M, the inverse decay rate β^{-1}, and the discount factor γ. These hyperparameters took on three values each for a total of 81 combinations.

The values of the hyperparameters tested are shown in Table 1. Each combination was trained for 200 games and evaluated for 100 games. The win rates and the final rewards were recorded. Final reward is the average final reward after a victory or 40 elapsed turns. It is calculated as the difference between the final army counts, normalized to $[-1, 1]$, and it serves as a measurement of how likely the agent is to achieve victory if the game were to continue. A win was determined by whether the agent controlled more armies than the opponent at the end of the game. In the case that both players controlled, the same number of armies, a draw was counted as half a win.

The best combination of hyperparameters, measured by win rate, was trained for 600 games and evaluated for 100 games. Every 50 training games, the win rate was evaluated in 100 games against a random agent. The loss curve and win rate were recorded.

6. Results

Table 2 shows the results of the best five combinations of parameters as determined by final reward. Figure 4 exhibits the loss curve for each of these five combinations. Figure 5 exhibits the loss curve and win rate for the best combination of hyperparameters, $(\alpha, M, \beta^{-1}, \gamma) = (10^{-7}, 64, 2000, 0.99)$.

$(\alpha, M, \beta^{-1}, \gamma)$	Win Rate	Final Reward
$(10^{-7}, 64, 2000, 0.99)$	0.96	0.62
$(10^{-7}, 128, 500, 0.99)$	0.94	0.57
$(10^{-7}, 128, 1000, 0.9)$	0.93	0.70
$(10^{-7}, 64, 500, 1)$	0.85	0.35
$(10^{-7}, 128, 500, 1)$	0.83	0.51

Table 2: Hyperparameter Results. The five sets of parameters with the highest win rate are exhibited. α is the learning rate, M is the batch size, β is the decay rate, and γ is the discount factor. Final reward is the average final reward after a victory or 40 elapsed turns. A win was determined by whether the agent controlled more armies than the opponent at the end of the game. A draw was counted as half a win. Small α yields the best results.

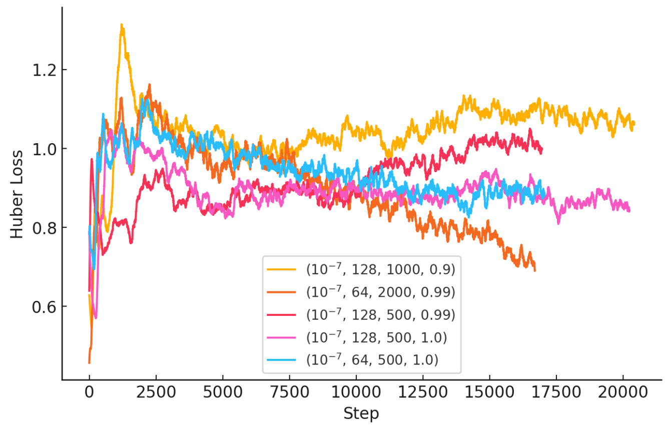

Figure 4: Loss Curves. The loss curves for the five sets of parameters with the highest final reward are exhibited. The loss data are smoothed as a moving average of 100 steps. The set of hyperparameters $(\alpha, M, \beta^{-1}, \gamma) = (10^{-7}, 256, 2000, 0.99)$ (dark orange) shows the most stable loss curve. This observation concurs with the result that the win rate was the highest for this set of parameters.

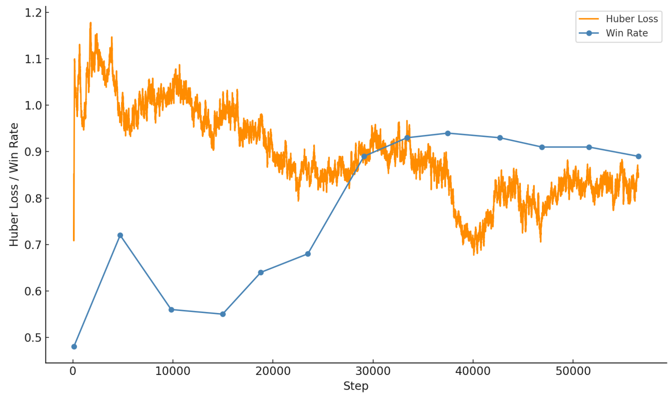

Figure 5: Loss Curve and Win Rate for Best Hyperparameters. The agent was trained with $(\alpha, M, \beta^{-1}, \gamma) = (10^{-7}, 64, 2000, 0.99)$ for 600 games. The loss data are smoothed as a moving average of 100 steps. Every 50 training games, the win rate was evaluated in 100 games against a random agent. The loss plateaus at around 0.8 after 400 games, and the win rate likewise plateaus at around 0.9 after 400 games.

7. Discussion

The sweep of parameters demonstrated that the learning rate α is the most important hyperparameter, with small α being favored for more stable loss convergence. Indeed, if $\alpha \geq 10^{-5}$, then the Huber loss diverges.

It was found that increasing the number of training games did not improve performance after 400 games as exhibited in Figure 5. The loss converged to a value of around 0.8, and the win rate likewise plateaued at around 90%. (Indeed, the win rate actually decreases after 400 games.) This suggests that the agent is capable of learning a policy that is better than random, but cannot continue to improve after a certain point.

Still, the win rate achieved by the agent is not as good as one would expect against a completely random opponent. One aspect that could have contributed to this less-than-desired performance could be that the random opponent contributed too much variance to the training, which could have made it difficult for the DDQN to estimate the true state-action value function. Another aspect is that the size of the state-action space is astronomically large, so the agent cannot explore an appreciable fraction of the state-action space. This may inhibit the ability of the DDQN to generalize to unseen states. Finally, the reward function chosen may have been too sparse, making it difficult for the agent to see the future results of its actions.

This agent improves upon the agent of Keppler and Choi in that it involves more actions that are decided by the RL agent. Morever, Kepler and Choi achieved only a 48% win rate in a game against two random opponents while the agent of this report improves this result to a 90% win rate in a game against one random opponent. Keppler and Choi address only the Attack, but they do not address Reinforce or Fortify. These decisions are rather controlled by a hard-coded heuristic strategy. The agent of this report is able to learn a policy for all three phases of the game, which is a significant improvement over the work of Keppler and Choi.

8. Conclusion

Though the agent achieved a win rate of over 90% against a random opponent, it is still not very good. The agent would not be able to win against an heuristic agent or against a human player. Future work to improve this performance could include designing a better reward function that gives a reward in each step, training the agent against a heuristic agent, or training the agent against itself. Once the agent is able to master this simplified version of Risk, the next step would be to train the agent to play a more complex version of the game, which may include more players, more states, and more actions.

9. Code Availability

https://github.com/dyao13/risk_agent

References

[1] P. Brothers, Risk: The Classic World Domination Game (1993).

[2] M. Pfeiffer, Reinforcement learning of strategies for settlers of catan (2004).

[3] A. Lukin, P. Johansson, M. Zhao, A survey of reinforcement learning in board games: From perfect-information to stochastic domains, ACM Computing Surveys 53 (5) (2021) 1–36.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, nature 518 (7540) (2015) 529–533.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, ..., D. Hassabis, Mastering chess and shogi by self-play with a general reinforcement learning algorithm, Science 362 (6419) (2018) 1140–1144.

[6] R. S. Sutton, A. G. Barto, et al., Reinforcement learning: An introduction, Vol. 1, MIT press Cambridge, 1998.

[7] D. Keppler, E. Choi, CS 473 - An Intelligent Agent for Risk, Tech. rep., Cornell University (2000).

[8] M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.

[9] C. J. C. H. Watkins, P. Dayan, Q-learning, Machine Learning 8 (3-4) (1992) 279–292.

[10] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 30, 2016.

[11] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al., Gymnasium: A standard interface for reinforcement learning environments, arXiv preprint arXiv:2407.17032 (2024).

[12] F. Chollet, et al., Keras (2015).

[13] M. A. et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, TensorFlow Development Team (2015).