

*Bachelor Thesis*

Reinforcement Learning for board game Crypt

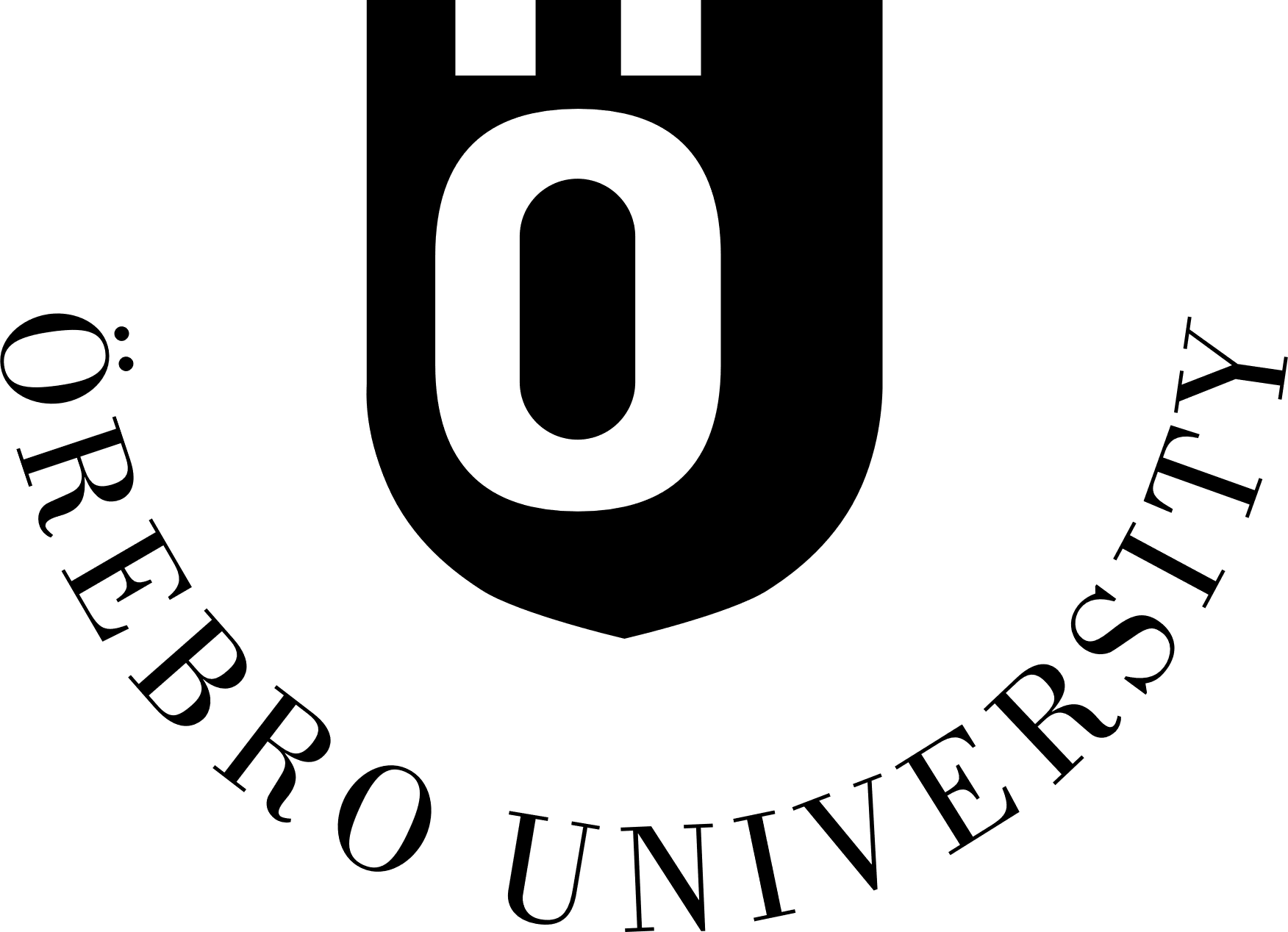
Pontus Wallquist

*Computer Science*

Studies from the School of Science and Technology at Örebro University

Örebro 2022

*Studies from the School of Science and Technology at Örebro University*



Pontus Wallquist

Reinforcement Learning for board game Crypt

**Supervisors: Simon Johansson, Piktiv AB**

**Fabien Lagriffoul, Örebro University**

**Examiner: Stephanie Lowry, Örebro University**

© Pontus Wallquist, 2022

# Abstract

Abstract is a short overview of the content of the work. The main goal of abstract is to give a reader an idea about the work, without the need to read it all. Thus, the abstract should be concise, but at the same time concrete on the content of the work.

Typically, abstract is not more than one page long, and presents the work in a brief and concise way. Often abstract follows the outline of the work presents a problem, method, and results.

### Keywords

Template, BSc Thesis, Computer Science, Computer Engineering

# Contents

[Abstract 1](#_Toc122619743)

[Contents 2](#_Toc122619744)

[1 Introduction 3](#_Toc122619745)

[1.1 Problem Formulation 3](#_Toc122619746)

[1.2 Outline 4](#_Toc122619747)

[2 Background 5](#_Toc122619748)

[3 Implementation 10](#_Toc122619749)

[4 Results 11](#_Toc122619750)

[5 Conclusion 12](#_Toc122619751)

[References 13](#_Toc122619752)

# 1 Introduction

The purpose of this project is to examine the viability of artificial players in stochastic board games. Mainly the board game Crypt [1] created by Andrew Nerger and Jeff Chin. This thesis will document the implementation of the game as well as the agents that will learn to play it.

## 1.1 Problem Formulation

The idea for this project comes from a company called Piktiv AB. They specialize in game development and software development. They see a potential for creating tougher adversaries in video games and board games through machine learning methods. They would like to see artificial players be able to play a videogame on a near human-level in the future.

The goals of this project are as follows:

1. Build the chosen board game as a video game. However due to time limitations, the goal is not to create a fun or good-looking game. The focus will instead be on implementing the game logic so an artificial player can play it.
2. Build a Reinforcement Learning Agent that will learn to play the game through self-play.
3. Evaluate the agent by playing games against random agent, another RL agent and human opposition.
4. Document the development and results in the form of a bachelor thesis.

## 1.2 Outline

The rest of this thesis is organized as follows:

* Chapter 2 will give an overview of the background work that led up to this project and introduce the reader to reinforcement learning concepts. This chapter will also provide the rules and details of the board game Crypt.
* Chapter 3 contains the development and implementation of the game as well as the reinforcement learning agent. At the end of this section the reader will understand the game and the chosen algorithm Q-Learning.
* Chapter 4 will present the results of the training and testing of multiple agents in the game environment.
* Chapter 5 contains the conclusions drawn from the results and a discussion on the future viability of reinforcement learning methods in board game AI.

# 2 Background

**2.1 Reinforcement Learning**

Reinforcement learning is the practice of learning by trial and error [2]. An agent can learn to navigate environments without any previous knowledge of the domain. It will map specific situations to maximize a certain reward signal. However, the agent must discover for itself which actions return the highest reward by trying them. Reinforcement learning is well suited to stochastic problems or simply problems with an immense search space far too big to be solved with dynamic programming [3].

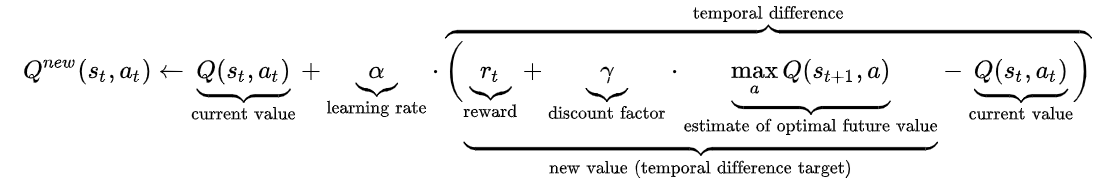
In 2017 the Google DeepMind team made a breakthrough in reinforcement learning and its use in board games with challenging domains. They created an agent known as AlphaGo [4]. Which became famous for defeating the world champion in the Chinese board game of Go. It achieved a level of superhuman performance previously unheard of. The game of Go is notorious for having one of the biggest board game domains in the world. Since then, there has been a growing popularity in creating reinforcement learning agents for more complex video games and board games. In 2019 the team at OpenAI created an agent capable of reaching superhuman performance in competitive esports games [5]. The game they chose was Dota 2, which is a multiplayer real-time strategy game. Their agent was able to defeat the current Dota 2 world champions in a best-of-three match.

The OpenAI team explains in the introduction of [5], that the long-term goal of artificial intelligence is to solve advanced real-world challenges and that games have served as steppingstones to this for decades. I believe it will continue to do so for many years to come.

These are some of the most successful AI agents out today and they have helped inspire more students and researchers to go into the field of artificial intelligence and reinforcement learning.

The agent created for this project will be using an iteration of the Q-learning algorithm first introduced in 1989 by Watkins [6]. The algorithm works by building a map between rewards and actions in an iterative manner. The action values will be called Q-values, and they will be stored in a Q-table. Good actions will obtain high Q-values and vice versa for bad actions. The Q-table will be saved and updated throughout the training of the agent, further improving the predictions for each iteration.

The algorithm works by adding the current Q-value with the chosen learning rate *α*, multiplied by the reward *r* from the last action, added with a discount factor **γ,** that’s multiplied with an estimate of the optimal future value max*Q (s, a),* subtracted by the current Q-value. The equation can be seen below in figure 2.1.



*Figure 2.1 Q-learning algorithm*

In 2010 Hasselt published a paper called Double Q-Learning [7]. He explains that the original Q-learning algorithm has poor performance when training agents in stochastic environments. Instead, he suggests the use of two Q-value functions, which was proven to increase the performance, especially in stochastic environments.

Hasselt later reiterated on his algorithm in [8]. He added Artificial Neural Networks to combine with the Q-learning algorithm, which in turn improved the estimation of actions and the convergence rate. It works by implementing two neural networks that work together. One network estimates the Q-value of all actions while the other network evaluates which action to take based on the first networks predictions. Since this paper was published, the Double Deep Q-Network (DQN) algorithm has been the go-to for implementing reinforcement learning agents with Q-learning. This is also the algorithm used for the agent in this project.

**2.2 Crypt**

The original game can be played by 2-8 people. However, the scope of the game has in this project been scaled down to a 2-player game. This decision was made to ensure this project and thesis would be finished on time.

The board game Crypt contains the following content in a 2-player game:

* 24 unique Treasure cards, where each card has a coin value between 1-4 and one of the 6 card types. A card facing up will reveal both the type and the coin value and if a card is facing down, only the type will be visible.
* 6 Servant dice, each player starts with 3 servant dice. These are later used for claiming Treasure cards on the board. The dice are also rolled when collecting the cards to determine if the servants will become exhausted or not.
* 6 Collector cards, where each card is a representative of the 6 card types available in the game. Each card also has a bonus that can be received by a player, should the player own enough treasure cards of the corresponding type to fulfill the requirement. Each card also has an A side, and a B side with different requirements and bonuses. This project is only using the A side.
* 1 Torch card, to indicate which player plays first. This is passed between the players throughout the course of a game.
* The card types and their respective collector’s requirement and bonus:
  + Remains
    - At any time, flip 2 Remains cards face-up to recover 1 exhausted servant die.
  + Idol
    - During the Collect phase, flip an Idol card face-up to re-roll one of your dice.
  + Jewelry
    - At game end, each player with 2 or more Jewelry cards scores their highest valued Jewelry card twice.
  + Manuscript
    - At game end, players with 2 or more Manuscript cards score each of their Manuscript cards as 4 instead of the value on the card.
  + Pottery
    - At game end, players with 2 Pottery cards score 2 bonus coins, 3 Pottery cards score 4 bonus coins, and 4 or more Pottery cards score 8 bonus coins.
  + Tapestry
    - At game end, the players whose combined Tapestries are worth the most score 5 bonus coins. Tied players each score the full bonus.

**How to play**

At the start of a new game, the 24 treasure cards will be shuffled and turned facing down in a deck. Each player will have 3 servants placed in front of them along with a collection that will hold all treasure cards collected by a player. The game board will hold 3 treasure cards at a time and pile for exhausted servants. The game itself is divided into four phases.

**I: Reveal phase**

Draw 3 treasure cards from the deck and place them in a row on the board. Place two cards facing up and one facing down.

**II: Claim phase**

The player with the torch card plays first and can perform one of the following actions:   
  
1. Claim a card

Place any number of servants on the desired treasure card, choosing a value of the dice and placing that value facing up. This will assign the servants effort value. The higher the value, the more likely the servant will be exhausted. Multiple servants can be placed on the same card if they have the same effort value. Subsequent players may choose to claim unoccupied treasure or push an opponent’s servants off a card by placing a higher total effort value on the card. Servants pushed off a card will be returned to their owner.

2. Recover

Skip a turn and take back all your exhausted servants from the exhausted pile.

3. Use a collector bonus

If a player owns enough cards to fulfill a collector’s requirement, they can choose to use that collector’s bonus, assuming the collector has a “usable” bonus. In this case only the Remains and Idol collectors will have bonuses that can be called upon.

The player with the torch will also take the last turn. When claiming a card on the last turn, the player can only place servants on one card. They can still recover or use a collector bonus.

**III: Collect phase**

Discard any unclaimed treasure cards. Add claimed cards to respective player’s collection. Roll your servant dice placed on the treasure cards. The dice needs to be rolled equal or above their effort value to be retrieved, otherwise the dice becomes exhausted and added to the exhausted pile until an action to retrieve the servant has been taken. If a player had all their servants pushed off, they will recover their exhausted servants from the exhausted pile.

**IV: Pass the torch phase**

Pass the torch card to the other player and repeat phases I-IV until the deck is empty.

Player’s final score is determined by adding the following:

* Coins from the collected treasure cards
* Bonus coins from collectors
* 1 coin for each unexhausted servant.

# 3 Implementation

**3.1 Crypt implementation**

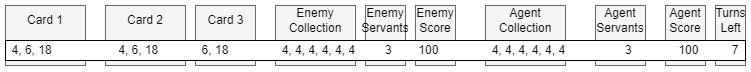
Environment architecture and design choices.

**3.2 Agent implementation**

**3.2.1 Encoded environment**

Since a reinforcement learning agent cannot take input from a visual representation in a terminal window. The game environment has been encoded as a numerical representation for the agent to be able to interpret the environment. The encoded environment is based on all the information that a human can get from looking at the game board during play. The agent should know exactly as much about the environment as a human player would. The agents’ input is therefore realized as an array of 25 separate integers. The figure below shows the input array with the maximum value each index can take.

1 2 3 4 5 6 7 8 9 10

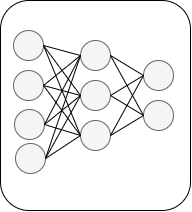


*Figure 3.3 Agent input array*

1. The first three integers represent the current card placed on the board place 1. They represent the coin value of the card ranging between 1-4, the card type ranging between 1-6 and the current placed bid which can range between 0-18.
2. Same goes for the card placed on the board place 2.
3. Card 3 however according to the game rules will be placed face down and therefore not show it’s value. It only reveals the type and the current bid to the players.
4. Enemy players collection. Six of these to represent each unique card type. There are only 4 cards of each type so these values can range between 0-4.
5. Servants available to the enemy. Range between 0-3.
6. The current enemy score. Range between 0-100.
7. The agent’s collection.
8. Servants available to the agent.
9. Agent current score.
10. Turns left until deck is empty. Range between 0-7.

**3.2.2 Agent architecture**

As stated in Chapter 2, the agent in this project will use an algorithm known as Double Deep Q-Network [8] which is utilizing two neural networks to perform actions in the environment. (Expand on DQN)

 *Figure 3.4 Dense neural network with 3 layers*

The neural networks architecture is built with an input layer consisting of 25 neurons, one hidden layer of 40 neurons and an output layer of 56 neurons. Each one of these output neurons represent a unique action that can be taken in the game. The input layer is the encoded environment for the current game state, every index in the array corresponds to one neuron in the layer. Every layer in the model is dense, meaning each neuron is connected to all other neurons in the previous layer. The hidden layer is using the *ReLu* [10] activation function while the output layer is using the *linear* [11] activation function. The neural networks also use the *Mean Squared Error* [12] as the loss function and an optimizer called *Adam* [13]. This loss function and optimizer have become somewhat standardized in machine learning practices of late and seemed the best choice for this project as well.

The algorithm is making use of the greedy epsilon policy which is more commonly referred to as exploration vs exploitation [2]. It works by taking random actions based on the value of an epsilon variable and decreasing this epsilon every time we take an action. By doing this we make sure to take a lot of random actions in the beginning also called exploring but over time we take less random actions and more informed decisions based on previous experiences which is also called exploiting.

**Reward system**

The rewards function is designed to give feedback to the agent during it’s training phase. Each action will have a reward associated with it, there’s also a reward at the end of the game which will alter depending on if the agent has won the game or not. The rewards for claiming a card in this implementation have already been defined by the game rules. The bonus action and the recover action have a constant reward while the claim action has a more fluid reward equation depending on if a collector’s requirement is fulfilled with the action. The end of game reward is the difference in score between the agent and the enemy. Meaning winning big is always preferable and gives a big reward while losing gives a negative reward.

Rewards for each type of action:

Claim 🡪 coin value of the claimed card + potential collector’s bonus

Recover 🡪 -10, -20 or -30 (depending on if the agent has 0,1 or 2 servants available)

Remains Bonus 🡪 10

End of game 🡪 (agent score – enemy score)

**Training the agent**

# 4 Results

Results from games (charts and tables).

# 5 Conclusion

Conclusion from results.

Discussion regarding future RL methods for stochastic board games.

# References

1. Andrew Nerger and Jeffrey Chin. R2i Games. Crypt. 2018 <https://www.kickstarter.com/projects/roadtoinfamy/crypt-1>  
   Rules: <https://roadtoinfamy.com/rules/Crypt_booklet_printerFriendly.pdf>
2. Richard S. Sutton and Andrew G. Barto. Reinforcement Learning, An Introduction. MIT Press, 1998 <https://redirect.cs.umbc.edu/courses/graduate/678/spring17/RL-3.pdf>
3. Ronald A Howard. Dynamic programming and markov processes. 1960.
4. David Silver et al. Mastering the game of Go without human knowledge. 2017 Nature, 550:354–359
5. OpenAI, Christopher Berner et al. Dota 2 with Large Scale Deep Reinforcement Learning. 2019 <https://doi.org/10.48550/arXiv.1912.06680>
6. Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989
7. Hado van Hasselt. Double Q-learning. 2010 <https://papers.nips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html>
8. Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning, 2015 <https://arxiv.org/abs/1509.06461>
10. Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), 2018.
11. Tomasz Szandala. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. CoRR, abs/2010.09458, 2020.