Department of Natural Science and Technology

Master Thesis for MSc in Engineering in Computer Science

Creating an AI opponent with super-human performance for Splendor

Author:
Jonatan Simonsson

Semester: VT2023

Supervisors: Jennifer Renoux, Örebro University.
Simon Johansson, Piktiv AB.

Examiner: Thomas Padron-McCarthy, Örebro University.

Abstract

This project is in collaboration with Piktiv AB. The purpose of the project is to create a reinforcement learning agent capable of reaching super human performance in the board game Splendor.

Three agents was implemented using a Double Deep Q-Network and certain improvements to the architecture. The improvements implemented was Prioritised Experience Replay and Multi-Step Learning. This thesis is the documentation of the implementation of the game Splendor, agents, training of agents, evaluation and result.

Table of Contents

1 Introduction

1.1 Motivation

This thesis is done on the behalf of Piktiv AB. Piktiv is interested in developing a superhuman level agent for a board game. Piktiv sees an increase in popularity for board game AI agents, they want competency and and a framework for developing such AI agents. Further down the road this and similar theses could be refined into a portfolio for Piktiv. Similar thesis work has been done at Piktiv earlier.

The goal of this project is to implement a discrete board game as well as an agent that can understand and play in this environment. Eventually this agent should

reach super human performance in the sense that the agent close to never loses against a normal human player.

1.2 Previous Work

1.2.1 Machine Learning in Games

Using machine learning to create agents for games is nothing new[1]. Historically board games like chess and go has been a driving force in pushing machine learning methods. Agents like Stockfish (chess)[2], AlphaGo (go)[3] and AlphaZero (chess and shogi)[4] has achieved incredible performance using machine learning. Both chess and go are complete games in theory, meaning that there is a perfect way to play. In practice the search space for these games are huge, chess has more possible board states than is feasible to calculate.

Using machine learning to play discrete board games with either randomness or imperfect information is also of great interest. Board games like Ticket To Ride, Risk and Catan has had agents created for them. A Double Deep Q-Learning agent[5] and Monte-Carlo Tree Search(MCTS)[6] has been created for Ticket To Ride as previous thesis works. Agents for Risk has also been created as previous thesis works[7, 8]. Agents for Catan has been created using MCTS[9]. The usage of machine learning doesn't stop at board games, using machine learning to play video games is very popular. When creating new reinforcement learning methods the Atari 2600 games has been a benchmark for performance. These games are fairly simple compared to what games machine learning can tackle today. OpenAI Five learned to play Dota 2 and beat the worlds best human team in 2019[10]. This was done with a mixture of supervised and reinforcement learning.

1.2.2 Q-Learning and DQN

Q-Learning is and has long been a staple in Reinforcement Learning. By combining Q-Learning and Neural Networks the architecture Deep Q-Learning was born. Both Q-Learning and DQN determines how valuable an action is by giving each action a Q-value. This will be further explained in Section 2.2.

1.2.3 Improvements to DQN

DQN and Q-learning has a well known problem with overestimating action values. The agent often ends up finding local maximum instead of solving the environment.

Double DQN is a technique to counter the overestimation of action values, this is done by using two neural networks. One network is used to select actions whilst the other used to evaluate the actions.

There has also been multiple improvements to learning speed, performance and efficiency. Some of these include Prioritised Experience Replay, multi-step learning,

Dueling DQN and Distributional DQN.

RAINBOW DQN is an architecture that combines multiple improvements. These extensions are Double-, Dueling-, Noisy-, Distributional DQN, multi-step and prioritised experience replay.[11]

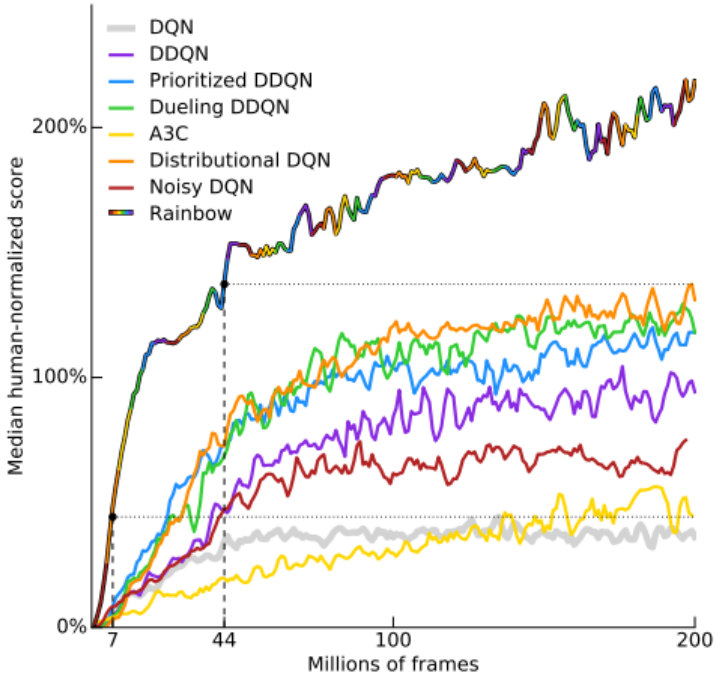


Figure 1: RAINBOW DQN compared to other reinforcement learning architectures on the Atari 2600 benchmark. (Hessel et al. 2017 p.1)

As shown in Figure 1, RAINBOW DQN achieves a much higher performance compared to other baseline architectures. The RAINBOW architecture is the main inspiration for this thesis.

2 Background

2.1 Splendor

Splendor is a board game in which you take the role as a merchant during the Renaissance. During the game you acquire gemstones represented by tokens which you use as a resource to acquire developments to produce new gemstones. Your developments further reduce the cost of future purchases as well as attracting nobles. The goal of Splendor is to reach fifteen points by collecting developments and nobles.

Splendor is designed by Marc André and was released and published in February 2014 by SPACE Cowboys[12].

2.1.1 Game Contents

The contents of the game box consists of 90 development cards, 40 tokens and 10 noble tiles. The cards are divided in levels containing 40 level 1 cards, 30 level 2 cards and 20 level 3 cards. The tokens are divided by color containing 7 green, blue, red, black and white tokens as well as 5 gold tokens.

When the everything is laid out and the game has started the board will look similar to what can be seen in Figure 2. In the top right corner of each card is a gemstone which represents a token, in the top left corner is a number which represents how many points the card is worth.

Figure 2: The game board of Splendor for two players

2.1.2 Game Rules

Splendor can be played by two to four players. The goal of the game is to be the first player to reach fifteen points. This is done by buying cards using tokens and gathering nobles using cards. Each player can take one action on each round. There are three actions available, reserve a card, take tokens or buy a card. A card can be reserved from any level as well as the face down cards. When a player reserves a card they are the only one that can buy said card, they also receive a golden token. A player can have a maximum of three reserved cards at a time. Tokens

can be picked from the common board in combination of up to three or two tokens of the same color, a player can have a maximum of ten tokens at any time. When a card is bought the player exchanges tokens for the card. The cards have a color which represents a token, when a card is bought it counts as a "permanent token" for future purchases. A card can also reward the player with zero to five points. If the starting player reaches fifteen points first, the other players get to take an extra turn, ensuring all players take the same amount of actions.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a machine learning paradigm and is about learning how to act in an environment through trial and error. By interacting with the environment, the agent receives rewards and/or penalties which in turn change how the agent acts. Receiving a reward makes the agent favour actions which took it to the reward whist a penalty makes the agent disfavour the actions. Exploration and exploitation are very important for an RL agent. Exploration is when the agent takes random actions to gain knowledge about the environment. Exploitation is when the agent utilizes past experiences to take the action deemed best in the current state of the environment.[13]

The key components in RL are rewards, states, actions, policies and value functions. A reward is given depending on the action the agent takes in a state. The state describes the current environment. An action is taken by the agent to transition from the current state to another state depending on the action chosen. The policy determines the agent's behaviour meaning the action to choose given the current state. The value function describes what states and actions are good in the long run. Unlike rewards which are given directly the value function values "paths" to bigger rewards.

A RL agent always strive to maximize the reward during a task. Until the task is completed, the agent and environment interacts through an interaction loop. Figure 3 shows how an interaction loop can look when an agent and environment interacts.

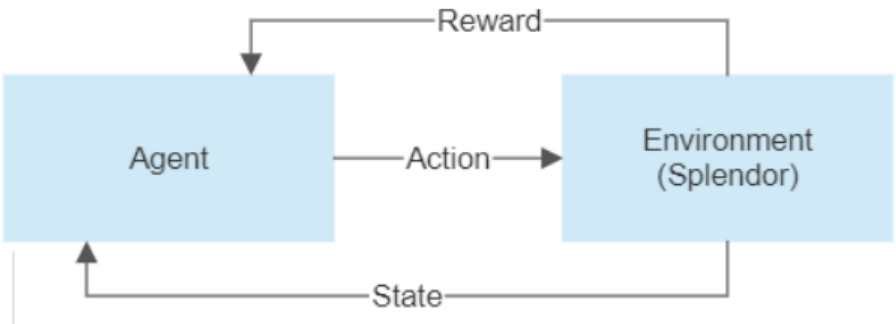
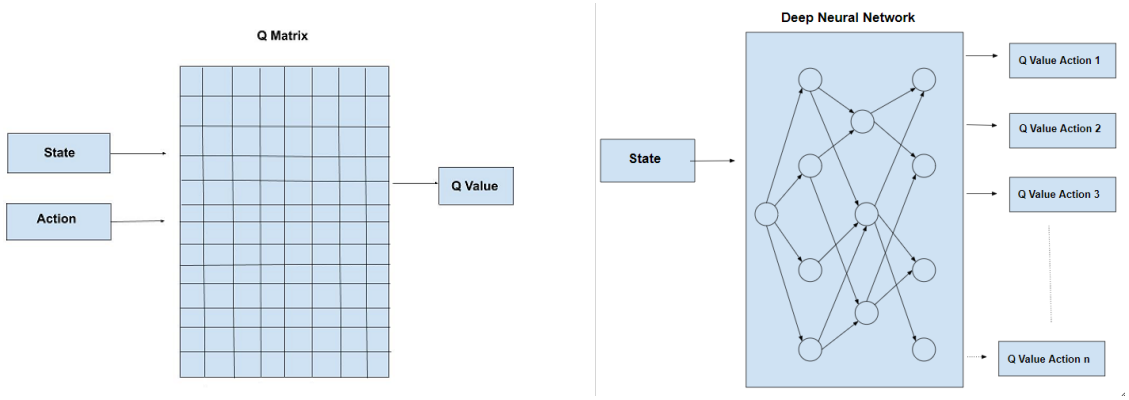


Figure 3: Interaction loop between an agent and an environment

The interactions between the agent and the environment can be represented mathematically through Markov Decision Processes (MDP)[13]. They interact with each



(a) Q-matrix (b) Deep Neural Network

Figure 4: Difference between usage of a Q-matrix and a Deep Neural Network (AlindGupta 2019 [16]).

other at time steps t. At each time step the agent chooses an action A depending on the state S. In the next time step the agent receives a reward R depending on the following state S_{t+1}. This produces a sequence of S, A, R as $S_0, A_0, R_1, S_1, A_1, R_2, S_2$ etc.

2.2.1 Q-Learning

Q-Learning is a fundamental architecture in RL which has provided a foundation for learning optimal policies as well as newer RL architectures. Q-Learning updates the values of state-action pairs, Q-values, based on observed experiences until the Q-values converge to the optimal solution. Q-Learning can be used when the environment can be modeled as a MDP which an agent can interact with through actions and rewards.[14]

2.2.2 Deep Q-Learning

Deep Q-Learning is an extension to the RL algorithm Q-Learning which integrates a deep neural network to the agent. Q-Learning has a Q matrix, this matrix takes an action and the state as input, the output is the Q value of that state-action pair. Deep Q-Learning utilizes a neural network to instead take only the state as input, the output in this case is the Q value for each action regarding the current state[15]. The difference of using a Q-matrix and a Deep Neural Network can be seen in Figure 4.

A problem with Q-Learning and Deep Q-Learning is that some action values get overestimated in certain situations. This leads to the possibility to get stuck in a local minimum or maximum which can also be called the optimizer's curse[17].

2.2.3 Double Deep Q-Learning

To combat the issue Q-Learning has with overestimating the Q-values, Double Q-Learning was introduced. This method uses two set of weights, one set of weights determine the policy and the other set of weights determine its value.

When integrating Double Q-Learning with Deep Q-Learning (DDQN) two neural networks are used, a target and a main network. The target network is initially a copy of the main network but the parameters are updated more slowly. The action is chosen by the main network while the actions are evaluated using the target network. By using DDQN the accuracy of the learned Q-values can be improved.

2.2.4 Prioritised Experience Replay

The key idea of prioritised experience replay is that transitions can be more or less redundant, surprising or relevant. Certain transitions might be more useful to the agent later rather than when the agent is in the beginning of the learning phase. By measuring the temporal difference error a transition can be prioritised based on the expected learning progress. Those transitions with a high priority get replayed more often which in theory will make the agent learn faster.

The probability for sampling a transition is calculated as $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ where p_i is the priority of transition i and k is the number of sampled transitions in the batch. There's two variants of calculating the priority for a transition, proportional or rank-based prioritisation. In proportional prioritisation the priority is calculated as $p_i = |\delta_i| + \epsilon$ where δ is the TD error and ϵ is a small constant which ensures the prioritisation is never 0. In rank-based prioritisation the priority is calculated as $p_i = \frac{1}{rank(i)}$ where rank(i) is the rank of a transition when the memory is sorted by $|\delta_i|$.

There arise a problem with bias when using PER since the distribution changes uncontrolled which in turn changes the solution the agent converges to. By introducing weighted importance sampling the bias can be corrected for. The IS weights are calculated as $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$. These weights fully compensates for the probabilities if $\beta = 1$. The loss is then calculated using $w_i\delta_i$ instead of δ_i when updating the Q-values.[18]

2.2.5 Multi-Step Learning

Multi-step learning is an architecture which extends the standard one-step learning to check multiple time steps into the future[19]. The agent considers cumulative rewards over a set n time steps before updating the policy or value function which is useful when the impact of an action isn't immediately observed. Multi-step learning is often used when a task either require planning over several actions or there is delayed consequences of an action.

2.2.6 RAINBOW

RAINBOW DQN[11] is the architecture which was the most inspirational for this project. As mentioned in 1.2.3 RAINBOW combines multiple improvements into one architecture. As shown in figure 1 RAINBOW both learns faster and reaches a higher performance than the then current baseline architectures. Hessel et al. also did an ablation study in which they removed one improvement at a time from the final RAINBOW DQN agent to see which improvements provided the most to the agent. In figure 5 it is shown that by removing either multi-step, prioritised experience replay or distributional DQN the performance takes a big hit. Therefore multi-step and PER was the most interesting to further investigate for this project.

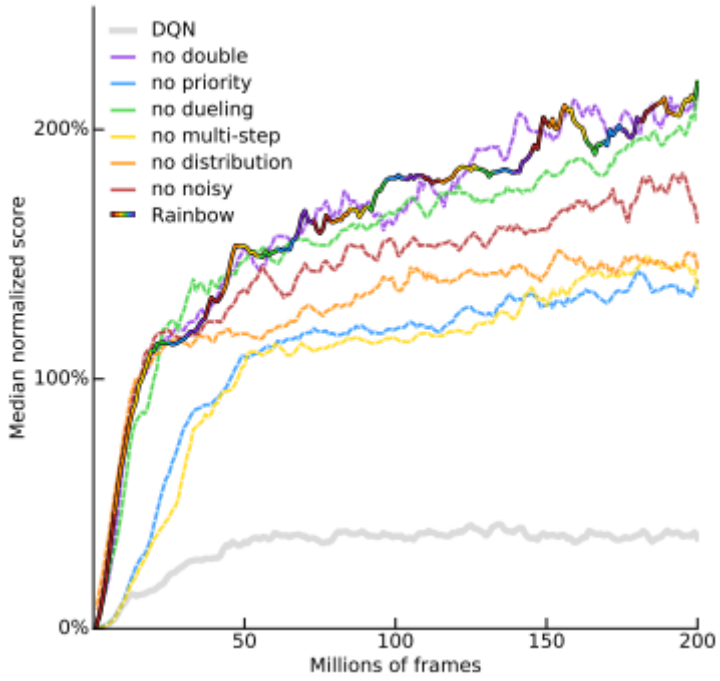


Figure 5: Comparison between RAINBOW and different ablations (Hessel et al. 2017 p.6)

3 Implementation

This chapter explains how Splendor and the different RL architectures was implemented. It will also cover training of the agents, including platforms, hyper parameter tuning and certain design choices.

The ML framework chosen for this project was PyTorch[20] which is built on Torch[21]. PyTorch supports Python, Java and C++, Python was chosen for all implementation due to most familiarity.

3.1 Splendor

The game's main loop handles user input and ensures there's no illegal input, taking card 14 when there's only 12 cards e.g. It starts by loading the board, environment and a metrics logger. There's branching depending on the type of players turn, a human player and an agent doesn't have the same flow, the human player gives input while the agent observes the environment and chooses an action. The board checks if a player has won, if this is the case, the episode is logged and the board and environment is reloaded, otherwise the current player changes and the next round starts. Otherwise the game is essentially divided into four classes, Board, Players, Cards and Decks.

The game implementation supports two actors which can be any combination of human, random or agent players.

3.1.1 Player

The player contains information about the current points, cards on hand, reserved cards and current tokens for a player. This class is responsible for ensuring that tokens are returned correctly to the board when a card is bought. The player also contains a command line interface in which a human player can enter actions. The interface first show the possible actions (Take tokens, Buy card, Reserve card, Buy reserved card and Return tokens). The interface can be seen in Figure 6, this board represents the same board as in Figure 2. After an action is chosen the human player can enter either a card id or tokens to take depending on previous choice of action.

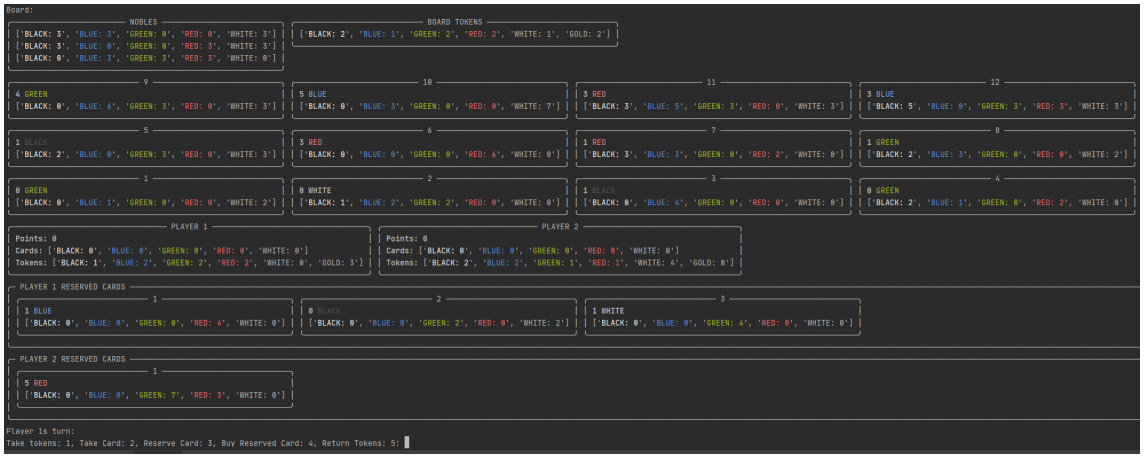


Figure 6: Interface of the board state

3.1.2 Board

The board class contains information of players, current available cards and tokens of the board. Most of the game logic is defined here, checking if an action is legal, giving points, cards and tokens to players, changing players, checking for wins as well as printing the state of the board.

3.1.3 Card and Deck

The card class contains information about the color, price and score of a card. There is also a method for encoding the information a card holds using ordinal encoding. A card before encoding might look like {'BLACK', 2, {'BLACK' : 2, 'BLUE' : 0, 'GREEN' : 0, 'RED' : 5, 'WHITE' : 0}} and after encoding the same card would be described as [0 2 2 0 0 5 0].

There's three different decks in Splendor, level one, level two and level three cards. The deck class contains information about the cards that's part of that deck. The deck class is also responsible for shuffling the cards part of that deck when a new game is started.

3.2 Environment

The environment class is the interface between the agents and the game environment. The environment consists of a state, all actions and it also has full information of the board. The state is a sequence of integers which describe the current board, this sequence gets altered by the player(s) actions.

An action is a tuple of ('Action type', identifier), (BUY_ACTION, Card), (TAKE_TOKENS_ACTION 123) e.g. The environment creates a list of all possible actions as well as all currently legal actions. By creating a list of legal actions there's an insurance that the agent can't choose an illegal action later.

3.3 Agents

3.3.1 Random Agent

The random agent is a very simple agent created so the later agents have something to train against. Random agent has all properties of a player as well as choosing a random action from the available actions. The agent chooses uniformly from the five actions using a weighted probability. If there are three available actions; (BUY_ACTION, 1), (TAKE_TOKENS, 123) and (TAKE_TOKENS, 034) the weighting will be $[\frac{1}{2}, \frac{1}{4}, \frac{1}{4}]$ meaning that both BUY_ACTION and TAKE_TOKENS have the same probability to be chosen.

3.3.2 DQN Agent

The DQN agent has all properties of a player as well as a deep neural network, optimizer, loss function and a memory buffer. The neural network consists of an input layer with 156 input nodes which represent the length of the state space, 4 hidden layers and an output layer with 371 output nodes which represent all possible actions. All layers are linear with a leaky ReLU activation function[22], there's also a dropout layer[23] between the 2nd and 3rd hidden layer. This dropout layer zeroes

elements in the input layer randomly with a probability of 20%. By using this dropout layer certain nodes in the network will be unreachable, this forces the agent to explore more nodes during training. The chosen optimizer was Adam[24].

The experience replay buffer stores previous transitions between states. The agent sample from these transitions and run them through the network which give each action a Q-value. A loss is computed by comparing the *Huber Loss* between the current Q-values and the expected max Q-values.

The step function is the DQN agents way of choosing an action. It either chooses a random available action or the best available action. This is what's called exploration and exploitation, the choice of exploration or exploitation is controlled by ϵ. ϵ starts at 1 and slowly decreases towards 0.01 during training. When the agent takes an action it's important that the action is currently available. Taking a card that's not currently on the board would be illegal e.g. To avoid this action masking is implemented, this is done by comparing all actions to allowed actions. Actions that aren't present in allowed actions will have their Q-value set to $-\infty$, when the agent is exploiting it will then never choose an illegal action as the value for such action is extremely low compared to the allowed actions.

There's several hyper parameters controlling the behaviour of the agent; gamma γ, epsilon ϵ, learning rate and a burn in. The γ decides how the agent prioritises later rewards vs immediate rewards, this is set to 0.99 which mean that the agent will value later rewards more. The ϵ decides how often the agent explores, the epsilon starts at 1 and slowly decrease towards 0.01. Burn in decides how many time steps the agent takes before the ϵ starts decreasing, this is used to give the agent a buffer of completely random actions for some experience. The learning rate influences how often new information replaces old information, a high learning rate can overshoot the minima whilst a low learning rate can get stuck in a local minima or take way to long to converge. The learning rate chosen was 25e-5 which is the same as Hasselt et al.[25] chose for their DDQN implementation.

3.3.3 DDQN

The DDQN agent uses two neural networks, a current network and a target network. As mentioned in 2.2.2 this is to avoid a common problem in Q-learning with overestimating the Q-values. The implementation of DDQN is simple, the expected max Q-values are run through the target network instead of the current network and then the loss is calculated the same way as in DQN. At certain time steps the target network is copied to the current network, Hasselt et al.[25] updates their network every 10000th time step. In this implementation the network updates every 1000th time step since the total number of steps is way lower, 50M compared to 1-2M.

3.3.4 Prioritised Experience Replay

The difference between prioritised and uniform experience replay is that the transitions get a priority depending on the TD-error. This mean that transitions with

a high TD-error get used more often than others. Since the sampling in PER is weighted and there are up to a million transitions that can be stored in memory, a SumTree was implemented to store the transitions. The time complexity for calculation cumulative sums is O(n), since the sampling happens quite frequently that's too slow. By using a SumTree the time complexity is O(logn) instead which becomes way faster during longer runs. The hyper parameters used are $\alpha = 0.6$ and $\beta = 0.4$ which is the same as Shaul et al.[18] used in their implementation for proportional PER.

3.3.5 Multi-Step Learning

The implementation for n-step Q-learning requires the use of two memory buffers. By using one buffer for n-step transitions and another for the usual transitions the indices will always be connected. Transitions isn't stored unless the n-step buffer has enough steps in it.

3.4 Training

The training of the agent have been through many iterations, starting with training on an Intel Core i5-11400F CPU to training on a NVIDIA GeForce RTX 3060Ti GPU to training on a virtual headless machine. Most of the final training was done on a headless machine using an Intel Xeon Platinum 8259CL CPU and a Tesla T4 Tensor Core GPU[26] via Amazon Web Services[27].

The agents have been trained for a different amount of games over time, between 1000 and 50000 games. Different hyper parameters has been used, there has been a lot of changes through trial and error. The agents have both been training against themselves or against the random agent.

The training parameters for the final training is shown in table 1.

Different reward structures was used and tested during training, the most simple being 1 reward if the agent wins, otherwise 0. This reward function provided OK results against the random agent, against the human player it was quickly realised that the agent took questionable actions way to often. This was actions such as taking one single token or returning tokens just to take the same tokens again. It became clear that this agent would never actually win against a human.

Another reward function which relied on the number of turns was tried, the reward the agent received was $30 - numturns$. This reward function mostly confused the agent since the reward was so different every time.

The final reward function gave the agent 5 reward for a win, to avoid the agent making obviously stupid moves a penalty of -1 was given every time the agent decided to return tokens to the board or taking one single token from the board.

Training parameters	
Parameter	Value
gamma γ	0.99
start epsilon ϵ_{start}	1
final epsilon ϵ_{final}	0.01
batch size	32
burn in	50000
learning rate	25e-5
network sync	1000
PER exclusive	
alpha α	0.7
beta β	0.5
Multi-step exclusive	
steps n	20

Table 1: Parameters during final training

4 Evaluation and Results

4.1 Evaluation

Since the goal is that the agent(s) should reach superhuman performance evaluation is a bit tricky. There is no real standard for what a human performance level is in the game Splendor. In competitive Splendor it's usually a goal to reach 15+ points in ≤ 25 turns. This is however true for experienced players, the ideal turns to win for a more casual player might not be ≤ 25.

Figure 7 shows a diagram over the number of turns to win over 7307 games Splendor played by human players on Spendee[28]. The data shows that most games last between 24-30 rounds. Two player games last slightly longer, 27.31 rounds compared to 25.52 for four player games as shown in Figure 8.

Generally players that play on Spendee are semi-experienced to very experienced players. A more casual player should probably win in a higher number of turns than a Spendee player which mean that the agent should aim to win in a higher number of turns.

The agent will be evaluated based on win ratio and turns to win against a random agent and a semi-experienced human player.

The goal is for the agent to achieve a 100% win rate against a random agent and 80-100% win rate against the semi-experienced human player. Turns to win should be ≤ 30 against both the random and human player.

The agents that will be evaluated are: DDQN, DDQN + PER and DDQN + n-step. Each agent will have it's performance measured based on:

- Win rate against random agent (1000 games)

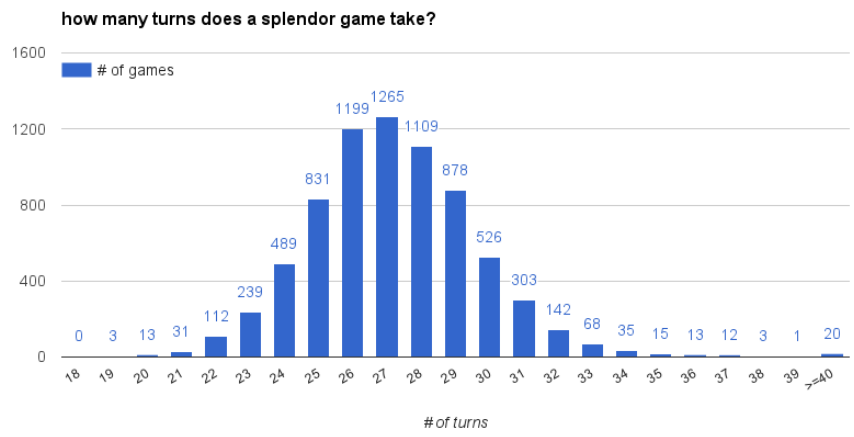


Figure 7: Number of turns to win in Splendor (Kim 2017).

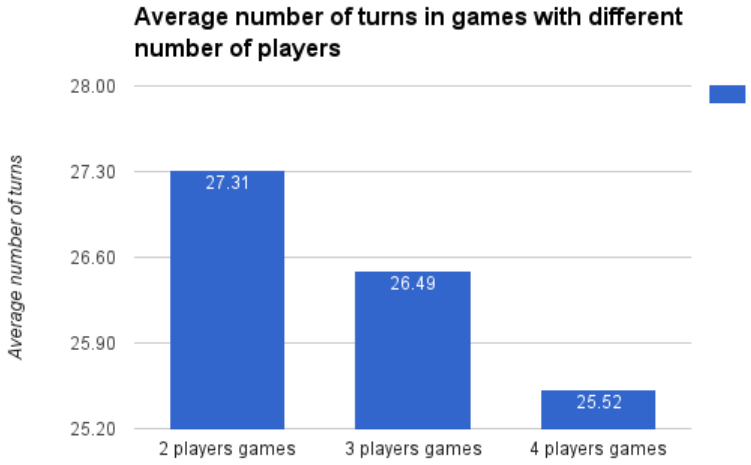


Figure 8: Average number of turns with different number of players (Kim 2017).

- Average turns to win against random agent (1000 games)
- Average score against random agent (1000 games)
- Win rate against human player (10 games)
- Average turns to win against human player (10 games)
- Average score against human player (10 games)

4.2 Results

This section presents the results of the agents evaluated against the random agent for 1000 games and a human player for 10 games.

4.2.1 DDQN

Two DDQN agents were trained for 20000 games each, during the training the agents experienced approximately 850'000 rounds. One agent was trained against the random agent whilst the other agent was trained using self-play.

Table 2 shows the results when evaluating the DDQN agent which was trained against the random agent. The win-ratio against the random agent is 96,7% which can be considered OK. The turns to win is however fairly long at 41.39 turns, this indicates that the agent shouldn't win against a human which has played the game before. This is also shown in when the agent was evaluated against the human player, the agent won 0 games and the human managed to win in around 27 turns.

DDQN Agent Performance 20000 games			
Opponent	Games won	Average TTW	Average score
Random Agent	96,7%	41.39	15.87
Human	0%	26.71	8.43

Table 2: DDQN agent trained against the random agent. Performance during evaluation against the random agent and against a human player.

Table 3 shows the results when evaluating the DDQN agent which was trained using self play. The win-ratio against the random agent is 99,7% which is to be considered good. The turns to win is significantly better at 34.21 turns, this indicates that the agent could have a chance to beat a human. When the agent was evaluated against the human player, the agent won 0 games and the human managed to win in around 27-28 turns.

DDQN Agent Performance 20000 games self-play			
Opponent	Games won	Average TTW	Average score
Random Agent	99,7%	34.21	16.13
Human	0%	27.52	10.82

Table 3: DDQN agent trained using self-play. Performance during evaluation against the random agent and against a human player.

It's clear that the DDQN agent trained using self-play perform better than the DDQN agent trained against the random agent. Neither of the agents does however reach super-human performance or even human performance.

4.2.2 DDQN + PER

Two PER agents were trained against the random agent to investigate how training length might change the result. One agent was trained for 30000 games and the other for 50000 games, they experienced around 1'300'000 and 2'100'000 rounds each. There was also a PER agent trained using self-play for 20000 games, this agent experienced around 850'000 rounds.

Table 4 shows the results when evaluating the PER agents which were trained against the random agent. For the PER agent which trained for 30000 games, the win-ratio against the random agent is 94,2% which can be considered OK. The turns to win is however fairly long at 43.69 turns, this indicates that the agent shouldn't win against a human which has played the game before. This is also shown in when the agent was evaluated against the human player, the agent won 0 games and the human managed to win in around 27 turns.

More interesting is that the PER agent which trained for 50000 games actually performed significantly worse. This agent has a mere win-ratio of 76,9% and turns to win at 47.18 turns when evaluated against the random agent. These results are considered bad. The difference in performance will be further discussed in Section 4.3.

PER Agent Performance 30000 games			
Agent	Games won	Average TTW	Average score
Random Agent	94,2%	43.69	16.29
Human	0%	27.32	7.46
PER Agent Performance 50000 games			
Agent	Games won	Average TTW	Average score
Random Agent	76,8%	47.18	15.04
Human	0%	25.18	5.21

Table 4: Two PER agents trained against the random agent. Performance during evaluation against the random agent and against a human player.

Table 5 shows the results when evaluating the PER agent which was trained using self play. The win-ratio against the random agent is 99,8% which is to be considered good. The turns to win is also significantly better at 35.23 turns, this indicates that the agent could have a chance to beat a human. When the agent was evaluated against the human player, the agent won 0 games and the human managed to win in around 26 turns.

PER Agent Performance 20000 games self-play			
Opponent	Games won	Average TTW	Average score
Random Agent	99,8%	35.23	16.34
Human	0%	26.17	11.41

Table 5: PER agent trained using self-play. Performance during evaluation against the random agent and against a human player.

Similar to the DDQN agents it's clear that the agent trained using self-play perform better. Neither of the agents does not manage to reach super-human performance.

4.2.3 DDQN + Multi-Step

The Multi-Step agent were trained against the random agent for 10000 games. The results of the evaluation can be seen in Figure 6. Since the results when evaluating

against the random agent were incredibly bad this agent never got the chance to be evaluated against a human. The win-rate was 46% and the turns to win was 350 turns which is unbelievably slow. There is something wrong in the implementation of this agent, it will thus be excluded from the discussion.

Multi-Step Agent Performance 10000 games			
Opponent	Games won	Average TTW	Average score
Random Agent	46%	352.76	10.35
Human	—%	—	—

Table 6: Multi-Step agent trained against the random agent. Performance during evaluation against the random agent.

4.3 Result Discussion

Unfortunately none of the agents does well against a human, they do however have an OK performance against the random agent. An interesting observation by looking at the log files is that all agents heavily favours reserving cards in the start of the game. This is necessarily not a bad strategy, except that the cards reserved and future token picks doesn't really have any correlation. It seem like the agents reserve cards just to get access to the gold tokens.

The agents take three tokens close to never, it's almost always two tokens. This is not a good strategy at all since it just slows the agents tempo down. The agents does however favour two colors, this is generally a good strategy but by playing using this strategy you need to be flexible which the agents fail to be. It seem that the agents becomes over fitted to pick two colors constantly, this can also be seen in the case of PER trained against the random agent, where training for an even longer time the performance become worse. It is clear by looking at the log files that the agent would close to never win against a human that played more than one game.

5 Conclusion

5.1 Discussion

The goal of this project was to implement an agent which can understand and play a board game and eventually reach super human performance. The agents implemented seem to understand the environment in Splendor, unfortunately they seem to have no chance against a human player, against a random player the agents have a pretty good performance.

The goals mentioned in 1.1 have somewhat been accomplished. The game Splendor has been implemented and the implemented agents can play the game as well as understand it. Since the performance reached could be similar to a human which has never played the game before, super human performance seems far away.

There's a few things that could be further improved in this project. Firstly Multi-Step should be properly re-implemented and trained using self-play similar to the other agents. It would also be interesting to see how an agent which utilizes both PER and Multi-Step would perform compared to the other agents. The structure of the neural network, optimizer and loss function and their combinations can be further explored. The training could also be done differently. It would be interesting letting the agents observe human play and learn from that. Since the training is only against themselves and the random agent the knowledge of certain strategies and counter-strategies seems to be hard to learn. The evaluation step could further be improved by having the agents be evaluated against more humans with different skill. The agents have currently only been evaluated against a single human with above average skill. It would be interesting to see if they could have a chance against a human of lesser skill.

5.2 Contextual Aspects

The social impact of this project is very limited in comparison to what other RL usages might have. None the less RL and especially AI as a whole is a heated topic today. As a society we currently have AI that's able create art[29], music[30], answer our questions[31] and impersonating politicians[32] among others. The social impact of this project might stretch as far as a slight energy waste since training RL agents requires a high amount of computing power which in turn consumes energy. It might be a stretch but if the agents were to reach super-human performance the project could be used for profiling behaviour in humans. To consistently win against a human the human's behaviour would have to both be predicted and interpreted. An agent with these properties could be used in a harmful or devious way.

The economical impact of this project is limited outside of a potential future development at Piktiv.

6 Self Reflection

6.1 Knowledge and Comprehension

Prior knowledge of both machine learning and reinforcement learning was very limited and basic. Throughout this thesis the knowledge of especially reinforcement learning has increased. A deeper knowledge has been gained of Deep Q-Learning and certain improvements, like Prioritised Experience Replay and Multi-Step Learning among others, of the architecture.

I think relevant knowledge for technologies, theories and methods are demonstrated, especially in the implementation part of the thesis. The written report might be a bit bare boned when it comes to knowledge demonstration.

6.2 Proficiency and Ability

During this thesis knowledge has also been gained in how to search for and read scientific and technological papers. Problems from such articles has been related to this thesis in a way I think is clear. A lot of articles has also been excluded for not being relevant enough to the problem at hand.

The project has been presented both orally and written. The report is a bit bare bonded and doesn't completely justify the work behind the project. I feel that the report is well structured, relevant and comprehensible overall. The results has been evaluated and discussed in what I feel is a correct way.

6.3 Values and Attitude

The work done has been put in both a social and a business context. There has been weekly meetings with both the supervisor at Piktiv and at Örebro University. The meetings at Piktiv has been SCRUM-like where we each week plan what work should be done that week. The meetings at Örebro University has mostly been to go through what's been done the previous week and how well the time plan is held. In hindsight, I wish that I asked more questions but the questions asked has been well answered. The time plan was going according to plan in the beginning of the thesis. Toward the end I fell a bit behind, especially with the written report. If I had to redo the thesis I would absolutely put more focus on the written report earlier. The final product is a software in which different agents can either be trained or played against as both a human or another agent. The software is usable and reliable for those purposes.

References

[1] Imran Ghory. Reinforcement learning in board games. *Department of Computer Science, University of Bristol, Tech. Rep*, 105, 2004.

[2] Wikipedia contributors. Stockfish (chess) — Wikipedia, the free encyclopedia, 2023. [Online; accessed 21-May-2023].

[3] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[5] Linus Strömberg and Viktor Lind. Board Game AI Using Reinforcement Learning.

[6] Carina Huchler. AN MCTS AGENT FOR TICKET TO RIDE.

[7] 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.

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

[9] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-Carlo Tree Search in Settlers of Catan. In H. Jaap van den Herik and Pieter Spronck, editors, *Advances in Computer Games*, pages 21–32, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[10] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[11] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning, October 2017. arXiv:1710.02298 [cs].

[12] SPACE Cowboys. Splendor. https://www.spacecowboys.fr/splendor-english.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[14] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[16] AlindGupta. Deep Q-Learning, June 2019. GeeksforGeeks, Section: Machine Learning.

[17] James E. Smith and Robert L. Winkler. The Optimizer's Curse: Skepticism and Postdecision Surprise in Decision Analysis. *Management Science*, 52(3):311–322, March 2006.

[18] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

[19] Kristopher De Asis, J. Fernando Hernandez-Garcia, G. Zacharias Holland, and Richard S. Sutton. Multi-step reinforcement learning: A unifying algorithm. *CoRR*, abs/1703.01327, 2017.

[20] PyTorch. https://www.pytorch.org.

[21] Torch | Scientific computing for LuaJIT. http://torch.ch/.

[22] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[23] Pierre Baldi and Peter J Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26, 2013.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[25] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[26] NVIDIA T4 Tensor Core GPUs for Accelerating Inference. https://www.nvidia.com/en-us/data-center/tesla-t4/.

[27] Cloud Computing Services - Amazon Web Services (AWS). https://aws.amazon.com/.

[28] Kim. Splendor strategy - data analysis - part 1, 2017. https://spendee.mattle.online/lobby/forum/topic/mzXQmzjCBmyC56Dgx/splendor-strategy-data-analysis-part-1.

[29] DALL·E 2. https://openai.com/dall-e-2.

[30] Chloe Veltman. When you realize your favorite new song was written and performed by ... AI. *NPR*, April 2023.

[31] Luciano Floridi and Massimo Chiriatti. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*, 30(4):681–694, December 2020.

[32] Mika Westerlund. The Emergence of Deepfake Technology: A Review. *Technology Innovation Management Review*, 9(11):40–53, 2019. Place: Ottawa Publisher: Talent First Network.