Reinforcement Learning Agents for Onitama

Jake Crampton(5033), Noah Zemlin(5033)

University of Oklahoma, CS5033 Fall '19 Long Reinforcement Learning Project

All code used can be found at https://github.com/noahzemlin/PyOnitama. Code is spread throughout branches.

Abstract

We use reinforcement techniques including TD learning, TD-λ, minimax and alpha-beta pruning decision trees, Q-Learning, and Watkins' Q-λ Learning to create agents to play the chess-like strategy board game Onitama. Because Onitama is a perfect information non-stochastic game, it is very well suited for rewards based learning. The largest difficulty is the branching factor. Though much simpler than chess, it still has an average branching factor of 10-20. We discuss how this caused problems for specific learning methods, what choices we made or what new methods we adopted to account for these problems, and ultimately how we took our learning and made an agent that was able to defeat a generally intelligent human player.

Onitama: Domain

Onitama is a Japanese-Themed board game similar to chess. It is played on a 5x5 grid where the goal is to capture an opponent's king piece by landing one of your pieces on top of it. You can also capture opponents' pawns to remove them from the game, just like in chess. This is where the similarities to chess end, however.

In Onitama, there is a secondary win condition, where either can be met to win the game. The square your king starts at is called your "throne", and should you land your king on top of your opponent's throne, you win. This provides a layer of balance between offensive and defensive strategies and disincentivizes players from keeping their kings out of the action, as they are removing one of their win conditions in doing so.

The second and final change, though in many ways the most important, is the movement system. 5 cards are dealt from a deck of 16 at the beginning of the game to determine pieces' movement options, though in our experiments we kept the same set of 5 throughout. Each card defines allowable moves for a piece. When you use a card to move a piece, it must be oriented so that the name of the card is facing the player.

When you use a card to make a move, you also must swap it with the card in the middle. This means that you must get rid of a card to use it, so though some cards are incredibly good, it might be better to not use them and keep them as a threat than to give them up for a short term advantage.

Related Work

No researchers have published on Onitama. However, chess is a very similar game, and many people have heard of Deepmind's Alpha Zero and AlphaGo. In the paper on the aforementioned programs (Silver), the creators talk about how they used neural nets, reinforcement learning, and self play to create the most successful chess program ever. We don't use neural nets, but reinforcement learning is what this project is for, and self play is used in most of our experiments.

TD-λ learning, which is one of the learning methods we'll be discussing, was pioneered by Richard Sutton. In his 1988 paper "Learning to Predict by Methods of Temporal Difference", Sutton talks about how to train algorithms that learn based on chains of states and actions instead of just actions. This becomes extremely useful in a game with a massive branching factor like Onitama, as it allows you to look at the value of the starting state much more quickly.

Q-learning was first introduced by Chris Watkins in his 1989 thesis (Watkins 1989) and is the basis for our Q-learning agent. The idea behind Q-learning in contrast to TD learning is to learn an approximation for Q* as opposed to V* so that we not only learn the best states to be in but also the best actions to take from those states. This then will give us a policy that is an approximation of $\pi*$ by following the best actions from each state. As described in Watkin's thesis, Q-learning also has the advantage of being model-free meaning that it does not require any knowledge of the world to train.

One downside of Q-learning is that in environments with large amounts of states or low information, it can be extremely difficult to learn enough of Q to make a good agent. Therefore using accumulating traces like in TD(λ) is desirable to train more of the states for every action. Rummery and Niranjan (1994) use a combination of TD(λ) and Q-learning to create an agent that learns Q using accumulating traces, Watkin's Q(λ), as described in Watkin's thesis. Additionally, they experiment with a similar method to learn Q with accumulating traces which is SARSA(λ). These agents all proved to be able to explore large states much quicker than the traditional implementations of Q and SARSA. We chose to use Watkin's Q(λ) for our experiments due to its simplicity to implement within existing Q-learning code.

An interesting way to train RL agents for competitive games, like Onitama, is to train them against themselves. In the recent paper by the people at Open AI, (Bansal et al., 2018) they use self play to creative competitive agents that learn from each other. Agents learn to take actions that lead to maximal reward, but because the agent is playing against itself it also learns how to play against those

strategies at the same time. This allows the agents to explore many different strategies for winning. As shown in the paper, self-play also helps generalize the learning of the agent to new settings. This same method was also used in the later Open AI paper featuring agents playing hide and seek (Baker et al., 2019). These agents train against themselves and another competitive agent to maximize reward within hide and seek. This proved to create agent that performed very well and showed many interesting strategies. We use this self-play strategy to improve the performance of our own agents.

Aside from learning algorithms, move decision algorithms are the next most important part of a strategy game AI. In Jake's TD-λ experiment, he uses alpha-beta pruning. Donald Knuth analyzed this algorithm in 1975 (Knuth). In his paper, he describes how it can be used to consider less subtrees than a minimax algorithm and offers several insights as to how it can be used for chess, including state evaluation. This is incredible, as it was done 40 years before AlphaZero, which uses just this idea. We use a similar method as well, though not as advanced.

Aside from training a state space evaluator, another option is to run simulations of the game to get win rates. Combining decision trees with this is done in Monte-Carlo searching, which is described by Robert Harrison in his 2010 paper "Introduction to Monte Carlo Simulation". This is a very interesting way to "estimate mathematical functions and mimic the operations of complex systems", which would save on training time, but it is a project for another day, as it isn't reinforcement learning.

Experiment: TD Learning

Hypothesis

Our first hypothesis was that TD learning, with almost no help given to the agent, would be able to learn to beat the heuristic agent 50% of the time.

Methods

We implemented TD learning using the backup equation:

$$(1)\ V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

with $\alpha = 0.1$ and $\gamma = 0.98$. Here V(s) represents how good a state is based on rewards that it has gotten for taking actions that are available in state s, denoted by r_{t+1}, as well as how good the states it moves into are, denoted by $\gamma V(s_{t+1})$. We gave a reward of +5 for winning and -5 for losing with no sub-goal rewards.

When choosing an action to take, we used epsilon-greedy decision making with $\varepsilon = 0.15$, meaning that 15% of the time it would make a random move and 100%-15%=85% of the time it would take the greedy action, or the action a with the highest V(s') associated with the state s' after taking action a.

During training, the agent plays 2700 games against itself following the method mentioned above then enters 300 exhibition games against the heuristic agent, for a total of 3000 games per cycle, and at the end of this cycle data for the exhibition games is recorded and the agent continues to learn. We chose an exhibition period significantly larger than 100 because with 100 games the standard deviation was higher than we wanted.

For exhibition games, the agent's epsilon for decision making is set to 0 such that it always moves to the best state it can see. After the exhibitions, epsilon is set back to 0.15 for training.

Results

We measured our agent's performance in terms of its win rate over the 300 game exhibition matches. Results are pictured below.

As is evident from the scale of the vertical axis, we never got close to our hypothesized win rate of 50%. In fact, the agent does not appear to have learned very much at all after a slight increase in performance over the first 5000 games.

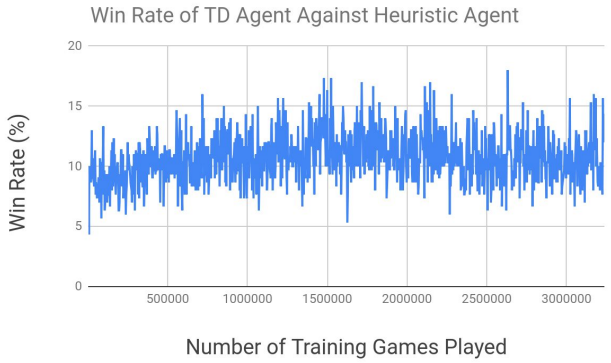

Figure 1: Performance of TD agent in exhibition matches against heuristic agent over training period of 3,500,000 games.

Conclusions

After thinking about what TD learning does in a qualitative sense, these results are not surprising considering the branching factor of this game. TD learning updates the previous state when you get a reward, but since we only gave rewards when the agent won or lost, the agent would receive a reward for the position before the win, but it would have to play randomly to get back there and observe that reward in the second to last state, then again in the third to last and so on. A branching factor of 15-20 and game depth of 50+ while playing randomly mean that it's almost impossible to get back to the end state and see the reward, and doing it 50 times to see it at the starting state is almost impossible.

Experiment: TD-λ Learning

Hypothesis

The next hypothesis was that TD-λ Learning with a minimax search tree with 2 move depth hard coded to take winning moves and avoid losing moves will be able to beat a heuristic agent 90% of the time.

TD-λ updates all the way back to the first move when it gets a reward, weighting this change less and less the further back it goes. This is much more in line with how humans learn.

Looking at the position 2 moves ahead is analogous to asking "what will this move allow me to do next", which is the most basic of strategies that humans develop when playing these kind of games.

Methods

Similar to TD learning, TD-λ uses a backup equation on a V state:

$$(2)\ V(s) \leftarrow V(s) + \alpha e(s)[r_{t+1} + \gamma V(s_t) - V(s)]$$

This looks the same as the previous equation except with the eligibility trace e(s), which determines how much of the reward the previous the state should care about, since this time the equation is applied to all states seen in the current game.

In this experiment, incrementing traces were used, meaning that when a state was seen, its eligibility trace was incremented by 1, and every time a different state was seen, the following equation was applied

$$(3)\ e(s) = \gamma\lambda e(s)$$

Where λ is a new hyperparameter which determines how quickly the "how much the agent cares about reward" metric drops off.

In this experiment, $\alpha = 0.005$, $\gamma = 0.98$, and $\lambda = 0.7$. We used the same reward function as before so as not to incentivize goals that don't optimize for winning.

We also used epsilon-greedy decision making with $\varepsilon = 0.15$ again. In training, it chooses the best V(s) of its children, but now in exhibition matches, the greedy option searches 2 moves in advance through a minimax tree. At leaf nodes, it uses V(s) as the value estimation, takes forced wins and avoids forced losses.

We used the same setup of self play and exhibitions over 3000 games as in TD Learning.

Results

We measured our agent's performance in terms of its win rate over the 300 game exhibition matches. Results are pictured below.

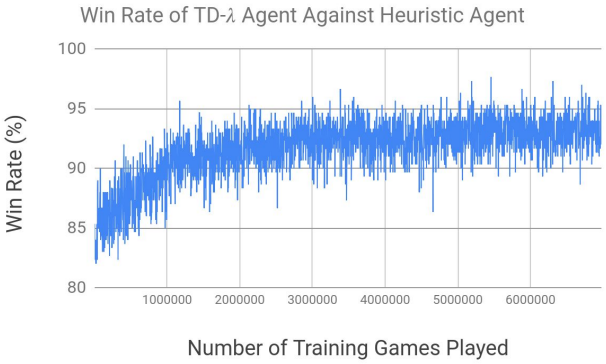

Figure 2: Performance of TD-λ agent in exhibition matches against heuristic agent over training period of 7,000,000 games.

This agent performed far better than the base TD agent. It appears to learn quite linearly for the first 1.5 million games before its improvements start to drop. The agent began with a center of around 86% win rate and max of 89% and ended with a center around 93% and low of 91%.

Conclusions

The hypothesis of achieving a 90% win rate was confirmed, with a final center of 93% and low above 90%. All of the improvements made to this agent were likely influential.

The instant choosing of forced wins both increases win rate dramatically, and increases learning speed because it removes time taken to learn searchable strategies and also speeds up the games since they end more quickly. It slows down exhibition games by searching at depth 2, but that is eclipsed by the gain from such a behavior.

By searching at depth 2, the agent is able to implement actual strategy. It is also to trap the opponent far more easily, since the heuristic agent can only take wins if they are immediately available.

This experiment was a success, and with such a dominating win rate over the heuristic agent, the next logical question is whether or not it can beat a human.

Experiment: Agents vs Human

Hypothesis

The final hypothesis for TD agents was that TD-λ Learning that has trained extensively will be able to beat a human if its search tree's depth is extended to 6 moves with alpha-beta pruning and if it keeps training during the match, starting each training game at the current state of the exhibition match.

In playing human vs human games in preparation for the project, we realized that tracing the cards was very difficult and prevented humans from easily thinking more than a few moves ahead. At depth 6, the agent would be thinking 3 moves ahead and would be rivaling what normal humans can do. However, minimax takes an hour per move to look at depth 6, whereas alpha-beta takes 20 seconds.

We also noticed that since its knowledge is based on how often it sees states, it gets bad in late game when it hasn't seen states before. This is why we do learning in parallel from the current state. It gets very fast when the turns left are low, and it helps the agent not be defeated by defensive strategies that extend move count.

Methods

The knowledge file from the successful TD-λ agent was taken and plugged into the agent with improved searching using alpha beta pruning, which Jake played against in an evening of exhibition games.

Then as mentioned in the hypothesis, while the agent was training, the same TD-λ experiment as before was being run, where the agents shared V representations, and the training agent started at the exhibition agent's current gamestate when a new game happened.

Jake also played against the TD agent, heuristic agent, and TD-λ agent with depth 2 and no continual learning for reference.

Results

Results were measured as a win percentage over 5 games if the agent did not beat the human in the first 5 games or 10 games if it did. Results below.

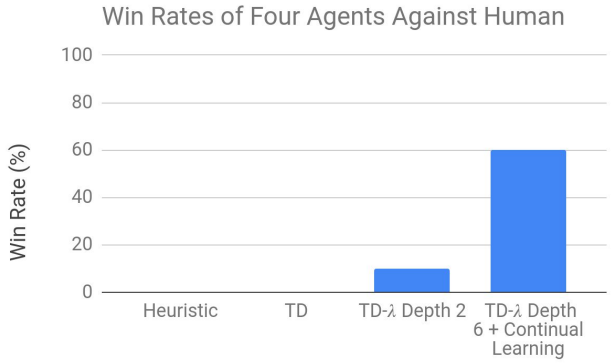

Figure 3: Performance of four agents in 5 (H and TD) or 10 (both TD-λ agents) game exhibition matches against Jake

The heuristic and TD agents did not manage to take any games, but the TD-λ agent from the previous experiment took 1, and the new TD-λ agent had a positive record winning 6/10 games.

Conclusions

The agents discovered that one of the ways the five set cards could be dealt resulted in a forced win in 3 moves, or depth 6 in a search tree. The TD-λ depth 2 agent won its only game using this win, but the TD-λ depth 6 agent also only won 1 game this way, meaning that 5 of the games it won were non-trivial victories. It should be stated that the depth 2 agent could not have found this by searching, so it is a good demonstration of learning.

The agent that could look three moves ahead was very hard to defeat. In most of the non-trivial games Jake lost, he made a single mistake more than 10 moves in and the agent immediately capitalized. In most of the games he defeated the agent, he had to take all of the agents pawns and force a long end game because he couldn't think as far ahead as the agent, and it defended perfectly.

Though the goal of the project was to demonstrate learning, the ability to defeat a human is considered a huge success.

Experiments: Q-learning and ε-decay

Hypothesis

- Q-learning will be able to beat a heuristic agent at least 50% of the time.

Methods

Q-learning aims to learn Q* by playing over many games and creating better approximations for Q* as it plays. Q* describes the value of choosing a specific state from a specific action. By learning Q*, we are able to learn an approximation to the optimal policy π^* by taking the best action according to Q* from each state. The following equation governs this learning process:

(4) $Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a))$

For this experiment, we used the hyperparameters $\alpha = 0.10$ and $\gamma = 0.98$. These values were chosen arbitrarily based on what traditionally works for Q-learning.

The reward the agent receives for this experiment was kept simple to not incentivize anything other than winning. As such, it was given a reward of +1 for moving to a winning state and -1 if it moved into a losing state. A result of choosing this reward structure would be that it will train much slower since it receives no feedback for the majority of the states it crosses. The future estimation portion of (4) is critical in this case.

The agent chooses an action to take each turn based on the Q it has learned so far. It will simply search for the best action based on the state it is in according to Q and take that action. While training, it uses ε-greedy to promote exploration of states. This means that $\varepsilon * 100\%$ of the time it will choose a random action instead of the best action. In order to encourage lots of exploration of different opening hands, we let ε start at ε=0.9 at the start of every game and let it quickly decay to ε=0.10 over the course of a game. I present this approach to decaying ε within each episode as opposed to over the episodes as novelty.

For benchmarking this experiment, we trained the agent in batches of 5000 games and recorded the performance after every batch. The performance is measured by having it play 250 games against a random agent and 250 games against a heuristic agent. The agent does not learn from these games.

Results

The following graph displays the learning rate for a Q-learning agent without ε decay and one with ε decay.

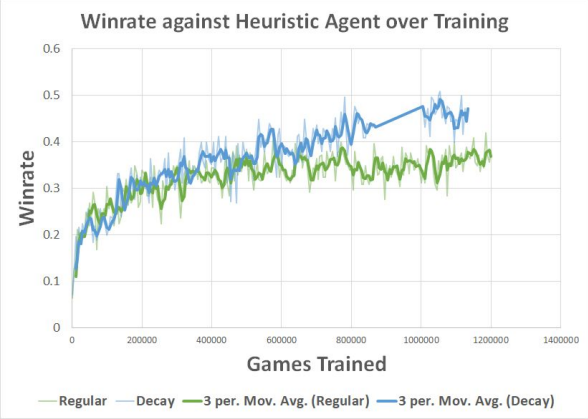

Figure 4: Performance of Q-learning agent in exhibition matches against heuristic agent over training period of around 1.2 million matches. The thick lines are a moving average of 15000 games. Note that the gap in the decay agent is due to minor data loss.

As seen in Figure 3, neither agent is able to consistently beat the heuristic agent 50% of the time, although it does briefly reach that mark a few times. It can also be seen that the Q-learning agent with decaying ε outperformed the static ε agent significantly. The static ε agent flatlined at a win rate of only 0.35 while the heuristic agent was able to reach .45.

Conclusions

Due to the agent not being able to consistently reach the 0.5 win rate threshold, the first hypothesis is not satisfied. The agent did learn however, and the ε-decay agent does appear to still be improving and could likely become better after more games.

After training the data, we experimented with playing against the agent and watching it play itself to attempt to see what it had learned. To our disappointment, we had found that the agent learned to stall games out by moving pieces back and forth so as to never lose and never receive negative reward. We can hypothesize that this is due to the limited reward structure of only +1 for winning and -1 for losing.

Additionally, we also found that the agent would have lots of training in the early moves (as seen by the relatively large magnitude of Q values) of each game but would quickly enter into states it had never seen before after 5-10 moves. We believe this is a large factor in the poor performance of the agent, but we are not surprised by this outcome. Because we represent the state of the board using the naive approach of giving it the entire board state, the state space is huge and the state-action space of Q is even bigger. One solution to this problem would be to use Q-learning with a neural network or some other function approximator to reduce the state space into something much more manageable. Given our limited knowledge of these methods at the time we did not try to accomplish this.

Experiments: Watkin's Q(λ) and Self-Play

Hypothesis

- Watkin's Q(λ) will outperform traditional Q-learning.
- Training Watkin's Q(λ) against itself will give better and more generalized results compared to against one type of agent.

Methods

Watkin's Q(λ) learns to approximate Q* in a slightly more robust approach in comparison to traditional Q-learning. Watkin's Q(λ) employs eligibility traces to not only learn from lookahead (via the \gamma term) but also update states in the past when a reward is given. We hypothesize that using accumulating traces will help the agent to learn more quickly as the rewards are given very infrequently and many states do not receive any changes to Q until it propagates backwards after reaching the same exactly victory states over and over.

In addition to training the Watkin's Q(λ) agent against only itself like in the previous experiment, we wanted to find out if self-play in this manner is the best way to train the agent. To test this, we performed two more learning attempts. One trained only against the random agent and the other trained only against the heuristic agent. We believe that the self-play agent will perform the best and generalize the best because it will learn to defeat itself over many different ways. By playing against only the heuristic agent it will only learn to beat the simple strategy it has. When it plays against the random agent we expect it to find a strategy that beats the random agent quickly since the random agent does not have the intelligence to stop any strategy.

All training in this experiment was performed using the exact same hyperparameters and rewards as in the previous experiment. Win rates are still calculated over 250 games.

Results

The following figure comparses the training and results of the traditional Q-learning and Watkin's Q(λ).

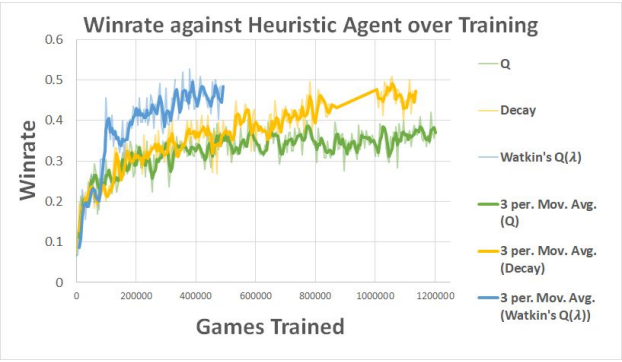

Figure 5: Performance comparison of Q (green), Q with ε-decay (yellow) and Q(λ) (blue) over time. The thick lines are a moving

average of 15000 games. Note that the Watkin's Q(λ) has much fewer games due to a bug that crashes the agent when it reaches large games.

As seen in Figure 4, our agent using Watkin's Q(λ) is able to quickly reach a win rate beyond the win rate of our traditional Q-learning. While it doesn't surpass the performance of the Q agent with ε-decay, it does train to that level much faster. It reaches the 0.4 win rate consistently at only 200k games played compared to 650k games.

In addition to training Watkin's Q(λ) with self-play, we also experimented with training it against other agents. The results are as follows:

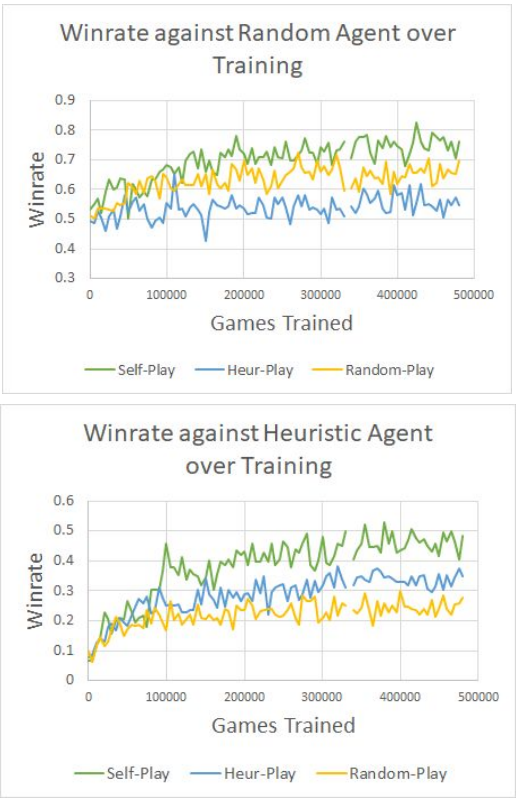

Figure 6(a) and 6(b): Performance comparison of Self-play (green), Heuristic trained (blue), and Random trained (yellow) against random agent and heuristic agent.

The results shown in Figure 5 show that training the agent against itself produces the best results when using the random agent or heuristic agent for performance measurement. Also as hypothesized, we can see that the heuristic trained agent performs better on the heuristic agent tests over the random trained agent and vice versa. The self-trained agent however consistently wins and shows that it is able to generalize better, at least to the random and heuristic agent.

Conclusions

Because Watkin's Q(λ) not only trains faster but reaches a higher win rate than any Q agent, we can support the hypothesis that Watkin's Q(λ) outperforms traditional Q-learning. Additionally, we can see that using self-play is the best way to train the agent such that is generalizes to more agents and also obtains the highest win rate. This supports the second hypothesis. We can extrapolate the results to also say that self-play will be the best way to train if we want this agent to beat humans as it can generalize to beating the game against more agents.

Comparisons

TD learning was not able to beat the heuristic agents because it could not get to the states that got rewards often enough.

TD-λ was able to beat TD learning's shot at the heuristic because it made percolated the reward all the way back to the first move, letting the agent choose good options again. The searching was also a huge boon.

Q-learning using ε-decay was able to beat the heuristic agents occasionally but not consistently. Watkin's Q(λ) had similar results but was able to reach that near 50% win rate much quicker. Traditional Q-learning was not able to reach above 42% winrate.

Novelty

Jake's novelty came in the form of using alpha-beta pruning and minimax to improve the decision making as well as having the agent train in parallel to playing.

Noah's novelty was to decayε within an episode as opposed to over the episodes. Onitama, like chess, has many opening hands that can wildly change the play of the game and by reducing ε within each episode the agent will explore more possible early games while still learning optimal play later on.

References

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B. and Mordatch, I., 2019. Emergent tool use from multi-agent autocurricula. arXiv preprint arXiv:1909.07528.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I. and Mordatch, I., 2017. Emergent complexity via multi-agent competition. arXiv preprint arXiv:1710.03748.

Rummery, G.A. and Niranjan, M., 1994. On-line Q-learning using connectionist systems (Vol. 37). Cambridge, England: University of Cambridge, Department of Engineering.

Watkins, C.J.C.H., 1989. Learning from delayed rewards.

Sutton, Richard S., 1988. Learning to Predict by the Methods of Temporal Differences. Machine Learning (Vol 3.1) pp9-44

Silver, D., Hubert, T., Schrittwieser, J., 2017, Mastering Chess by Self-Play with a General Reinforcement Learning Algorithm. Published in ArXiv 2017

Knuth, D., Moore, R., 1975. An Analysis of Alpha-Beta Pruning. Artificial Intelligence (Vol 6) pp 293-326

Harrison, R., 2010. Introduction to Monte Carlo Simulation. Published in AIP Conf Proc. 2010 Jan 5; 1204: pp 17-21