EXTRACTING EXPLICIT RELATIONAL INFORMATION FROM A NEW RELATIONAL REASONING TESTBED WITH A LEARNING AGENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
MIDDLE EAST TECHNICAL UNIVERSITY
BY

FARUK KÜÇÜKSUBAŞI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF MULTIMEDIA INFORMATICS

JULY 2021

EXTRACTING EXPLICIT RELATIONAL INFORMATION FROM A NEW RELATIONAL REASONING TESTBED WITH A LEARNING AGENT

Submitted by FARUK KÜÇÜKSUBAŞI in partial fulfillment of the requirements for the degree of **Master of Science in Modeling and Simulation Department, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics, METU** __________________

Asst. Prof. Dr. Elif Sürer
Head of Department, **Modeling and Simulation, METU** __________________

Asst. Prof. Dr. Elif Sürer
Supervisor, **Modeling and Simulation, METU** __________________

Examining Committee Members:

Assoc. Prof. Dr. Erdem Akagündüz
Modeling and Simulation, METU __________________

Asst. Prof. Dr. Elif Sürer
Modeling and Simulation, METU __________________

Prof. Dr. Haşmet Gürçay
Computer Engineering, Hacettepe University __________________

Date:

Name, Last name : **FARUK KÜÇÜKSUBAŞI**

Signature : ________________

ABSTRACT

EXTRACTING EXPLICIT RELATIONAL INFORMATION FROM A NEW RELATIONAL REASONING TESTBED WITH A LEARNING AGENT

Küçüksubaşı, Faruk
MSc., Department of Modelling and Simulation
Supervisor: Assist. Prof. Dr. Elif Sürer

July 2021, 70 pages

In recent studies, reinforcement learning (RL) agents work in ways that are specialized according to the tasks, and most of the time, their decision-making logic is not interpretable. By using symbolic artificial intelligence techniques like logic programming, statistical methods-based agent algorithms can be enhanced in terms of generalizability and interpretability. In this study, the PrediNet architecture is used for the first time in an RL problem, and in order to perform benchmarking, the multi-head dot-product attention network (MHDPA) was used. By using the PrediNet module, relational information among the objects in the environment can be extracted explicitly. This information is in a form that can be processed in logic programming tools, and the network becomes more interpretable. In order to measure the relational information extraction performances of these two methods, a new test environment, relational-grid-world (RGW), is developed. RGW environment can be generated procedurally from objects with different features, pushing the agent to make complex combinatorial selections in this environment. In the performed tests and the RGW environment, a baseline environment called Box-World is used for comparing both environments and networks separately. The results show that both MHDPA and PrediNet architecture have similar performances in both environments, and the RGW environment is able to measure the relational reasoning capacity of the networks.

Keywords: Reinforcement learning, relational reinforcement learning, relational reasoning, relation networks, attention networks

ÖZ

YENİ BİR İLİŞKİSEL AKIL YÜRÜTME TEST ORTAMINDAN BELİRGİN İLİŞKİSEL BİLGİLERİ BİR ÖĞRENME AJANI İLE ÇIKARMAK

Küçüksubaşı, Faruk
Yüksek Lisans, Modelleme ve Simülasyon Anabilim Dalı
Tez Yöneticisi: Dr. Öğr. Üyesi Elif Sürer

Temmuz 2021, 70 sayfa

Son çalışmalarda, pekiştirmeli öğrenme (RL) ajanları görevlere göre özelleşmiş şekillerde çalışmaktadır ve çoğu zaman karar verme mantıkları yorumlanamamaktadır. Mantık programlama gibi sembolik yapay zeka teknikleri kullanarak, istatistiksel yöntemlere dayalı aracı algoritmaları genelleştirilebilirlik ve yorumlanabilirlik açısından geliştirilebilmektedirler. Bu çalışmada, PrediNet mimarisi ilk kez bir RL probleminde kullanılmış ve kıyaslama yapmak için "multi-head dot-product attention network" (MHDPA) kullanılmıştır. PrediNet modülü kullanılarak, ortamdaki nesneler arasındaki ilişkisel bilgiler açıkça çıkarılabilmektedir. Bu bilgiler, mantık programlama araçlarında işlenebilecek bir formdadır ve ağ daha yorumlanabilir hale gelmektedir. Bu iki yöntemin ilişkisel bilgi çıkarma performanslarını ölçmek için yeni bir test ortamı, "Relational-Grid-World" (RGW) geliştirilmiştir. RGW ortamı, farklı özelliklere sahip nesnelerden prosedürel olarak oluşturabilir ve bu ortamda ajanı karmaşık kombinatoryal seçimler yapmaya zorlamaktadır. Gerçekleştirilen testlerde ve RGW ortamında, hem ortamların hem de ağların ayrı ayrı karşılaştırılmasını sağlamak için Box-World adı verilen bir referans ortam kullanılmaktadır. Sonuçlar, hem MHDPA hem de PrediNet mimarisinin her iki ortamda da benzer performanslara sahip olduğunu ve RGW ortamının ağların ilişkisel muhakeme kapasitesini ölçebildiğini göstermektedir.

Anahtar Sözcükler: pekiştirmeli öğrenme, ilişkisel pekiştirmeli öğrenme, ilişkisel akıl yürütme, ilişki ağları, ilgi ağları

in memory of Mehmet Dursunoğlu

ACKNOWLEDGMENTS

I would like to express my gratitude and respect to my advisor Assist. Prof. Dr. Elif Sürer for all her support and efforts during my three years of graduate studies. She always encouraged me to work, thanks to the trust she showed in me and the motivational speeches she gave. In these difficult days with the pandemic in the world, she never allowed our work to be interrupted, and I was able to complete my work.

I would like to thank my professors and friends with whom we always exchanged ideas during my studies.

I would like to thank my colleagues and managers, whose support I always felt during my graduate study.

Finally, I would like to thank my mother Şükran Küçüksubaşı and my father Nadir Küçüksubaşı for the love and support they have given me so far. I would like to thank my brother Fatih Küçüksubaşı for our brainstorming, which I have used very much in my studies. I would also like to thank Esma Sena Keçeci for her support and understanding; I could not complete this work without her.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
BW	Box-World
CNN	Convolutional Neural Network
DL	Deep Learning
DRL	Deep Reinforcement Learning
GPU	Graphics Processing Unit
IMPALA	Importance Weighted Actor-Learner Architecture
LSTM	Long Short Term Memory
MDP	Markov Decision Process
MHDPA	Multi Head Dot Product Attention
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
PGM	Procedurally Generated Matrices
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RGW	Relational-Grid-World
RL	Reinforcement Learning
RN	Relational Network
RPMs	Raven’s Progressive Matrices
TD	Temporal Difference

CHAPTER 1

INTRODUCTION

1.1. Problem Statement

Most complex decision-making and optimization problems cannot be solved with analytical methods or provide time-efficient and exact solutions. In order to overcome this issue, there are problem-specific numerical methods. However, these tools may not reach the global optimum solution if the problem has a high order state-space system. Even for low-dimensional systems, there is a risk of finding a local optimal solution. Therefore, there are some analytical and numerical tricks to decrease the difficulty of the problems, such as linear approximation and random search. However, these methods are not sufficient for some of the nonlinear and high-dimensional systems.

There is not much major progress on numerical techniques in the current literature, but there are significant improvements on computer hardware and data acquisition sources. These improvements with some numerical tricks bring significant changes to high-dimensional data processing studies. Because of the mathematical structure of the method, they are called Deep Neural Networks. These networks can quickly process nonlinear and high-dimensional data (image, sound, and text). However, these networks are often task-specific, and they are unsuitable for interpretability. Apart from these, they are data inefficient compared to classical optimization methods. This thesis focuses on data inefficiency, generalizability, and interpretability problems of these methods.

Another problem is that if the problem comes from real-life applications, there should be a sufficient model of the system of interest to solve it. Games can model simple everyday life decision-making problems. The games are the easiest ways of generating data for well-defined decision-making problems because an agent's (controller or intelligent unit of the system) action in the environment can be guided easily by giving a reward or penalty. The optimization algorithm that drives the agent (training) will find optimal solutions to the problem, environment, or game. However, when the solution is determined based on a model, there will be some model errors. These errors will affect the performance of the optimized solutions in real life. Therefore, models like games help develop optimization algorithms before using them on actual data.

Moreover, humans make decisions based on their complex ideas and emotions, which are difficult to model. In order to model this kind of problem, huge modeling errors should be taken into account. Also, if the purpose is to understand human nature in decision-making and improve this process, this modeling method may not be suitable.

1.2. Aim of the Thesis

This thesis reduces the disadvantages of statistical decision-making algorithms such as inefficiency, generalizability, and interpretability by merging statistical methods (Deep Reinforcement Learning) and symbolic AI methods based on relational operators. This will enable us to reach a trainable agent model that can find associations between the environmental objects and their cause and effect relations. This thesis focuses on extracting more robust associative information of the environmental objects. Although determining the cause and effect relations from data is very puzzling, accessing this information brings robust and reliable decision-making algorithms.

In order to measure the performances of the decision-making algorithms, there should be test environments where experiments are done appropriately. These test environments should contain interrelated objects, and they should be procedurally producible. There are not enough simple environments to conduct relationality and causality experiments. Therefore, this thesis aims to develop an environment where relationality tests can be conducted. Then, the developed test environment will be compared with a baseline environment from the literature. Also, the baseline and developed environment will be used for training and comparing decision-making algorithms.

1.3. Contributions

Two simple environment models are used in order to generate samples for optimization algorithms. The first test environment is Box-World (Zambaldi et al., 2019), which models a sequential color matching problem. This model is used in order to test the relational information usability performance of the algorithms. It is a visual environment, so there is also a need for image processing operations. The second environment is called Relational-Grid-World (RGW), a new environment that models a path and sequential decision-making problem. These two environments are used for two relation-based reinforcement learning methods. One of them is a Relational Network with multi-head dot-product attention (MHDPA) module, also used in (Zambaldi et al., 2019). In this way, the novel RGW environment is also tested by the same network as the Box-World, and its reliability is checked. In addition to MHDPA, PrediNet (Shanahan et al., 2020) is also used for training the RL agent. PrediNet architecture is used in an image-based task in a reference study. Its network depends not only on object relations but also on scalar relational values in different relational representation dimensions. In this study, it is used for the first time in a dynamic environment. After the training process, its performance is compared with the MHDPA network. The ultimate aim of this study is to contribute to combining symbolic and statistical methods in AI studies.

The main contributions made by this thesis can be briefly summarized as follows:

- Development of the RGW[1] environment to measure the relational reasoning capacities of neural networks
- The first use of the PrediNet architecture in a reinforcement learning problem
- Comparison of PrediNet architecture with relational network architecture based on MHDPA in RGW and the Box-World environment (baseline)
- Demonstration of the functionality of the RGW environment by training two different relational network architectures in the RGW

The work reported in this thesis formed the basis of the following journal publication:

- Küçüksubaşı, F. & Sürer, E. (2021). Relational-grid-world: a novel relational reasoning environment and an agent model for relational information extraction. Turkish Journal Of Electrical Engineering & Computer Sciences, 29(2), 1259-1273.

1.4. Outline of the Thesis

In the following chapters of the thesis, studies about the subject of focus will be presented. In Chapter 2, studies in the literature on the related subjects will be presented. Later, in Chapter 3, the methodology of the used reinforcement learning model established by utilizing the literature will be explained in detail. In Chapter 4, the principles of the two environments will be shared and discussed. Then, in Chapter 5, the results of the preliminary tests for selecting some critical parameters for the agent architecture will be shared. This chapter will share and interpret the test results obtained with the final environment and model configuration. Finally, the obtained results and their contributions to the literature will be summarized, and future studies will be mentioned in Chapter 6.

[1]RGW (2020). Relational-Grid-World - The source code [online]. Website https://github.com/farukksubasi/Relational-Grid-World [accessed 22 August 2021].

CHAPTER 2

RELATED WORK

2.1. Basic Reinforcement Learning Methods

In order to find the optimum way of acting in an environment, there should be an intelligent unit that explores the environment and acts based on its observations. There are many ways of finding the optimal solution. This thesis will focus on the Reinforcement Learning (RL) method, which is improved by neural networks (Tesauro, 1995; Tesauro, 2002), and specifically the Deep Reinforcement Learning (DRL) method (Mnih et al., 2015; Mnih et al., 2019). DRL field has varying sub-methods, and most of them are developed for limited cases. There are many open problems about RL, such as the exploration-exploitation problem. This problem occurs because of the uncertainty of the observations. The agent does not know if it observes all of the patterns in the environment. Therefore, it should always decide between exploiting its past observation and trying to find new observations. Even if the agent saw the all-state space of the model, there may be some state sequences that cause different returns. Therefore, the agent should act inconsistently with its observations in order not to miss other possible solutions. However, if the agent acts too inconsistent with observation, it will not converge to any solution. This situation is called the exploration-exploitation dilemma in the RL domain. Methods have also been developed for observing unexplored areas by including the frequency of new observations in the environment into the loss function (Pathak, Agrawal, Efros, & Darrell, 2017; Reizinger & Szemenyei, 2019). In RL problems, the delay between the instants of observation and rewarding is a fundamental problem in the field because the agent will be challenged during matching observations and their potential returns. Also, in environments whose state space is relatively large, it is hard to find optimal solutions.

There are two types of learning methods for the RL algorithms. If the environment model is partially or fully pre-known, the agent can control its actions by deducting this information (state transition and reward functions), creating a policy. This method is called model-based RL. However, most of the time, the environment model is not known or accurately modeled. Moreover, the computational cost will increase exponentially. Therefore, the agent should observe the model without any pre-knowledge so that this method may take high-risk actions early in the learning process. This method is called model-free RL, and a model-free RL method is used in the study. The agent should create a memory containing past experiences in the environment. Classic RL methods use tabular methods in order to create a memory. In these tables, the agent stores state-action or state-total reward expectations. However, when the state space of the environment increases, the tabular method will not work anymore because combinations of the state will grow quadratically.

In order to solve this problem, (Mnih et al., 2015; Schaul, Horgan, Gregor, & Silver, 2015) suggest value function approximators based on deep neural networks (LeCun, Bengio, & Hinton, 2015; Schmidhuber, 2015). These approximators can convert classic state lookup tables to continuous nonlinear functions (Mnih et al., 2019). Reinforcement learning and deep neural network fusion were tested on various Atari games in Arcade Learning Environment (Bellemare, Naddaf, Veness, & Bowling, 2013). This fusion has been used previously, but it faced problems such as policy divergence. Therefore, innovative mechanisms have been used in the algorithm to avoid divergence. The most important of these mechanisms is the experience replay method (Mnih et al., 2015); the agent's experience is stored and randomly used in the training process of the neural network. In this way, the network uses more randomized data, and the training process becomes data-efficient. For practical purposes, the screenshot of the game has been down-sampled after the conversion from the RGB representation to the grayscale. Action (policy) is taken as output after passing the input to two convolutional and fully connected layers. The Q-learning method (Watkins & Dayan, 1992) is used to calculate loss function (temporal difference error). So, this is a model-free, off-policy algorithm created by using RL from end to end. This algorithm has been tested without changing the hyperparameters and architecture on seven different games. During the test, the ϵ-greedy policy was applied, and the ϵ value was decreased linearly from 1 to 0.1. At the same time to reduce the runtime, three or four frames from the game were not taken as input. When the experiments were completed, the results were compared with a human player, SARSA (Rummery & Niranjan, 1994), HyperNEAT (Stanley, D'Ambrosio, & and Gauci, 2009) algorithms, and it achieved state-of-the-art results in six games out of seven games. This algorithm, called Deep Q-Learning (DQN), successfully integrated raw-sensor data by integrating Q-learning, neural network, and experience replay methods for self-control.

2.2. Statistical and Symbolic Methods

Function approximators are not preferable in small environments because the function approximation process needs many samples, and tabular methods can quickly solve this environment. Although it has disadvantages in small environments, it can provide solutions in complex and realistic environments. Not only data inefficient but also interpretability and generalization is a challenging problem for DRL methods. Because, in order to construct a nonlinear function approximator, there should be a massive amount of data. Even if the function approximator is constructed correctly, the agent's decisions will not be traceable because numerically trained nonlinear functions are just numbers from a high-level perspective. Deep RL methods try to find associations between the states, actions, and rewards. However, there may be a simple cause-and-effect relationship between them. So, understanding the causality in the data will be a solution for the disadvantages of DRL (Garnelo & Shanahan, 2019). Classic AI algorithms are mostly hand-crafted and based on symbolic manipulations of the state, objects, and actions. These methods are better than modern methods in interpretability, generalizability, and data inefficiency. All of the decision logic is well-defined, and the boundaries of the algorithm are prior knowledge.

In a study (Garnelo, Arulkumaran, & Shanahan, 2016), the authors say that deep reinforcement learning (DRL) methods need large training datasets. Their transfer learning skills may not be sufficient, and they do not have enough abstract reasoning capabilities. They state that, also, symbolic methods cannot use rich information from the real world without using statistical methods. Moreover, RL agents can be combined with classical symbolic AI methods for the development of robust agents. Then, they propose a new agent model under the title of deep symbolic reinforcement learning domain. After introducing the network architecture of a three-stage RL agent, they present experimental results conducted on the novel toy example. In this toy example, the agent travels through the environment and interacts with two types of objects that reward positive (“x” objects) and negative (“o” objects). The absolute value of these rewards is equal. Four different test configurations were created by using this toy example. In the first configuration, there are only “o” objects in the environment. In the second configuration, there are “x” and “o” objects placed uniformly. The third configuration is the random configuration of the first configuration. Finally, the last configuration is the random positioning of the second one.

The first stage of the presented agent, called “low-level symbol generation,” is used as an object detection algorithm that uses the high-level raw inputs in the CNN layer. In the second stage, called “representation building,” some operations are performed on the frames in the environment. The first of these operations, defined as spatial proximity, calculates the Euclidean distance between the two objects in a frame. Secondly, the type transitions operation follows whether the same object turns into a different object between the two consecutive frames. By using the third operation, the number of neighboring objects around any object is obtained. In this way, it is calculated whether another object is approaching. The fourth and final operation is symbolic interactions and dynamics. By this operation, the position differences of the objects between two consecutive frames are obtained. In this way, local navigation can be provided. In the third and final stage of the agent architecture, there is a reinforcement learning method. In the article, a Q-learning algorithm that uses epsilon-greedy exploration is used as the RL method. The DQN model was used as a reference to compare the proposed agent model. When the experimental results are examined, it is seen that the DQN algorithm is more successful in uniform positioning environment configurations. However, in the case of positioning randomly, the presented agent model gives better results. In this way, it has been seen that the RL agent, which uses symbolic operations, is more robust to a disturbance like an object positioning. However, training these algorithms is challenging because of the hand-crafted programming. Also, high-dimensional data cannot be processed with them. Therefore, classic and modern techniques should be merged to create robust algorithms (Garnelo & Shanahan, 2019).

2.3. Relational Networks

Some studies focus on object relations in the environment (Džeroski & Driessens, 2001; Driessens & Džeroski, 2004). Extraction of relational information from the environment may be vital in bridging between symbolic and statistical techniques. The relation Networks (Santoro et al., 2017) study shows that the relational dependency of objects in the environment improves the performance of the neural networks. Novel Relation Networks architecture is used for an image-based task based on positionally and visually dependent geometrical objects. In the Relation Networks study, the pairwise features of the objects in an image are processed through a multi-layer perceptron. The result obtained from this processing represents the relation between the two objects. This network is tested in more than one test environment in the study. In one of the test environments, there are objects with different colors and geometric shapes in the same image. Different questions are asked to the network about the geometric and positional relations between these objects. For example, the "How many objects have the same shape as the blue object?" question is asked to the network. This question is a relational question, and in order to be answered, the geometric shape of the blue object must be associated with the geometric shapes of other objects. If the question is "What is the shape of the blue object?" this question would be non-relational and could be answered independently from other objects. After processing the environment image with CNN, the obtained features are pairwise matched and combined with the embedded vector obtained with the LSTM architecture of the question sentence. Then, the relations of the entities are determined by processing the combined vector with a Multi-Layer Perceptron (MLP) layer. These detected relations' values are summed up, and the answer to the question sentence is created with a different MLP layer. In the results, architectures using relational networks show higher performance in tasks requiring relational reasoning than architectures consisting only of MLPs. However, in non-relational questions, the relational network does not differ because understanding the relationship between the objects is no longer important. Instead of a relational network, a mapping function between each object and its shape will be sufficient for non-relational questions.

In the other test environment in the relational network study, there are moving circular objects with different colors. Some of these circular objects are connected by springs and rigid connections, and some are moving freely. The environment image is transferred to the relational network without showing these connections. After CNN processes the environment image, it obtains the features of the objects in the frame. Each feature is paired with another and transferred to the multi-layer perceptron. As a result of the MLP output, the value of how many of these circular objects are connected is obtained, and the network is trained by comparing its output with the ground truth values. As a result of the training, it is seen that thanks to the relational network method used, higher accuracy is obtained than complex MLP networks. Via the Relational Network, moving objects with invisible connections are detected. It can be said that the network's awareness of connections means that the relational module indirectly recognizes the physical law underlying the movement of the objects. By using this connection information, the existence of the spring equation can be extracted. By using powerful and complex networks, the images used in these problems can be processed quite efficiently. However, the cause-and-effect relationships underlying the environment should be well understood to solve the problem. Relational networks can solve problems that require relational reasoning with higher performance compared to other methods.

Moreover, complex image problems like Raven's Progressive Matrices (RPM), which measure humans' abstract thinking capabilities, can be solved with RN (Barrett, Hill, Santoro, Morcos, & Lillicrap, 2018). In the reference study, RPM (Raven & Court, 1938) questions are tried to be solved through neural networks. RPM questions are frequently used in visual IQ tests prepared for humans. RPM questions are about guessing what happens in the following image by understanding the relationship between consecutive images created according to a particular rule. These relationships need to be established through features such as shape positions and line colors in the image. Also, there are multiple answer options in RPM questions. Of course, there will be more than one rule in which each answer is somehow suitable for the given visual sequence, but in tests performed on humans, the correct answer is the answer in which this rule is created most simply. Therefore, the neural network model must have sufficient abstract reasoning ability to solve RPM questions that can even be challenging for human intelligence. The authors created new tests procedurally based on RPM tests, and they proposed a new dataset called Procedurally Generated Matrices (PGM). They carried out experiments on PGM using six neural network models, one of which is the proposed novel architecture. The architecture of the novel Wild Relation Network (WReN) network model, which is proposed in the article, is a very similar variant of Relation Networks (Santoro et al., 2017). WReN network outputs the prediction score for each of the eight answer options. Apart from the proposed network, baseline models used in experiments are as follows: CNN-MLP network is a simple four-layer Convolutional and two-layer fully connected combined neural network. Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) experiments are based on the standard implementation of LSTM. ResNet-50 is a popular CNN-based benchmark network architecture. The Wild-ResNet model, an adaptation of the ResNet architecture, can produce scores similar to WReN. Finally, Context-blind ResNet architecture is used. Instead of showing the questions, only answer options are shown to the neural network in this model. In this way, it was measured by the possibility of giving the correct answer only by looking at the answer options. According to the results obtained in experiments on the PGM dataset using these models, it was reported that the novel WReN network model outperformed other models and produced nearly 14% higher correct solution than even the second most successful model.

2.3.1 MHDPA Models

In the study of the relational network (Santoro et al., 2017), the authors accessed relational information using pairwise combinations of the objects in the environment. On the other hand, in this study, attention networks are used to determine the relations between the objects, which are frequently used in the Natural Language Processing (NLP) domain. The MHDPA algorithm performs the same operation as the relational network architecture presented in the study of the relational network. The MHDPA algorithm has the advantage of being more well-defined due to its use in the field of NLP compared to the method used in the relational network. However, in both algorithms, the process of crossing object features is used while detecting object relations. Thanks to attention networks, the low-level feature vectors of the objects are made dot-product (MHDPA (Vaswani et al., 2017)) with each other (as much as the number of heads). As a result of this operation, the attention of the objects in the environment is found on each other, and an N x N (N: object number) matrix is obtained. This matrix is then passed through different Multi-Layer Perceptrons to produce the policy and value functions of the agent. In this study, they train the agent with a synchronous advantage actor-critic (A2C) framework. In this way, the agent's neural network training process becomes much faster. Apart from this agent architecture, a new relational reasoning environment is also presented in the article. In the environment called Box-World, there are boxes of different colors. Each box acts as a key that unlocks the other box in its color. In this way, it releases the other box to which it is attached. In this novel environment, the agent must learn the relationship between the boxes' colors, establish a forward relationship between the lock-key boxes, and decide where to move. The architecture presented in the article has been tested in the Box-World environment. With the increase in the number of attention modules in the agent architecture, it was seen that the agent acted with quite a high solution accuracy in the Box-World environment experiments. It can be seen on the graphs that the agent was trained for a long time to solve the environment at almost 100% accuracy. This agent neural network architecture's success using the MHDPA method is significant, but it has weaknesses due to its long training period.

The possible problem of this method is that objects from the Convolutional Neural Network (CNN) output may not be correctly separated when processing a visually complex environment image. If the object features spread into each other, the relations of the objects may not be noticed as a result of the MHDPA algorithm. This clustering problem requires different expertise apart from RL. However, the clustering problem is not experienced in toy environments used for the development method of the architecture. Therefore, this potential problem can be neglected in an RL problem. Two-layer CNN is used for image processing in the relational MHDPA study. The image processing part of the relational network pipeline is essential to extract the right features of the objects. Although two CNN layers are sufficient to process the image of the environment used in the reference study, it may be necessary to define a more powerful image processing architecture for different environments. Although the agent is trained with the actor-critic method in the reference work, different RL frameworks can be used for training because the MHDPA module can work in different ways, regardless of the RL framework used.

Moreover, using MHDPA with more advanced RL methods will increase the reliability of the obtained relational information.

2.3.2 PrediNet Models

On the other hand, in the PrediNet study (Shanahan et al., 2020), the authors say that statistical deep learning methods will be more generalizable and interpretable using symbolic methods. Therefore, they try to establish a bridge between these two frameworks. They offer an agent model that can extract relational information between the objects in the image given and make it available later with post-processing. In this neural network architecture, the feature vector of different objects is applied dot-product with another object. In this way, the attention weight of the objects on each other can be calculated. The most important feature of this new architecture, presented in the article named PrediNet, allows different objects to be compared in the same relational representation space. In this way, the relationship between the objects can be obtained explicitly for a specific relational representation. PrediNet has been tested in an environment called Relations Games, with various tasks, to evaluate the architecture's performance. In the same environment, architectures created with different MLP network configurations have also been tested. As a result, the accuracy of the PrediNet algorithm in the test sets is much better than baseline architectures. In addition, weights of neural network architecture, which stores relational information, trained using Relations Games dataset are processed in Prolog logical programming language for post processes. As a result of this processing, the relational information between the objects in any image can be compared explicitly using the Prolog language.

The fact that explicit relational information obtained with PrediNet architecture can be processed with logical programming languages indicates that it can be boosted with propositional operations during its online operation. If the objects in the environment affect the solution of the environment due to any of their features, PrediNet architecture can realize the relationships of these features. These noticed feature relations contain the characteristics of the objects in the environment and can be used in the control algorithm based on their propositional relations by using correct symbolic operations. In the PrediNet study, the network is trained with the tasks made on static images, but the obtained explicit relational information is not used in the algorithm. After completing the network training, the relational information extracted is compared offline by determining certain thresholds. The task called "between" used in the study asks whether one of the three objects in the image of the decision algorithm is in the same row or column as the other two objects. In order to perform this task successfully, the agent must learn the positional relations of all objects between each other. It will be sufficient to look only at the positional relationship of the middle object between the other two objects. However, the algorithm can look at the positional relationship between the two objects at the ends. Although this situation does not harm the solution of the problem, it can be considered that the algorithm has chosen any solution instead of the easiest one. In the sample Prolog output presented in the PrediNet study, the explicit relational information values obtained after the "between" task are symbolically defined with different relation symbolic parameters. Then, by looking at these relationship values, the threshold value required for an object to be in between was defined off-line. After this value is defined, all objects that provide the given value can be listed simply. If the threshold used in this example can be defined online within the network, these symbolic operations can also be used in the control or decision-making algorithm of the agent. However, to define this value, it is necessary to use relation values of the agent whose training has been completed and whose performance is reliable. Calculating this value online may cause a different stabilization problem. Even if the thresholds can be selected appropriately, the information about which features of the objects should be compared and which direction will not be taught in the network. In addition, the difficulty of correctly clustering objects in visually complex environments can make it difficult for this architecture to extract appropriate relational information.

2.3.3 Causal Models

Some of the RL problems can be solved by making inferences from the relations of objects in the environment. However, these inferences are mostly based on correlations without relying on cause and effect relationships between the objects. For example, in the study of the relational network (Santoro et al., 2017), it is an important factor that some object positions make correlated movements in estimating invisible connections between circular objects. However, by looking at this correlation, the physical relationship between objects cannot be formulated. In order to achieve this information, it is necessary to conduct controlled experiments in the environment. For example, moving one of the connected objects should be observed to detect which objects started the movement and in what pattern these objects moved. Only in this way can the connection between objects be clearly understood. A generic understanding cannot be developed simply by looking at correlations over random observations. Through controlled experiments, cause and effect between objects can be better understood, and object movements can be predicted in more general conditions using cause and effect information. In order to carry out such controlled experiments, the trained agent should only be able to make observations in which the control parameter is changed. The agent itself can perform these controlled experiments, but when it does, it will no longer be an RL agent, and it becomes a hand-crafted algorithm. It is convenient to create these controlled test conditions through the environment at the first stage. Procedurally controlled experiments can be produced thanks to the environment called Causal World (Ahmed et al., 2021). In this environment, three robotic arms and various tasks created for these arms are simulated realistically. Different dynamics can be created in the environment thanks to the cube-shaped objects of different weights, sizes, colors, and adjustable robotic arm weights. In this environment, during training, the agent (robotic arms) can be asked to lift cubes of varying weights in each episode and move them to the designated places. In this way, at the end of the training, the agent can learn how the cubes' weights affect the solution of the task. It will then be able to learn how the dimensions affect the solution of the task by simply changing the cube dimensions. In this way, the agent learns the pure cause and effect relationships in the environment. By using this information, it will be able to produce more robust solutions when it encounters different task combinations: when the dimensions and weights of the cubes change at the same time, or when the tasks that the agent is asked to do with the cubes (push, pick and pick and place). In addition, since the environment model simulates an actual robot arm mechanism, the trained agent can achieve the same performance in real robotic arms. The agent tries to implicitly understand the dynamic equations of the robotic arms and cubes using the causal world environment. The agent, which can solve each task with high performance, will have kept the information of what weight and size objects can be lifted by applying force. This knowledge should be consistent with the physical equation of the operation to be performed.

CHAPTER 3

METHODOLOGY

3.1. Markov Decision Processes

This study is based on modern applications of the Reinforcement Learning concept. Reinforcement learning is a subfield of the machine learning field as supervised and unsupervised learning topics. There are two fundamental units in the system (Figure 1). The first one is an environment that contains states and reward functions. State elements may depend on past states. Reward function depends on states and state transitions. These are the fundamental elements of the environment. The agent unit, which can be called a controller, can act in the environment. The agent's action may be transmitted to the environment, and according to the environment state transition policy, the agent's state is changed by the environment. Simultaneously, the environment generates a reward to the agent from its reward function. This reward and state are transmitted to the agent from the environment, and it is called a feedback signal to the agent. This concept is the same as the controller (agent), the control signal (action), and the plant (environment) in the control theory. However, the all-state information of the environment may not be transmitted to the agent. These types of environments are called partially observable environments. On the other hand, if the agent can see all of the states' environments, this is called a fully observable environment.

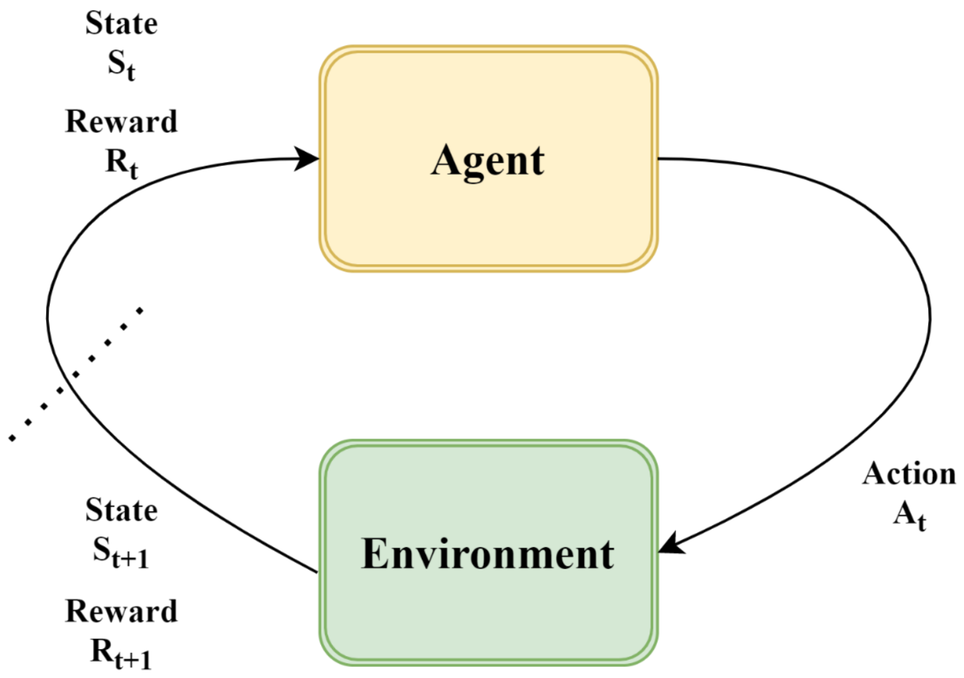


Figure 1: Interaction of the agent and environment (Adapted from Sutton & Barto, 1998).

When there is a decision problem in an environment, and the future state of the environment can be extracted from the current state, these types of problems are called the “Markov Decision Process” (MDP) (Bellman R., 1957). In order to use methods for the MDP problem, Equation 1 must be satisfied by the environment. In these problems, there are four main attributes: state, action, reward, and transition. The sequence between these attributes are following: $S_t, a_t, R_{t+1}, S_{t+1}, a_{t+1}, R_{t+2}$ ... Moreover, predicting the next states of the environment only depends on its previous state.

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_t,\ S_{t-1},\ S_{t-2}, \dots, S_1\ ) \qquad (1)$$

State transition matrix stores the information of the transition probabilities between states in the environment. It has three inputs: two states and action and gives the probability of transition between two states under an action (Equation 2). This probability distribution determines the dynamics of an environment that is operating under MDP rules. Also, there is a reward function (Equation 3) for an MDP. Reward function and the transition possibilities are the characteristics of the MDP process.

$$\mathrm{P}_{\mathrm{transition}} = \mathrm{P}(\mathrm{S}_{\mathrm{t+1}} = \mathrm{S}' \mid \mathrm{S}_{\mathrm{t}} = \mathrm{S}, \mathrm{a}_{\mathrm{t}} = \mathrm{a}) \qquad (2)$$

$$\mathrm{R} = \mathrm{f}(\ \mathrm{S}_{\mathrm{t}}, \mathrm{S}_{\mathrm{t+1}}, \mathrm{a}_{\mathrm{t}}) \qquad (3)$$

An MDP ruled environment can be generated by Equation 2 and Equation 3. The Reinforcement Learning field’s ultimate aim is to maximize the received cumulative reward from the environment, making the situation an optimization problem. In order to find the optimized solution, the agent should find an optimum state-action map while acting in the environment. The function of the state and action map is called the “policy” ($\pi(s)$) of the agent, and most of the time, it is the optimized property of the agent. Also, another important property of the agent is the “value function” ($V(s)$) of the agent. It is also a map from the current state and the weighted sum of the expected rewards from the rest of the episode. Collecting the incoming rewards by multiplying with a certain weight (discount factor) eliminates the possibility of reaching an infinite sum and prioritizing the rewards to be received in time. Also, a Q value function is the same as the value function, except it gives the expected return only for a specific action in the current state.

The Bellman Equation can solve an MDP problem, a linear equation, but it is computationally expensive for large state space problems. In order to solve large state space problems, there are some numerical techniques that are based on Bellman Equation, such as dynamic programming (Bellman R. E., 1957), Temporal Difference (Sutton & Barto, 1998) and Monte-Carlo tree search (Coulom, 2006) methods. The most challenging case is where the state transition and reward functions are not known. In order to solve this model-free problem, there are several techniques. In this thesis, the used model is based on temporal difference (TD) methods. The most popular version of the TD is called Q-learning (Equation 4) (Sutton & Barto, 1998).

$$Q(S,a) \leftarrow Q(S,a) + \alpha(R + \gamma \max_{a'} Q(S,a') - Q(S,a)) \tag{4}$$

In this method, the Q value is updated according to observations without knowing the environment (model-free). This method is value-based reinforcement learning because the agent tries to map states-action pairs with their expected returns. The other way of controlling the agent in an environment is optimizing the policy function directly. This type of optimization is called policy-based reinforcement learning. The value-based methods also generate policy function, but they produce implicitly, unlike policy-based methods.

3.2. Deep Reinforcement Learning Methods

Deep learning is a sub-branch of machine learning and has gained popularity with improvements in hardware and the ability to access large amounts of data. With the outperforming of image processing algorithms created using deep neural networks (Krizhevsky, Sutskever, & Hinton, 2012) in competitions, deep learning has become one of the hot topics in academic and industrial studies. Deep neural networks can be briefly summarized as a very nonlinear optimization process to create the function that will give the desired output from the input vector given to the network. At first, the data fed as input is multiplied by random trainable weight matrices, and scalar trainable bias values are added to this multiplication. After adding the bias values, the output is put into various nonlinear activation functions such as ReLU, sigmoid, and softmax to create nonlinear approximations. The output of these functions is processed in the same way with another set of trainable weights and biases. The final output is processed with the weight and bias set to bring the processed input-output dimensions to the desired output size. The resulting output is then evaluated via a loss function. If the ground truth output from the relevant data is known for that data, the loss function is obtained by comparing the network output and the ground truth values, and this method is called supervised learning. By looking at the change of this loss function, the network weight and bias parameters are changed at a specific rate, and this process is called the training process. If the ground truth values are not known beforehand, the data given to the network as input is reconstructed over the network, and the network tries to reproduce the input. In this way, the hidden patterns in the data fed as input are displayed within the network. This representation can then be used to process data that the network has never seen. The process of training is called unsupervised learning.

The third and last subheading of the machine learning field is reinforcement learning. In this field, the decision-making algorithm is optimized in an environment that acts according to specific rules. The agent acting in the environment tries to make the optimum actions according to its observations. In this respect, it is somewhat similar to supervised learning. This method allows classical tabular methods such as Q-learning to be used as functions via deep neural networks. Studies in which reinforcement learning methods are used with the deep neural network are collected under the sub-title of deep reinforcement learning. Since networks need many samples and providing stability is more complex than tabular methods, some problems that are not seen in tabular methods can be seen when using neural networks in these problems.

3.2.1 Actor-Critic Architechtures

The policy-based algorithm’s main disadvantage comes from the instability of the network due to the high variance in the gradients. However, the value-based methods are not good at environments with high state space. The actor-critic method (Konda & Tsitsiklis, 2000) (Figure 2) combines both policy-based and value-based methods. It takes advantage of both algorithms. The value network tries to estimate the expected returns of the states. Simultaneously, the policy network generates state-action mapping according to feedback that comes from the value network. This way, the policy network drives (actor) the agent, while the value network sends feedback (critic) to the policy network’s loss function.

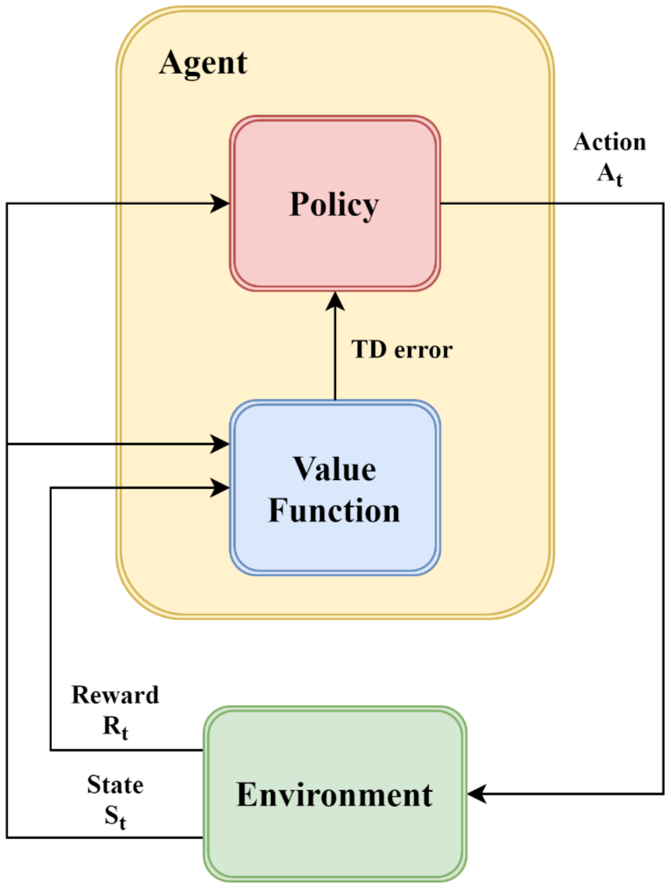


Figure 2: Actor-Critic method architecture (Adapted from Sutton & Barto, 1998).

The actor-critic method can be modified with parallel training frameworks like A3C (Mnih et al., 2016) and A2C methods. These networks use more than one agent (worker networks) in order to train actor-critic models. For example, the A3C network uses N different policy and value networks, and their training process continues collectively. These worker networks send their gradient matrices to a global actor-critic network asynchronously. Then, these worker networks update their weights from this global network. In this way, the global network is trained with a high variance network. In this thesis, the A3C architecture is used to train the relational network and PrediNet modules. Unlike regular actor-critic models, A3C is using a different loss function parameter for the policy network. The policy network's gradients are based on the advantage function rather than the value function. The advantage function is calculated as:

$$A(s) = Q(S, a) - V(S) \tag{5}$$

Using the advantage function, the policy network realizes the relative advantage of making an action over the other possible actions in a state. Moreover, there is also an entropy term in the policy loss function to direct the agent to explore the environment to avoid local minimum solutions. Then, the loss functions of the value and policy networks are given in Equation 6 and Equation 7 (Mnih et al., 2016).

$$L(\theta_1\,)_{value} = \sum (V(S) - R)^2 \tag{6}$$

$$L(\theta_2)_{policy} = -log(\pi(s)) \,*\, A(s) \,-\, \beta * H(\pi) \tag{7}$$

Then, gradient calculations are made over the obtained loss functions, and the network weights are updated as follows:

$$d_{\theta_1} \leftarrow d_{\theta_1} + \frac{\partial L(\theta_1'\,)_{value}}{\partial \theta_1'} \tag{8}$$

$$d_{\theta_2} \leftarrow d_{\theta_2} + \nabla_{\theta_2'} L(\theta_2'\,)_{policy} \tag{9}$$

, where θ_1 and θ_2 are global network's trainable parameters, and $\theta_1^{'}$ and $\theta_2^{'}$ are local worker's trainable parameters. With the asynchronous update of the trainable parameters of the policy and value networks, the agent's network quickly becomes stable thanks to the high variance data of the A3C model.

3.2.2 Attention Networks

Attention networks are very frequently used tools in the field of the Natural Language Process (NLP). These tools can be used in any sequence-to-sequence algorithms as well as in NLP problems. In the sequence-to-sequence transforming process, attention networks are a good way to represent the sequence's attention independently from their distances. The sequence (E) is multiplied with the query weight matrix (W_q) which is the network's trainable parameter, and then it is multiplied by the key weight matrix (W_k) which is also a trainable parameter to get the *Q* and *K* matrices (Equation 10 and Equation 11).

$$Q = EW_q, \tag{10}$$

$$K = EW_k, \tag{11}$$

Then the dot-products of the resulting Q matrix and the transpose of the K matrix are calculated. As a result of this process, due to the nature of the dot-product operation, the values of the object features in both matrices that are close to each other are increased, and the values of the distant ones are further reduced. In this way, the level of attention of their objects on each other is obtained. Then the softmax operation is applied to the matrix obtained after dot-product, which is divided by a scale factor ($\sqrt{d_k}$). The obtained attention distributions are normalized between 0 and 1 by the softmax operation. The matrix obtained as a result of this process is called the attention weight matrix (Equation 12).

$$\mathrm{M_{att}(E)} = \ \mathrm{softmax}\left(\frac{\mathrm{Q}K^{\mathbf{T}}}{\sqrt{\mathrm{d_k}}}\right) \tag{12}$$

Finally, the attention matrix is multiplied by the *V* matrix (Equation 13), and the resulting matrix (Equation 14) is sent from the relational module to be used in the continuation of the network.

$$V = EW_v \tag{13}$$

$$\mathrm{A_H(E)} = \mathrm{M_{att}}V \tag{14}$$

This attention matrix can be examined in post-processing, and the attention levels of the words on each other can be seen. The name self-attention is used in this process because *Q*, *K*, and *V* matrices are created for the sequence itself. The (Vaswani et al., 2017) study showed that generating more than one of these matrices for the same sequence and then combining them (Figure 3 and Figure 4) and creating multiple attention representations improved the performance of the NLP algorithms. Every set of the query, key, and value matrices are called one head, and the method which concatenates several matrices is called Multi-Head Dot-Product Attention (MHDPA).

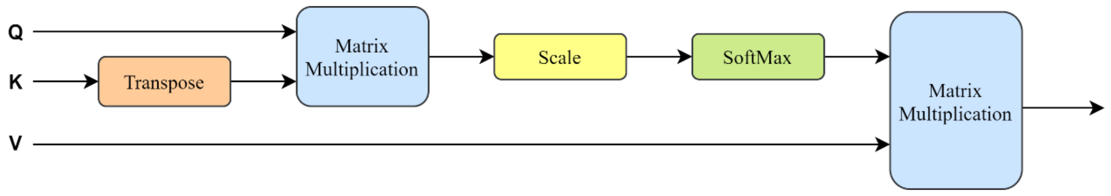


Figure 3: Scaled Dot-Product Attention Architecture (Adapted from Vaswani et al., 2017).

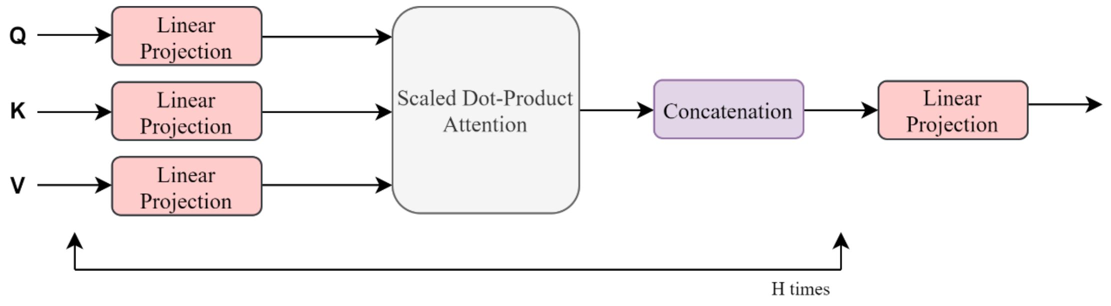


Figure 4: Multi-Head Attention Architecture (Adapted from Vaswani et al., 2017).

More than one representation space is created thanks to the multi-head transformer architecture, and more than one attention matrix is obtained. The increase in representation diversity enables the sampled data to be processed more detailed and unbiased. For these reasons, multi-head attention networks, which are becoming more common in the field of NLP, can also be used in reinforcement learning to strengthen the relational reasoning capacity of the agent algorithm. If the attention weight between the objects in the agent's environment can be derived correctly, the agent can create its policy in the environment more accurately. Apart from optimizing the agent's learning process, the agent's decisions based on when the training is completed can be interpreted through the attention matrix. Therefore, attention networks can offer a solution to the interpretability problem. However, to create a solution to the generalizability problem, it will be necessary to know the more precise (cause-effect) relations instead of the object relations in the environment (a kind of associational information). The use of this information, on the other hand, will enable making both more precise and more generalizable decisions by using symbolic mathematical operations instead of statistical methods. Therefore, attention networks will not provide a definitive solution to the generalizability problem.

3.2.3 Relational Networks

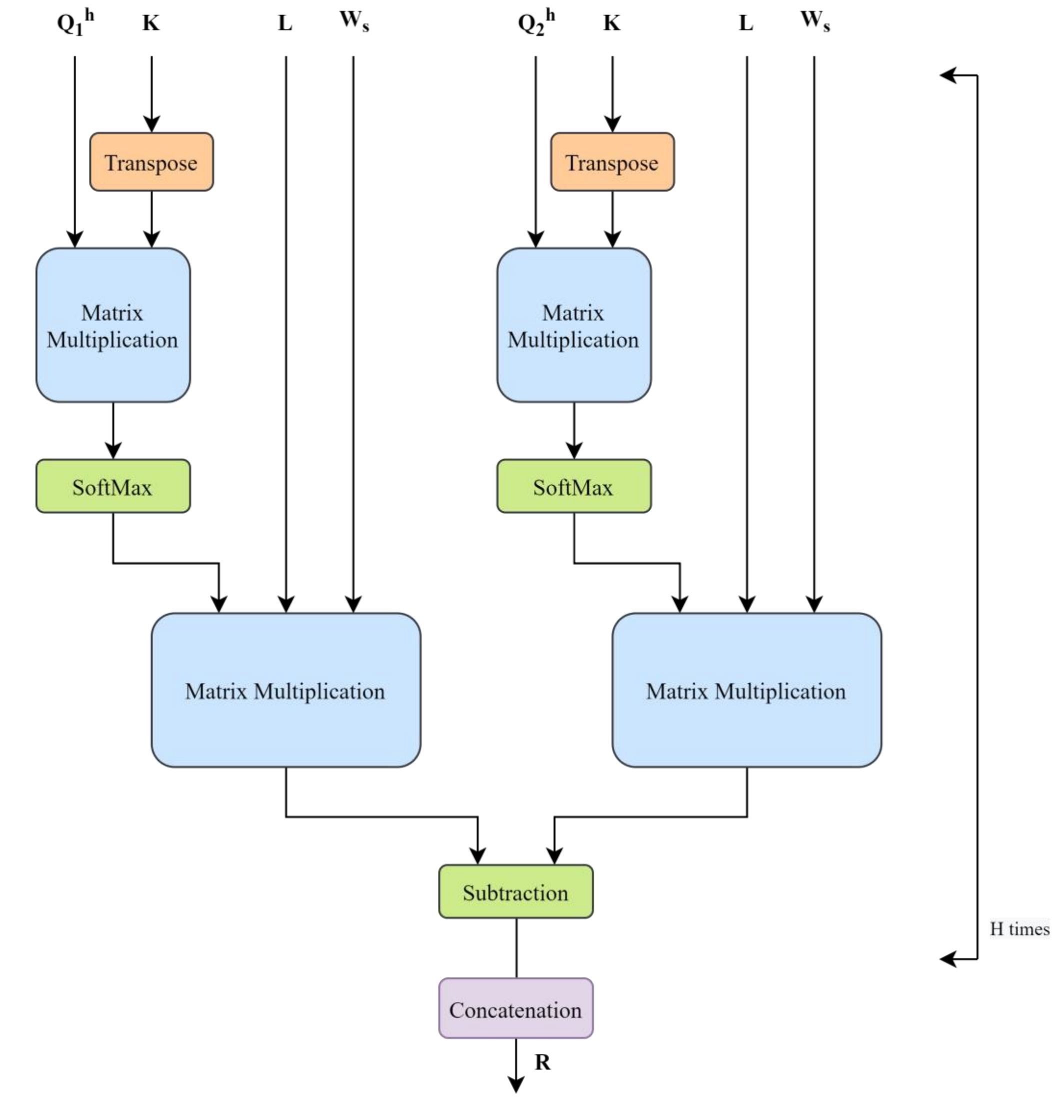
Relational networks are deep reinforcement learning methods with relational reasoning capability. Thanks to these methods, objects or states with relatively strong relations can be detected, enhancing the agent's decision-making and controlling performance. In order to understand these relational connections, it is necessary to determine the interactions of objects with each other. The most direct way to detect these relations is to close the optimization loop by inserting the feature vector of complete combinations of the objects into a neural network. Attention network tools mentioned in the previous section can be used to create all combinations in this way. This tool can infer the relations of the objects in an MDP environment, just as the levels of relations of words in a sentence are inferred. In this way, it will be easier to interpret the decisions made by the RL agent, and it will be easier to notice the cause-effect-based phenomena in the environment. In this way, more interpretable and generalizable results can be obtained.

The study (Zambaldi et al., 2019) that relational networks are used in a DRL problem, the environment frames (observations) are processed with a two-layer CNN, and resultant feature maps are sent to the operation called attention module. Each image frame's feature maps (E) are put into the MHDPA process in this operation. The arrays that come out of the MHDPA process are then passed through the fully connected layer, and the output of the combinations is updated. The representations formed later are passed through four sequential fully connected layers, and $\mathrm{n}+1$ outputs are provided, of which n policy outputs and one expected value. In this way, a reinforcement learning agent algorithm is driven using multi-head attention architecture. The attention matrix in the MHDPA core algorithm of this algorithm provides essential outputs in terms of the interpretability of the agent algorithm.

Different attention values for more than one head from the MHDPA operation carry the relational information of the objects in the environment together with the agent's observations. This information is also used iteratively by the agent in policy generation. The reason why the output of the MHDPA architecture cannot be used directly by the agent is that this information carries the attention information of the objects directly, instead of being a representation that the agent can generate policy. For these reasons, attention information is moved to a different representation by passing through more than one fully connected layer. This final representation is suitable for generating policy logits and value estimation values for the agent's algorithm. In addition, softmax operation is applied to the first four outputs of fully connected layers to create policy logits. In this way, the MHDPA method is put into the form that a reinforcement learning agent can use. Instead of this method, as the same authors applied in their previous work (Santoro et al., 2017), the feature combinations of each object pair can be obtained with a multi-layer perceptron to extract relational reasoning information. Deriving relational information from this combination method would also have quadratic complexity, like MHDPA. Therefore, the two methods do not have many advantages over each other.

3.2.4 PrediNet Architecture

The PrediNet architecture provides exploring various relationships by using multi-head attention, as in relational network architecture. Differently, the object pairs in the environment are distributed to each different head. Since the same attention matrices are used in all heads, except for the W_q, the same relational representation is used in each one. The fact that the W_q matrices are different in each head, allowing the *j* relations of different object pairs to be examined. By subtracting the results from this multi-head network (Figure 5), the relationships in *j* different relational representation spaces are revealed. It is anticipated that the network derives relational information independent from the position information of the objects. Therefore, in this study, unlike the reference procedure, the position information of the entities is not included in the difference vector. This relational information can be used for optimizing the actor-critic policy and value estimations. The relational information resulting from this network is formed in the propositional representation, and it is available to be used beyond the network pipeline. Because PrediNet can derive propositional information from high-dimensional data, it is suitable for symbolic operations and, therefore, induction operations for the data obtained statistically.



L: Input Q_N^h*: Query array of* N^{th} *entity in* H^{th} *head*
K*: Shared key matrix for all heads* W_s*: Shared linear mapping matrix for all heads*

Figure 5: PrediNet Network Architecture (Adapted from Shanahan et al., 2020).

The computational complexity in the PrediNet architecture is less than Relational networks because the relationship between only two entities is examined in each head instead of all the entities. Moreover, the W_k and W_s matrices are shared in all heads. For this reason, its performance as a decision-making algorithm is expected to be lower since it establishes fewer relationships than a relational network. For this reason, it is the most important advantage to extract propositional information from the environment. Since the positional information of the entities in the relational network is not used directly, also it is not used in PrediNet. This information is one of the differences between the original study and its use in this thesis. The propositional information created by PrediNet is expressed with the *R* vector. *R* vector represents the scalar magnitudes of the relations of binary entity combinations at *H* numbers heads for *j* different relational representations. By using this scalar information, the magnitudes of the relative relational representation between two entities can be compared. For example, suppose it is known that the relational representation is the object dimensions. In that case, the dimensions of the two objects in the environment can be easily compared over these values, and additional operations can be performed according to the size of the objects in the environment. If there is a positional relationship in one of the representations, the distances of the objects can be read through these values, which can improve the decision-making algorithm. In the PrediNet architecture, relational information between the two entities is obtained by subtraction of the individual representations of the entities. The reason subtraction is preferred in this process is that the relative representational difference between the entities represents their relationship. It is essential to extract a relative representation based on the signs of the representations. For example, if the Euclidean distances are used instead of the subtraction operation, it would be impossible to understand the aspects of the relations. The absolute value of the difference between the entities will be obtained by using Euclidean distances. For example, the magnitude of the size difference between the objects will be known, but it will not be possible to obtain information about which one is larger or smaller. This difference information can also be obtained by using another signal-sensitive operation instead of subtraction. For example, another network that can extract the relative relations of representations can be used as an alternative for this operation.

One of the problems of relational reasoning neural networks for raw visual data is to cluster the objects on the image correctly. The ability of the agent to distinguish objects in the environment from each other and the background will dramatically change the ability to do relational reasoning. If different objects are in the same entities for the agent, it creates misleading information that they are entirely dependent on each other. Moreover, the background image of the environment is included in the entities will create misleading information during the comparison of two objects. For example, in an environment where object dimensions are important for the solution, the dimensions will be evaluated incorrectly because the background will spread to the objects. For these reasons, it is important to do the object clustering correctly over the environment image. However, this problem is an image processing problem rather than an RL problem. For this reason, in theoretical RL algorithm studies, it would be appropriate to send the objects to the agent in a completely clustered manner.

The *R* vector containing the relation information obtained from the PrediNet architecture can be processed in the Prolog logic programming language. The relational information for each head obtained from the *R* vector can be compared within the Prolog. In this way, queries can be made in relational representation spaces that are dimensionally large/small objects, positionally close/far objects, or physically meaningless representations for a certain object. These queries can be manipulated with symbolic operations within the agent architecture and used in the agent's decision-making or control algorithm. In this way, generic algorithms can be developed independently from some of the environmental parameters. In addition, if the agent is controlled with symbolic manipulations in an environment whose rules are pre-known, by using this relational information before the training process, the agent can move these relational vectors (R) to the desired relational representation. For example, if there is a connection related to object dimensions in the environment, the agent's actions can be directed by making symbolic decisions as if this dimension relation information is obtained in the agent algorithm. The relationship vectors of the agent in the PrediNet module will also be optimized to give priority to the object size information at the end of the training process. However, this method means that the task in the environment is known in advance, and thus the difficulty of the problem is reduced in advance. Because before the agent starts training, the algorithm will be trained knowing that the dimensions are an important parameter. For this reason, the symbolic operation added to the agent algorithm must be created without relying on prior knowledge of the environment, which prevents the symbolic operation from being complex. These symbolic operations must also be expressed with trainable parameters. Processing symbolic manipulation output of the previously prepared physical relations between two objects with a neural network can help overcome this problem. Although the PrediNet architecture is more suitable for generalization than relational networks, this cannot be said in terms of interpretability. Because, thanks to the attention matrix in relational networks, it is easier to visualize and understand which decision of the agent is due to which object's relationship. However, in PrediNet architecture, this should be done for combinations of two objects over scalar relation values, reducing the interpretability of the agent's decisions. On the other hand, the relationship of the exact combinations of all objects in the environment can be seen in the relational network.

3.2.5 Mixed Architecture

A3C, MHDPA, and PrediNet modules are used together in the network used in the experiments (Figure 6). Twelve different agents (workers) act in twelve different environments parallel and asynchronously using their networks (with different random weights). A3C architecture is used in the outermost shell of the algorithm for training the agent. The maximum number of parallel agents allowed on the hardware where the tests are performed is 12. The reason for choosing this method is that it can provide faster training thanks to its high variance sampling capability. In the inner shell of the A3C architecture, there are two layers of CNN models to process the input image. CNN is a deep neural network method that is frequently preferred in image processing due to its capability of detecting low-level features of the image quite well. Small-scale CNNs were used due to the relatively small image sizes of the environments the experiments are conducted. The feature maps obtained from the CNN layers for each worker are transferred to the Relational Module selected at the beginning of the training. According to this selection made at the beginning, a switch is made between MHDPA and PrediNet to determine which network output will be used. In order to make appropriate switching, the input sections of the versions of the MHDPA and PrediNet modules used in the reference articles were updated. As the output of this switching, features of the same size are derived from both modules. These obtained vectors were then sent to four consecutive fully connected layers as in the reference article (Zambaldi et al., 2019). Five values are calculated for each observation from the fully connected layer at the end of the neural network pipeline. Four of them are softmaxed to create policy logits, and the resulting logits are used as policy function outputs for A3C. The fifth and last value is used as the value estimation value in A3C architecture without any extra processing.

A loss value for the value function ($V(s)$) is calculated by comparing the value estimations with the reward obtained from the environment at that time step. Then, the loss calculation for the policy function is calculated with the advantage function, which is obtained from the value estimation (Equation 5). With these loss calculations, the gradient required for the policy and value functions networks is calculated. In this way, the value estimation function, which is the critic unit of the A3C, is trained according to the rewards coming from the environment. If the value function predicts the value of the reward obtained from the environment well, it will ensure that the policy network is well trained. Using these two function estimators together, the disadvantage of value-based methods in large state space environments and policy-based methods being unstable due to high variance gradients are minimized. The policy network, which is the actor unit of the A3C, tries to optimize the policy logits (statistical distribution values for movement in four directions) according to the feedback it receives from the value network. One of the four actions is selected for each environment while calculating the loss values according to the policy logits. The individual worker in the environment performs this selected action. According to this action, the environment updates the agent's state and moves to the next time step. In this step, the worker observes the new state of the environment, and the same calculations are repeated. Each worker does not use the gradient values calculated in these iterations to update their networks. Each worker sends the gradients asynchronously to the global agent network, and the weight values in the global agent's network are updated according to the gradient values from all workers. Workers who send the gradient values for this update then copy the weights of the global agent network to itself. In this way, all workers benefit from highly variance-trained network parameters that other workers improve. Workers are trained through different relational modules, depending on the relational module selection selected before the training starts. It can be said that the tests performed for both modules are carried out under equal conditions since both modules had the same structure upstream and downstream of the pipeline. Also, their architectures are similar to each other.

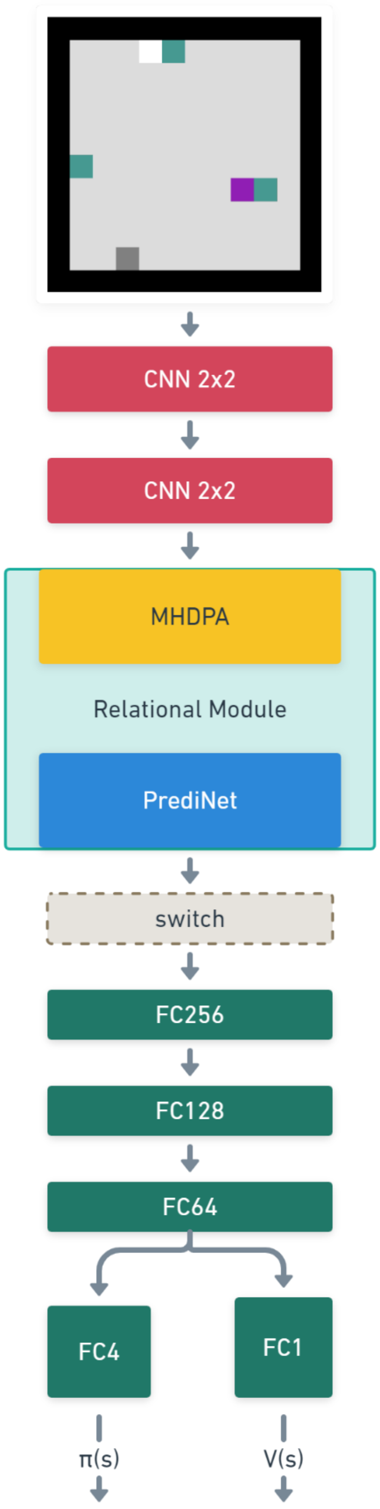


Figure 6: Mixed Architecture (Adapted from Küçüksubaşı & Sürer, 2021).

Most of the mixed architecture parameter sets are selected from the reference article of the MHDPA module. In order to compare the PrediNet and MHDPA methods, the same network parameters were used for both modules. Although the number of heads used during attention processes is the same, the unique number of relation values of the PrediNet algorithm was determined as a result of preliminary experiments (5.1). The only downside to using the same parameter set for both modules is that it misses its own best performance points from the two modules. However, finding the parameter sets that are the best is not preferred due to controlled experimentation (thus comparing their performance) and double the parameter tuning time. Therefore, the same hyperparameter set is used for both modules. As a result of preliminary experiments, it is understood that this approach is not wrong at all. Although the same parameter sets are used in the modules, the trainable parameter number of the two modules differs due to the architecture of the PrediNet. For this reason, differences in training periods are observed.

CHAPTER 4

ENVIRONMENTS

4.1. Relational-Grid-World Environment

Relational-Grid-World (RGW) environment is developed for the analysis of the relation-based reinforcement learning algorithms. There are eight objects (and empty grids) in the environment related to each other. RGW can be generated procedurally so that it can produce a high variance sample. Therefore, it can be used to train more robust neural networks. RGW configurations, which have 10 x 10 grid sizes as in Figure 7, are used in thesis experiments to understand objects' effects easily.

There are two configurations of RGW which is used. The only difference between them is the mountain and pit objects because they are penalty objects, increasing the complexity of the environment. The objects and their reward values are listed in Table 1.

Table 1: Reward mechanism of the Relational-Grid-World objects.

Object	Terminal	Enemy	Using Sword	Mountain	Pit
Reward	10	-1	1	-0.1	-1

The main properties of each object are following:

- *Agent*: The agent object (Figure 8-a) takes actions to the up, down, left, and right from its grid. It can go to only one grid for every step and cannot go on the wall objects. It is the only smart unit of the environment because it is driven by the output of the reinforcement learning algorithm. The main objective of the agent is getting maximum rewards from the environment. Also, it should reach the terminal state if there is a penalty for each step. This type of penalty is commonly used to find the optimum path between the spawn point of the agent and the terminal state, and there should be a penalty for spending time on empty grids. If there is no penalty for each step, no reward for terminating, and no limit on the number of steps, the agent may never finish the episode. There is no reason for it to go to a terminal state in this situation. Unlike the other objects, an agent is necessary for the RGW environment to train reinforcement learning algorithms.

- *Terminal*: The terminal object (Figure 8-b) is one of the objects which terminates the episode (pit object can also do). When it reaches this state, the agent is rewarded with +10 points to accelerate its search for an optimal path. Terminal state location is one of the most important objects for setting the configuration's difficulty. Therefore, while generating a configuration, locating the terminal state will save time for adjusting the difficulty of the environment.

- *Wall*: The wall object (Figure 8-c) is the most passive in the environment. They are used for users to create the basic structure of the environment configuration. Using these objects is the easiest way of adjusting the environment's difficulty. Most of the time, the wall objects do not give the agent any positive or negative reward. They change the complexity of the environment indirectly. Also, they are not the necessary objects for the creation of a working environment.

- *Enemy*: The enemy object (Figure 8-d) is one of the essential objects of the RGW environment in terms of the relational dependency with another object. The enemy object gives a penalty to the agent if the agent has not got the sword object. However, if the agent has it, the enemy gives positive or zero rewards. The most common reason for using this object is enforcing the agent to establish a relationship between the enemy and the sword objects. The most proper use of this object is to put it on the optimum path in the environment configuration and force the agent to pass over it.

  In the environment configurations used in this thesis, if the agent comes on the enemy object's grid without having a sword, the agent is given a -1 penalty point. On the contrary, if it reaches the enemy grid with the sword, +1 reward point was given to the agent. Another way of using the enemy object is to position it to the environment in a dummy manner. In order to do this, the enemy object is placed on a non-optimal route to attract the agent's attention. During the training process of the agent, while the enemy is on the optimal path in some episodes, it is kept as a dummy object in some episodes, and it is expected that the agent should understand whether or not it should go to the enemy object. Because in order to go to the enemy object, it will need to consider how it will go to the sword object, and in order to do this, the agent must be able to calculate more than one forward move. Moreover, increasing the number of enemy objects causes an increase in the complexity of the environment configuration.

- *Sword*: The sword object (Figure 8-e) can be seen as a pair of enemy objects. It will be useless without any enemy object because it has no property except for beating it. It can be used for the enemy objects multiple times. The sword and the enemy objects are the most important instruments to generate a relational dependency in RGW, which gives the name of the environment. Therefore, most of the time, they are used in the configurations.

- *Teleportation*: The teleportation object (Figure 8-f) creates instant changes in the position of the agent. Due to their functionality, they must exist pairwise (entry and exit) in the environment. When the agent reaches a teleport object, it will appear over the other teleport object at the next time step. The main reason to use these objects is to distort the environment's positional cause and reason structure. Therefore, the agent should be robust for dramatic changes in the position. Moreover, during configurations, teleport objects can provide great flexibility in shaping the optimum path. The agent can be directed to teleport objects by creating dead-ends at various points in the environment.

- *Mountain*: The mountain objects (Figure 8-g) are the most useful objects while building the optimal path in a configuration. They give negative rewards to the agent, unlike the wall objects. In the experiments which are presented in this thesis, they give a -0.1 reward to the agent. In order to understand their effect, they are not used in Configuration-1 (Figure 7) and used in Configuration-2 (Figure 9) of the RGW environment.

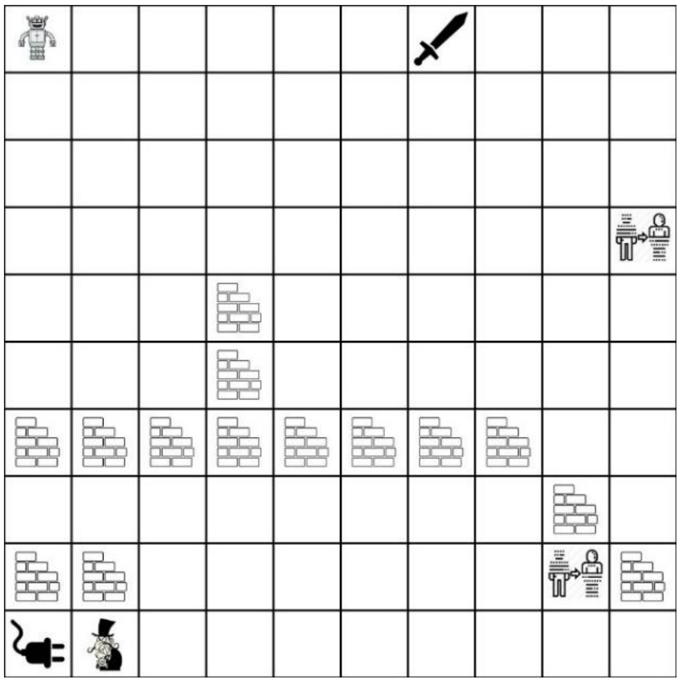
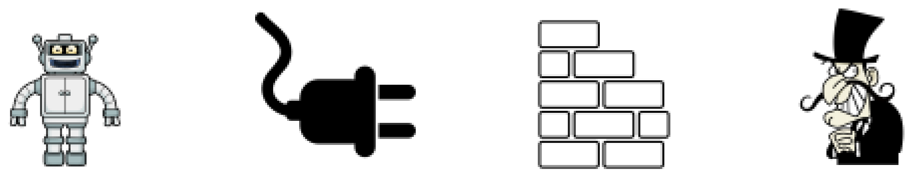
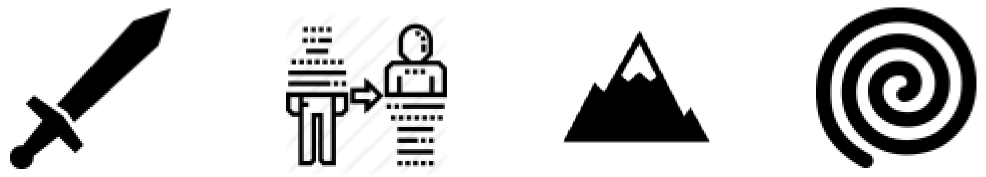


Figure 7: Relational-Grid-World Configuration-1.

- *Pit*: The pit object (Figure 8-h) is another object that can terminate the episode other than the terminal object. Therefore, the agent may tend to the pit objects when the environment is not configured well. If the agent does not make efficient actions for exploration, it may go to the pit object to terminate the environment. It may terminate the episode to escape from the other penalty points. Also, it is the second difference between the configurations which are used in the experiments. The pit object also gives a -1 reward point to the agent like the enemy object.



(a) Agent [2] (b) Terminal [3] (c) Wall [4] (d) Enemy [5]



(e) Sword[6] (f) Teleportation [7] (g) Mountain [8] (h) Pit[9]

Figure 8: Relational-Grid-World Objects.

During the generation of an RGW configuration, firstly, the grid size of the environment should be decided. The default setting for the grid size is 10 x 10. Increasing the grid size causes an increase in state space, requiring the agent algorithm to be trained in higher dimensions. Therefore, a larger grid size tests the stability and performance of the neural network instead of testing the agent's core algorithm's performance. For this reason, rather than enlarging the grid size, increasing the complexity with the variety and repetition of the objects in the environment is more useful in measuring the relational reasoning capacity of the algorithms. In order to do this, firstly, the location at which the agent spawns in the environment should be determined because the other objects should be

[2]FAVPNG (2020). Robot - Robot Clip Art Vector Graphics Image [online]. Website https://favpng.com/png_view/robot-robot-clip-art-vector-graphics-image-png/Jc5wb5r4 [accessed 17 July 2021].

[3]FLATICON (2020). Plug - Free Tools and utensils icons [online]. Website https://www.flaticon.com/free-icon/plug_31863 [accessed 17 July 2021].

[4]FAVPNG (2020). Wall - Clip Art [online]. Website https://favpng.com/png_view/brick-clipart-rectangle-square-brick-clip-art-png/XhH1JjMX [accessed 17 July 2021].

[5]PNGEGG (2020). Villain - Fictional Character [online]. Website https://www.pngegg.com/en/png-nfhkb [accessed 17 July 2021].

[6]PNGWING (2020). Sword - Clip Art [online]. Website https://www.pngwing.com/en/free-png-nxuvd [accessed 17 July 2021].

[7]PNGIO (2020). Teleportation - Clip Art [online]. Website https://pngio.com/images/png-a1788695.html [accessed 17 July 2021].

[8]HICLIPART (2020). Mountain - Clip Art [online]. Website https://www.hiclipart.com/free-transparent-background-png-clipart-stcbr [accessed 17 July 2021].

[9]FAVPNG (2020). Circle - Spiral Circle Clip Art [online]. Website https://favpng.com/png_view/circle-spiral-circle-clip-art-png/wLy3E82g [accessed 17 July 2021].

located according to this spawn point. After determining the spawn location of the agent, the condition for termination of the episodes should be determined. The terminal condition can be specified by the maximum episode length parameter, reaching the terminal object, or both. After the terminal condition is determined, it should be decided where the enemy (or enemies) and sword objects will be located in the environment. If these objects are not used, the RGW environment converges to a kind of maze problem. For this reason, these two objects are used most of the time. After the locations of these two objects are determined, the paths in which the agent will receive rewards are arranged using wall and pit objects. This path may be the shortest path to reaching the terminal or enemy objects by taking the sword. After these paths are specified, if it is anticipated to add an extra effect for increasing the agent's position robustness, the agent's optimum path can be manipulated with teleport objects. Also, the mountain objects can be located in the environment to make it difficult for the agent to detect the optimum path. Since the negative reward brought by the mountain object is lower than the other penalty objects, the optimum reward that the agent can earn can be adjusted by using this object. In this way, the agent's recognition of the optimum path in the environment is made difficult to distinguish between other paths.

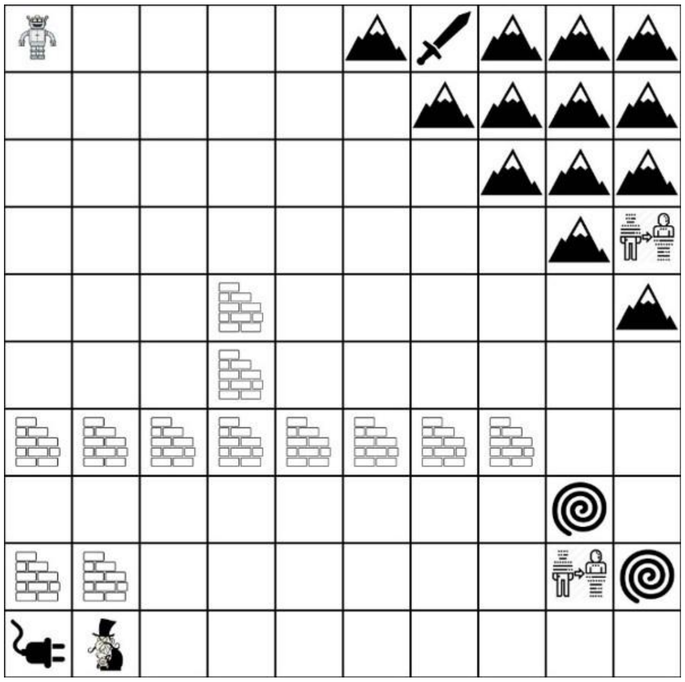


Figure 9: Relational-Grid-World Configuration-2.

It is expected that the relational information obtained from the generated RGW environment configuration can correctly point out the relations of the RGW objects. As expected, the strongest relationships should be between the agent and the terminal object because the agent wants to reach the terminal object no matter what is on the optimum path. This is because creating a configuration that will constantly give positive rewards in the RGW environment is not easy. For this to happen, there must be more than ten enemies in the environment. As long as this situation is not created and there is a terminal object, the agent's interest will be through the terminal object. Apart from this, if there are sword and enemy objects in the environment, and these two objects are located on the optimum path of the configuration, the agent is expected to realize the relationship between them. This relationship can also be noticed by the agent through other objects indirectly. For example, if there is a different object between these two objects on the optimum path, the agent can associate the object between the enemy and the sword object. The main reason for this situation is that the agent algorithms do not adequately understand the cause-and-effect relationships in the environment. Agents often make sense of relationships in the environment through correlations. Much stronger symbolic operations must be used in the agent algorithm to be able to detect the connection between related objects directly.

The main reason why the agent cannot realize this direct relationship is that after the agent receives the sword object, it visits another object, and then it stops on the enemy object. The agent supposes the object that is not causally related to the enemy object to gain positive points over the enemy. In this type of configuration, when a human player drives the agent, the player notices the relationship between the sword and the enemy for the first time. Human players realize this direct relationship because they use the knowledge of enemy-sword relation before playing the game. If two meaningless images are used instead of swords and enemy visuals, the human player may fall into the same mistake that the RL agent makes. However, as the human player drives the agent in the environment, the player will realize a direct relationship between these two objects after doing controlled experiments. Since the RL agent is not programmed to conduct controlled experiments, it will not fully determine this relationship. If the agent algorithm is programmed to perform such an experiment, the basic algorithm used by the agent will not be a statistical algorithm. It will be a logic-based algorithm. For these reasons, the fact that an agent trained in the RGW environment understands the direct relationship between enemy and sword objects will signify the capacity to notice cause-effect relationships in the environment. As a result of the training, the correct evaluation of the test in the environment needs to check the object relations of the agent. If the relationship between these two objects is noticed indirectly, the agent can notice the relationships but cannot understand the causality. Recognizing the cause and effect relationships in the environment will increase the generalizability ability of the agent algorithm. Apart from this, the interpretability of the agent algorithm also increases as the agent makes its decisions using this causal information.

When an agent, which was trained in an environment configuration where the link between these two objects affects the optimum path, is trained in a configuration where the two objects do not affect the optimum path, information about the effects of the agent's relational information on the policy can be obtained. For example, suppose the agent trained in an environment where these two objects are on the optimum path is put in an environment configuration where the enemy object is not on the optimum path. In that case, the agent will need to find the optimal path directly by not using this information, although it knows the sword-enemy relationship. This gives information about how much the agent can separate these two objects from their relationships with other objects.

4.2. Box-World Environment

The Box-World is a 12 x 12-pixel environment which is firstly introduced in the relational deep reinforcement learning study (Zambaldi et al., 2019). This environment is designed to test relational network algorithms. The most important feature of the environment is that it is not too complex visually, allowing one to focus on the essence of relational problems more easily. The environment consists of grids, as in RGW. The most important object in the environment is the grid that symbolizes the agent, and in this environment, as in RGW, it can take actions in the up, down, right, and left directions, one grid at a time. Some grids contain objects called keys, locks, and gems. There is a key object of the same color for each lock object, but not necessarily a lock for each key object. Apart from that, the lock objects stand adjacent to the object of another color (it can be a key or a gem). In this way, if an agent which has a key visits a lock with the same color, the object adjacent to the lock is released, and the agent can take this object. If this free object is a gem, the episode is terminated. If it is another lock, the agent takes this lock and continues the episode. Some keys in the section can open more than one lock. By using this property, scenarios can be created where the agent never reaches the gem block. Therefore, before using the keys, the agent must predict which lock to open. Otherwise, it will hit a dead end and may end the episode with a low reward: How many locks should be unlocked before reaching the gem object, and how many distractor paths can be set procedurally via the environment configuration parameters. The rewards given by the objects in the environment and the color codes of the objects can be seen in Table 2.

Three configurations of the Box-World environments (Figure 10, Figure 11, and Figure 12) are used in the Relational Network and PrediNet architecture experiments. The configurations are determined in order to create three levels of difficulty. These difficulties are settled with the solution length and number of the distractor path. Also, each configuration can be generated procedurally within itself. The colors and the locations of the key-lock blocks are randomly determined in each episode. This makes the network's training process harder in terms of robustness and generalizability. Also, the episode is terminated after 300 steps of the agent in every episode. In this way, long episodes are avoided, and training time is decreased. When the agents visit a key block, the key block is moved to the left corner of the environment rather than erased to simplify the network's task. Therefore, there is no need to memorize every key taken by the agent. When the agent visits the lock block in the baseline Box-World environment, it should also visit the adjacent box to obtain a key or gem. However, in this thesis, experiments are done differently. When the agent visits a lock, an adjacent key or gem box has been received even if not visited by the agent. In this way, the training process was shortened, and limited hardware is used more appropriately. Moreover, this situation will not affect obtaining relational information in the environment, and it will only accelerate the agent's learning process.

The first configuration's solution length is one step, with no distractor path (Figure 10). Therefore, the agent (grey) should go first to the key box (green) and then visit the other green box to obtain a gem. After reaching the gem, the episode restarts, then the key-lock pair's color and the objects' initial location (including the agent) are redetermined. This is the easiest configuration that can be generalized from the Box-World after the configuration, containing only the agent and gem boxes. Therefore, the performance of the agent in this configuration will be a reference for the other ones. The first required skill from the agent algorithm is learning to locate itself and attending to the gem's adjacent lock box's color. Then, driving the agent box to the key box of the lock of the gem. Also, the agent should learn that the lock gem cannot be obtained, so the agent should first look at the gem's adjacent box.

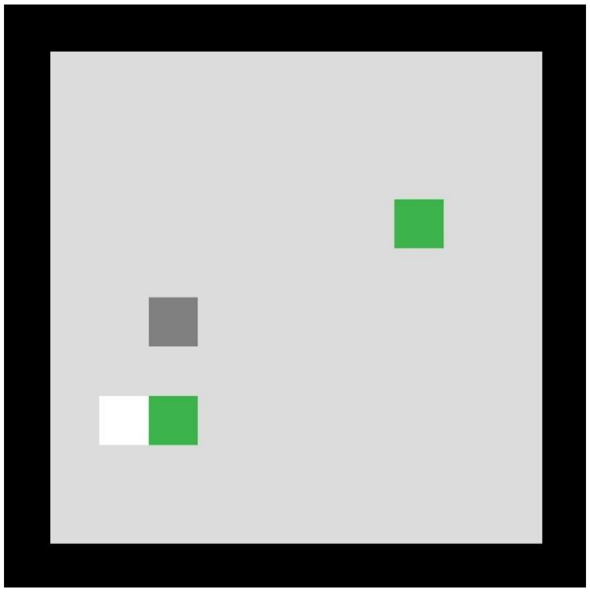


Figure 10: Box-World (Zambaldi et al., 2019) Configuration-1.

The second configuration consists of two locks (dark green) whose key boxes are the same (Figure 11). Therefore, one of the locks tries to distract the agents (the purple one) from the path of the gem's lock. Also, the agent should open only one lock before reaching the gem box, so this configuration's solution length is one step. In this configuration, the agent should notice that the dark green key box should be visited, and the agent should go to the adjacent gem box (white). If the agent goes to the distracter box (adjacent to the purple box), the -1 point penalty point will be given to the agent (Table 2).

Table 2: Rewarding of the Box-World objects.

Object	Agent	Gem	Key	Lock	Distractor
Color	Grey	White	Any Color	Any Color	Any Color
Reward	0	10	0	1	-1

The last configuration of the Box-World environments contains two pairs of keys and locks (in Figure 12). Also, no distractor path directs the agent to a dead end. Therefore, this configuration can be assumed of a more challenging version of the first configuration. To pass the episode optimally, it must understand the lock-key network from the gem object and trace back to the first key. This configuration needs the ability to understand object relations in a more complex way. Also, the agent should be robust in terms of color perception because the box's colors change in every episode.

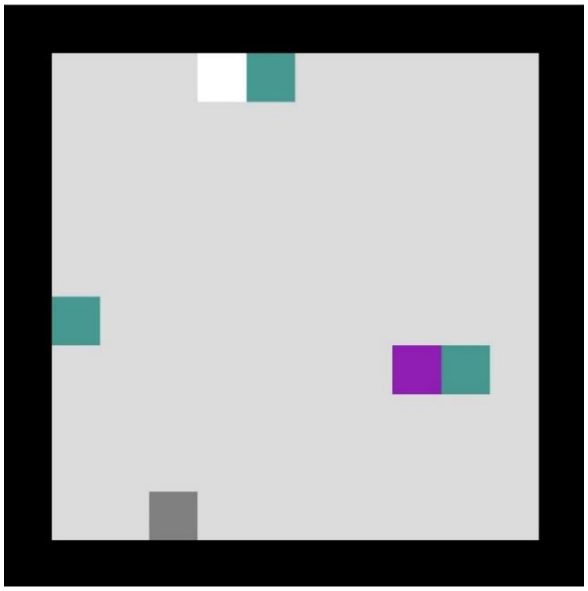


Figure 11: Box-World (Zambaldi et al., 2019) Configuration-2.

Moreover, more than one key-lock pair whose colors change will be more challenging than the first configuration in color robustness. The agent's exploration skill should be more in this configuration than the first and second one because the agent should recognize a more complex sequence of the boxes. In order to recognize this sequence, the number of random actions of the agents should increase. Moreover, in order to decrease the learning time of the agent, every step may be penalized. In this way, the agent will converge to the optimum path more easily.

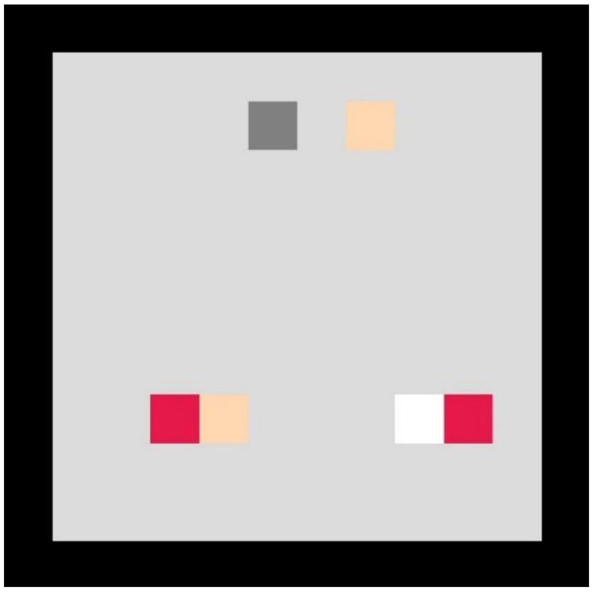


Figure 12: Box-World (Zambaldi et al., 2019) Configuration-3.

The Box-World environment provides a procedural model to generate primary cause and effect experiments in different difficulty levels. Solver algorithms for these environments can be easily written manually, but the experiment's main purpose is to create a solver with a self-learning capability. Then, more complex causality problems may be solved by the developed algorithms. Most of the phenomena in real-life work on causality, and understanding the mechanics of the cause and effect by self-learner tools will help humanity in real-life problems. Difficulty levels of the Box-World configurations can be increased, but to avoid hardware limits, more simple cases are selected in the configuration where they can still reflect the fundamental problem of the Box-World environment. If the complexity of the configurations increases, the agent's training process takes a longer time.

Unlike Box-World, the Relational-Grid-World environment does not change the objects' position in each episode because locating the objects randomly may cause unsolvable configurations. In future work, this feature will be added to RGW, and the agent must be robust for the positional changes of the objects. Therefore, RGW configurations are generated manually in this thesis. The visuals of example configurations created in the RGW and BW environments are given in Appendix. Moreover, unlike Box-World, the objects have got different features like positional and key-lock relation.

On the other hand, the Box-World environment's difficulty level can be adjusted systematically from the solution length, the distractor number, and the distractor path length from the environment's configuration parameters. The RGW has not got specific conjuration parameters that help to adjust the level of difficulty. Moreover, RGW's visuals are more complex than the Box-World, but this is not an essential property to measure relation-based capabilities of the decision-making algorithms.

CHAPTER 5

RESULTS AND DISCUSSION

5.1. Preliminary Experiments For Parameter Tuning

There are many hyperparameters of the networks because of the complexity of the network architectures. Before training all configurations of the environments, hyperparameter tuning is necessary. Reference articles (Zambaldi et al., 2019; Shanahan et al., 2020) of the networks have published some hyperparameters in their works. For the initial guess of the parameters, reference values are used. However, there is a need for fine-tuning these parameters, so a few pre-experiments are done for the MHDPA network. Then, all hyperparameters are synchronized with PrediNet in order to make reliable comparisons.

Firstly, the A3C framework's agent number, the outermost layer of the network, is tuned according to the limit of the processor of the working hardware. The number of parallel agents is selected as 12 because of the hardware limitation. Because of the difference of parallel computation framework with reference article, training took a longer time than the A2C and IMPALA (Espeholt et al., 2018) methods. However, this does not mean that there will be a difference in final results (Zambaldi et al., 2019) because core agent modules are the same. In order to accelerate the training process, experiments are done in different hardware simultaneously. Also, convolutional neural network architecture parameters are used directly from reference articles because trained environments are the same. Therefore, there is no need for a change in the input of the network.

Learning rates of the networks are selected according to preparation experiments. In Figure 13, the effect of the learning rate after 2000 episodes can be seen. There is a profound change in the performance of the model between 2E-2 and 1E-3 rates. With a large learning rate, there is no improvement in the neural network. When it is smaller than 1E-3, there are small changes in performance, so the 2E-4 value is selected for the learning rate to make time-efficient and performance-efficient training. With the increase in the learning rate, the success rate decreases because training speed decreases. Hence, with the same number of samples, a larger learning rate shows weaker performance. On the other hand, with the decrease in the learning rate, the network avoids the optimal point of networks' weight.

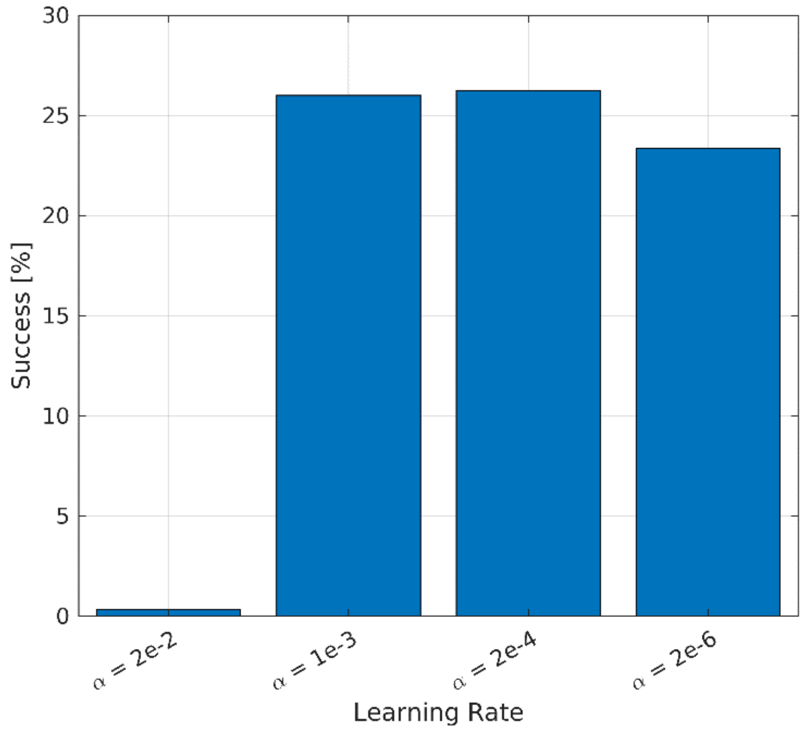


Figure 13: Learning rate experiments for Box-World with RN.

The same learning rate with Box-World is not the best learning rate for the RGW environment. From Figure 14, it can be seen less learning rate than the BW's gives better results in terms of the total reward getting from 3000 episodes RGW. RGW's success of measure is not boolean because there is a solution more than one. Hence, the success of the agent models is determined based on the total reward. The main reason for the difference in the larger RGW learning rate than BW is the difference in reward function rather than environmental dynamics. Also, Figure 14 shows that the agent's performance does not improve much with the decrease in the learning rate. However, higher learning rates have insufficient performance relative to the optimum rate.

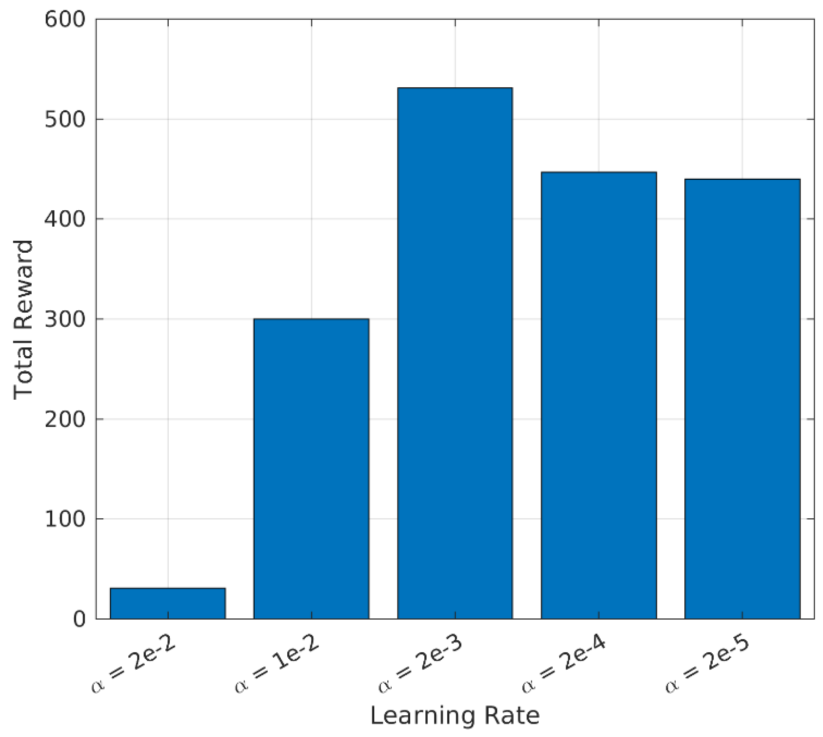


Figure 14: Learning rate experiments for RGW with RN.

Besides the learning rate, there may be dramatic changes in the network's weights during the training. In order to prevent this situation, gradients of the weights can be clipped with a specific value. Table 3 shows the performances of the model for different gradient values after 2000 episodes. From the table, it is easy to say that choosing 400 as a clip value is appropriate. However, there is no significant change in performance depending on the gradient clip. This is because the gradients are small enough to stay below the clip value most of the time.

Table 3: Gradient Clip-Performance Relation.

Gradient Clip	40	400	4000
Success	25.8%	26.3%	23.7%

As expected, the change on gradient clip is not effective on the model's performance and the learning rate. However, the number of heads parameter of the MHDPA network and the number of relation parameters of PrediNet models are essential as learning rates because the model's performance, computational efficiency, and post-process quality depend on these parameters. Figure 15 shows the success ratios for a different number of heads for the MHDPA network. It is easily seen that there is no dramatic change depending on the number of heads. However, the elapsed time of four training processes is distinguishable because tunable parameters grow almost proportional to head numbers. Hence, choosing heads number small enough to have efficient elapsed time but large enough to generate more information about relations of objects is favorable.

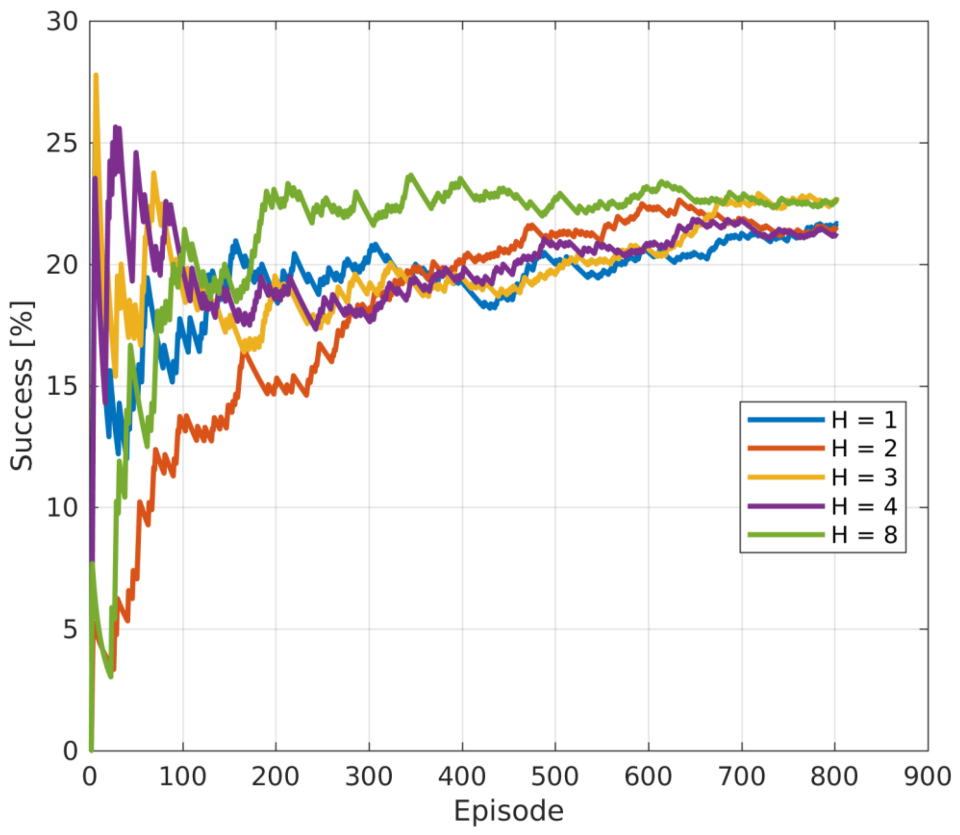


Figure 15: Number of head versus Success Ratios of MHDPA.

Experience replay is provided by a buffer that stores N number of observations, and then the stored observations are used to calculate the gradient of the network. Large buffer sizes increase the variation on sampled data, so the network trains healthier data. However, increasing the buffer size required more GPU memory. Therefore, the buffer size is used at the maximum value (40 observations) that the GPU memory can handle where the training is done.

Another critical parameter is the relation numbers in the PrediNet architecture. This value determines the number of relational representations of the network. For example, one of them may represent the color relations or may represent relative positional information. These relations may not be physical properties that are understandable by humans. In order to choose the number of relation values, a grid search is applied for five values. Results for each value can be seen in Figure 16. Success ratio increases with a decrease in the number of relations since the number of parameters in the network architecture decreases. Therefore, the network is trained faster for this specific configuration of the Box-World.

However, there is a nonlinear behavior at the value 32, and it has the best performance than other values. The possible reason for this situation is that the Box-World configuration for these experiments is more suitable to represent that specific value.

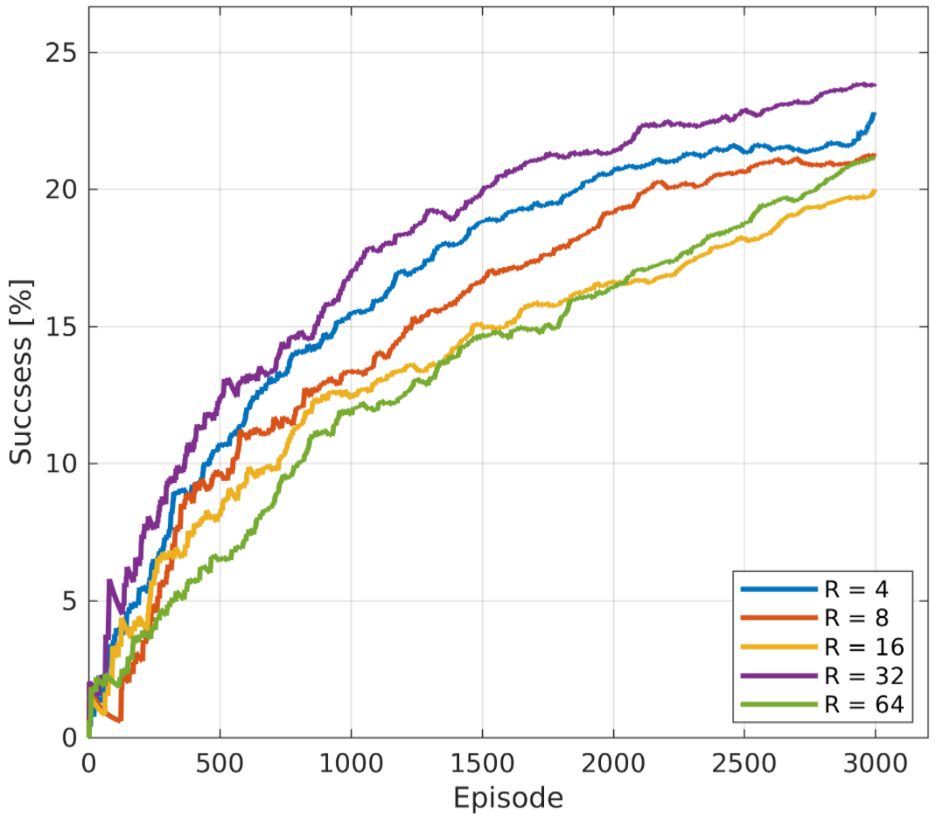


Figure 16: Dimension of relational representation versus success ratio of PrediNet.

On the other hand, the elapsed time of the training does not change much with an increasing number of relations. Hence, choosing a large relation dimension is more favorable because a larger representation space provides more information about the environment. However, the relation number to be used in the PredinNet module is chosen as 4, which is the value that provides the second-best performance, not to complicate the post-process operations. Creating a high number of relations for each pair of objects will increase the number of explicit relational values obtained and make it challenging to compare objects in post operations. The final parameter sets for the networks can be seen in Table 4.

Table 4: Network parameters.

Image Processing Section	
1st Convolutional Network (w/ ReLU activation)	
Output Channel Size	12
Kernel Size	2
Stride	1
2nd Convolutional Network (w/ ReLU activation)	
Output Channel Size	24
Kernel Size	2
Stride	1
Relational Module Section	
Option-1 *(Multi-Head Dot-Product Attention Network)*	
Number of heads	4
Query size	64 (for BW), 32 (for RGW)
Key size	64 (for BW), 32 (for RGW)
Value size	64
Stride (pooling)	1x1
Output size	26
Option-2 *(PrediNet)*	
Number of heads	4
Query size	64 (for BW), 32 (for RGW)
Key size	64 (for BW), 32 (for RGW)
Number of relations	8
Output size	26
Actor-Critic Section	
1st Fully Connected Layer	26 → 256 (w/ ReLU activation)
2nd Fully Connected Layer	256 → 128 (w/ ReLU activation)
3rd Fully Connected Layer	128 → 64 (w/ ReLU activation)
4th Fully Connected Layer (policy)	64 → 4 (w/ Softmax operation)
4th Fully Connected Layer (value)	64 → 1
Learning Parameters (for 12 actors)	
Optimizer	RMSprop without momentum
Optimizer epsilon	0.1
Optimizer decay	0.99
Gradient clip	400
Learning rate	2e-4 (for BW), 2e-3 (for RGW)
Buffer	40
Entropy scale	0.01 (for BW), 0.02 (for RGW)

5.2. Relational-Grid-World Experiments

5.2.1 RGW Configuration-1

Figure 17 shows the total rewards of the agents with MHDPA and PrediNet modules after the 60.000 episodes of training in the RGW Configuration-1. There is no significant performance difference between the two modules. PrediNet algorithm gives closer but less performance than the RN with MHDPA module since the PrediNet is not optimized for decision-making. It provides more processable information about the objects' relations while making reasonable decisions in the environment. The total reward returned from the environments is positive for both modules because Configuration-1 has no penalty objects.

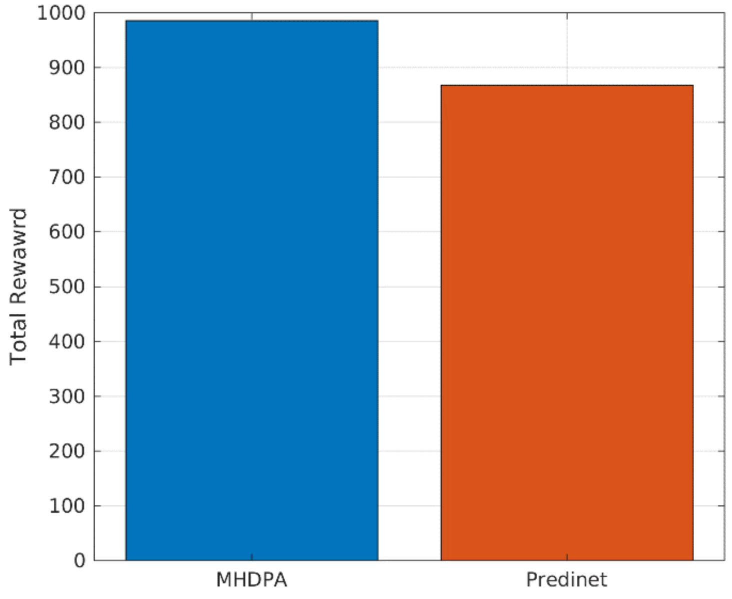


Figure 17: Total Rewards of the modules in RGW Configuration-1.

5.2.2 RGW Configuration-2

The difference between the second experiment on the RGW environment and the first one is mountain and pit objects. These objects give the agent a negative reward (penalty), so when we look at Figure 18, both modules' total reward is negative. Also, their final overall penalties are close. Hence, it can be said that there is not much difference between these two modules in the RGW Configuration-1 environment. Moreover, PrediNet provides extra information about the relations between the objects.

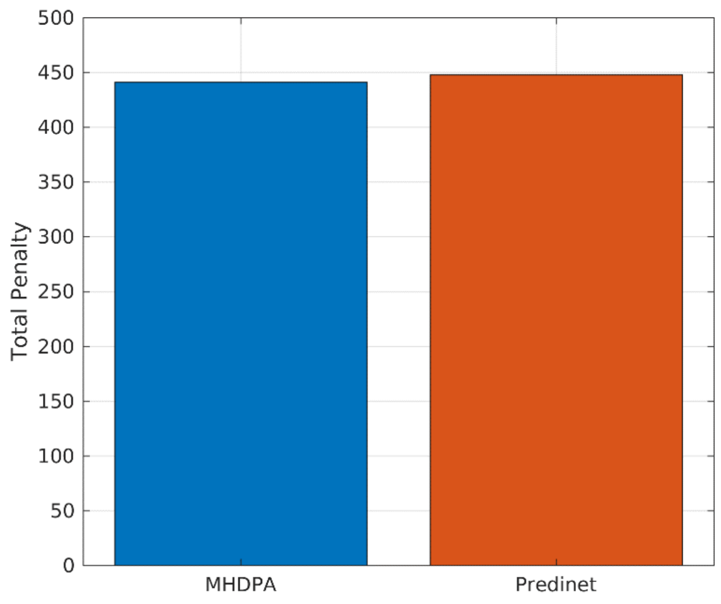


Figure 18: Total penalty of the modules in RGW Configuration-2.

Relational Network extracts interpretable information about its decision mechanism due to the nature of the MHDPA module. There is an object's attention matrix at the core of the module. The attention matrix for the agent objects at the first step in an episode can be seen in Figure 19 where:

- Object 1: Agent
- Object 2: Sword Object
- Object 3: Teleportation Object
- Object 4: Enemy

Figure 9 shows that the optimal path of the configuration for the agent is direct paths on objects at the following sequence: (2) Sword -> (3) Teleport -> (4) Enemy.

After the training process, the agent's attention matrix points to the enemy object, not the terminal state, because the enemy object is neighbor with the terminal state. Therefore, attention on terminal or enemy objects can be considered the same. It can be seen in Figure 19 that in addition to the enemy, the agent's attention is on the terminal, sword, and teleport objects, respectively. It is known that the agent is most interested in the enemy object (Figure 19). It can be seen which object is most interested in the enemy object through Figure 20. A teleportation object is an object that shows the most attention to the enemy object. This situation is consistent with the optimal path of the configuration because the agent should visit the teleported state to go to the enemy object and the terminal state.

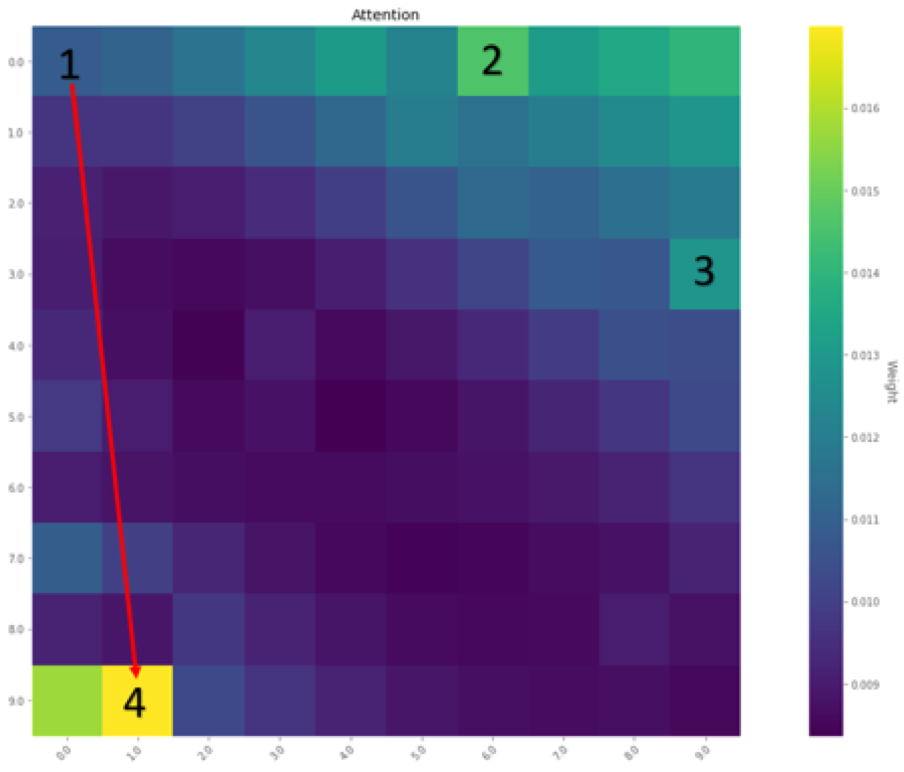


Figure 19: The agent's attention heat map in RGW Configuration-2.

Also, it can be seen that the second-most attention on the enemy comes from the sword objects, and this is what is expected in terms of the object sequence of the optimal path. There are no objects that have a strong relationship except swords and enemy objects. However, there is no direct evidence of this relationship from the attention heat maps of these two objects. The reason for this phenomenon comes from the bias of the statistical learning method. It determines the associations between the objects in the environment. If the agent had conducted controlled experiments, the direct link between the enemy and the sword object would have been noticed. This shows that the decision-making algorithm's symbolic methods are still weak compared to statistical methods. However, it is still more interpretable than other RL algorithms in terms of relational information.

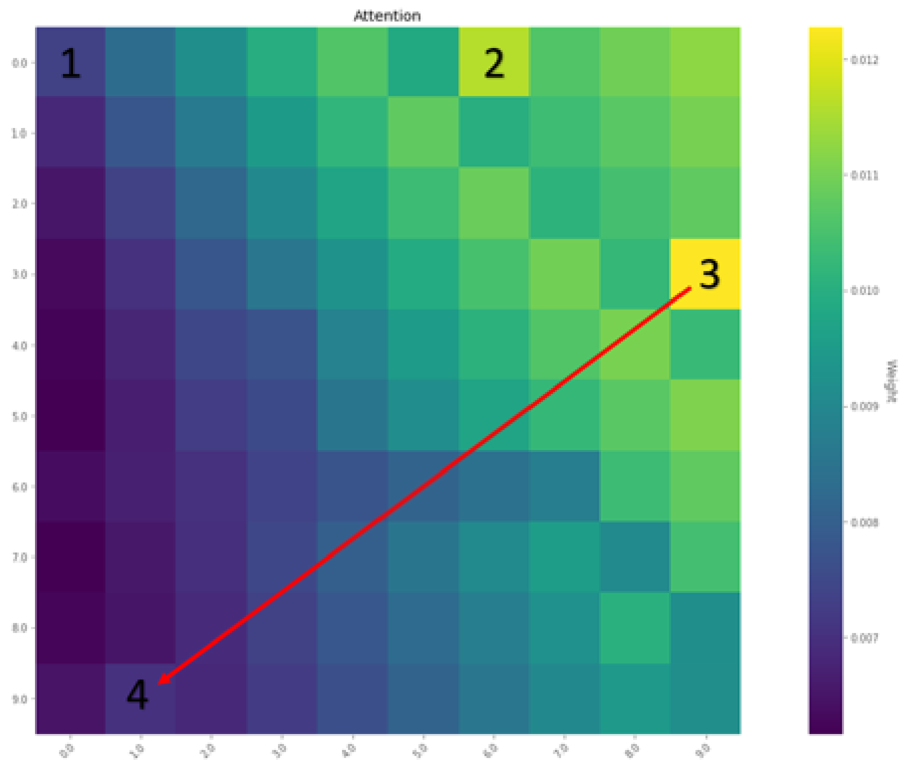


Figure 20: Attentions on the enemy object in RGW Configuration-2.

From the heat map of the attentions on the teleportation object (Figure 21), the most attending object to the teleportation is the sword object. This is also consistent with the optimal path sequence. Moreover, there is no strong attention on teleportation objects except the sword.

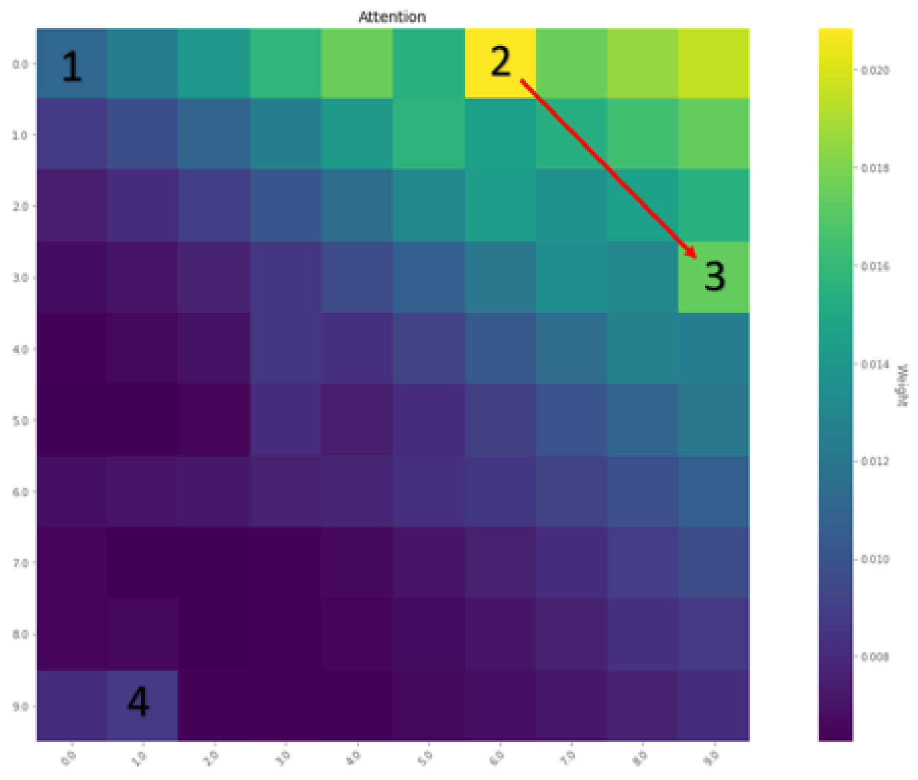


Figure 21: Attentions on the teleportation object in RGW Configuration-2.

Finally, Figure 22 shows that the attention matrix on the sword object. It can be seen that the most interest comes from the agent and adjacent grid regions of the agent. With this final step, the optimal path sequence of RGW Configuration-2 is reached in terms of high-level objects of the environment.

After training the RGW environment with Relational Network, the attention matrix gives the high-level interpretability of the dynamics of the objects. These results show that the RGW environment is suitable for relational information extraction experiments. Moreover, the PrediNet module shows comparable performance to the MHDPA module with the same hyperparameters. Also, according to the relational deep reinforcement learning study (Zambaldi et al., 2019), it extracts scalar relational values in different relational representation dimensions. Based on these experiments, it has been shown that the RGW environment is a suitable environment for such relational experiments, and the PrediNet algorithm can obtain extra information from the environment by providing a performance close to the equivalents.

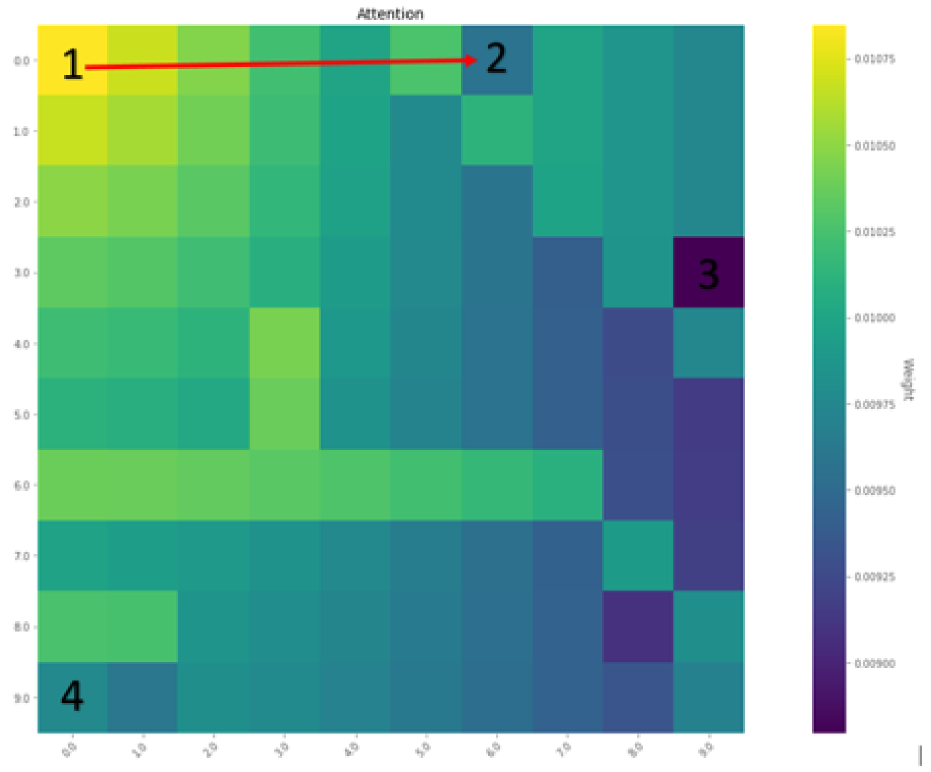


Figure 22: Attentions on the sword object in RGW Configuration-2.

5.3. Box-World Experiments

Three experiments are done in the Box-World environment. Each experiment is done in different configurations of the Box-World environment, which are mentioned in 4.2. The performance parameters of the MHDPA and PrediNet modules are determined from the mean and standard deviation of the last 1000 episodes. If the agent reaches the gem object, the episode counts as successful. Otherwise, episode counts are not successful. This way, it showed that the agent's success possibility randomly generated environments for each configuration.

5.3.1 BW Configuration-1

Configuration-1 consists of one pair of key-lock, and its solution length is one step. This configuration is chosen to present reference performance values. From Table 5, the performance of both models is almost above 30%. This means the agents of both learning algorithms show significant success in this environment. The chance that the agent accidentally finds the right solution is 2.3%. The PrediNet module's performance is ~7 points less than the MHDPA. The main reason for this, the PrediNet algorithm has not got a specialized architecture for this task, but it generates additional information rather than the MHDPA module. Unlike MHDPA, PrediNet can generate relational scalar values of the objects in different representational dimensions. Moreover, the standard deviation of the PrediNet is approximately half of the MHDPA's. This means PrediNet makes less accurate but more precise actions than the MHDPA.

The success rate of the MHDPA in relational networks is higher than 36.5% in the reference study because of the difference in training framework (A2C), the parallel agent number, and the length of the training time. However, these variables do not affect the final result of the network because they only accelerate training and increase the training sample variant. The core algorithms are the same as the reference work.

Table 5: Performance of the modules in Box-World Configuration-1 (mean and standard deviation).

Module	MHDPA	PrediNet
Success [%]	36.5 ± 1.3	29.3 ± 0.7

5.3.2 BW Configuration-2

The second BW configuration contains two locks with one key. One lock directs agents to the distractor path, which has a dead end, and the other directs the agent to the gem box. Therefore, the success ratios of the modules are less than Configuration-1, as expected. From Table 6, it can be seen that the successes of the modules decrease ~14 points for MHDPA and ~7 points for the PrediNet. Unlike Configuration-1, both modules' performances are close to each other. This means PrediNet architecture can replace the MHDPA algorithm while also providing extra information about the environmental dynamics. Moreover, as before, PrediNet architecture shows more precise performance than the MHDPA module.

Table 6: Performance of the modules in Box-World Configuration-2 (mean and standard deviation).

Module	MHDPA	PrediNet
Success [%]	22.8 ± 0.8	22.5 ± 0.5

5.3.3 BW Configuration-3

There are two pairs of key-lock objects in the third configuration of the BW environment. The complexity of the solution path is one level higher than Configuration-1, so it is appropriate to compare it with Configuration-1. From Table 7, it can be seen that performances of both modules decrease with increasing solution length. The performance decrease for the MHDPA is almost one-fourth, and for the PrediNet is almost one-third. This decline is completely coming from the increase in the solution length. Unlike the other configuration experiments, the PrediNet module shows three points better performance than the MHDPA module. Also, PrediNet is not more precise than MHDPA in these results. There may be an inverse relationship between the accuracy and precision of the modules.

Table 7: Performance of the modules in Box-World Configuration-3 (mean and standard deviation).

Module	MHDPA	PrediNet
Success [%]	8.8 ± 0.3	11.3 ± 0.9

According to the experiments and results summarized above, conducted in three different Box-World environments, it is understood that the MHDPA module performs better than PrediNet in most cases. Apart from this, it is mentioned in the MHDPA reference work (Zambaldi et al., 2019) that it performs better than regular network architecture. However, the PrediNet architecture performs closely with MHDPA and provides additional information about the environment. Therefore, the PrediNet architecture can be used to combine statistical and symbolic AI methods.

It was observed that the results obtained in the Box-World environment using the MHDPA module were lower than the reference article. The reference article's relational network can achieve over 80% success in Box-World environments with up to 10 solution lengths (the detailed calculation of performance is not included in the reference article. Therefore, the possibility of differences in performance measurements with this thesis should be considered). The biggest reason for this difference is the use of A2C architecture with 100 agents. Since A2C architecture uses hardware resources more efficiently than A3C, it can provide a more efficient learning base. Apart from this, due to the limitations of hardware resources, more than 12 agents were not used during the training in this thesis. The increase in the number of agents increases the variance of the samples taken from the environment and decreases the bias in the sample. Because of these differences, a lower-performing agent could be trained compared to the reference study. Realizing training processes with higher capacity hardware over cloud systems and A2C architecture in the agent algorithm will help overcome this situation. Apart from the factors related to hardware and algorithm structures, which are the main factors affecting the elapsed time, network parameters can also be considered as minor factors. The preliminary experiments determined that the number of heads and relation parameters also affected the elapsed time. Although these parameters do not change the network architecture, they cause the number of trainable parameters to grow.

Apart from the parameters showing the agent performance, another advantage of the PrediNet module over MHDPA is that the learning process progresses faster per unit time, not per episode. The most important reason for this is that the PrediNet module has a relatively simpler architecture than MHDPA, and the number of trainable parameters is less than MHDPA. Because as the number of parameters with trainable increases, the time required for the network to provide a stable performance also increases. Moreover, theoretically, this increase in training time will vary exponentially according to the number of trainable parameters. Therefore, a low elapsed time feature can be added to the advantages of the PrediNet module. Apart from this, by ensuring that the number of episodes per unit time is the same as MHDPA, higher performance training can be done by increasing the complexity in PrediNet. Limiting the number of steps made by the agent to 300 in the environment has enabled the training of both networks to be limited. The biggest reason for the variation in elapsed time between the two networks is not the long or short duration of each episode but the difference in the time the networks process the environment image to generate the required output for the agent. Parallel training of agents, an important advantage of the A3C method, ensures that each agent can be trained on different processor cores. Worker agents train global networks instead of training their networks so that the CPU can be used effectively.

CHAPTER 6

CONCLUSION

As a result of the increase in the amount of data, increase in processing capacities, and the improvements of deep learning methods, there are significant developments in decision-making algorithms. One of the biggest shortcomings of these algorithms is that they are often not generic for different data and environments, are inefficient in terms of data, and have weak interpretability. On the other hand, classical decision-making algorithms based on symbolic operations do not have these problems, but they do not have self-learning capability, which is more important than modern methods' problems. Therefore, using the advantages of modern methods and classical methods by fusing them, these problems can be solved. In order to achieve this combination, modern neural network architectures must be made open to symbolic manipulations. MHDPA and PrediNet architectures lay the groundwork for statistical methods to be applied to symbolic operations. In this thesis, various experiments were carried out in two different environments using these two architectures. One of the environments used is presented for the first time in the reference article using the MHDPA module. This is why both the MHDPA model and Box-World environment can be used as baselines for PrediNet and RGW environments.

As a result of the experiments, it has been seen that the PrediNet architecture has a lower performance than the MHDPA. However, it can play an important role in the bridge to be built between symbolic and statistical methods since the training time is shorter than MHDPA. It can extract propositional information suitable for symbolic post-processing from the environment. According to the reference article, this propositional information can be used downstream of the network pipeline to increase performance. The hyperparameters used for the two modules are chosen close to each other to provide a controlled experimental environment. If these parameters are chosen independently of each other, there may be significant differences in the performance of the two modules. Depending on these parameters of the architectures, inferences can be made about the estimated performance changes in the primary experiments section. Thanks to these tests, sensitivity analyzes are made for some parameters of the architectures. Apart from the deep neural network models used, it has been observed that the RGW environment, which is presented, can also measure the relational reasoning capacities of network architectures and, unlike the Box-World, it contains objects with more diverse features. It has been seen that the ability to adjust more systematic complexity in the Box-World environment is an advantage over the RGW environment. The image processing operation of RGW and BW environments are done in the same way, and it is shown that the architecture used for BW in the reference models is also sufficient for RGW. Since the RGW environment is visually more complex, using more powerful CNN structures during image processing will increase the agent's performance in the RGW.

The generalizability capabilities of the pipelines created using PrediNet, and MHDPA modules are still insufficient to reach artificial general intelligence because the agents are still explicitly trained for an environment. For example, when an agent is trained in an RGW environment with different visuals and the same mechanics as the RGW environment, it is clear that the agent's performance will be as an untrained agent. Although the problem does not change, since all the information learned by the agent is directly dependent on the visual characteristics of the environment, changing these visuals will cause them not to be able to use the information the agent learns. The agent needs to see the environment images only as a variable in the solution sequence and understand that the connection between the objects exists independently from the visuals. Apart from the generalizability problem, the data inefficiency problem is not solved with PrediNet and MHDPA methods. Because at the center of the agent algorithm lies a method trained with a statistical method, and large numbers of samples are needed to adjust a large number of tunable parameters of this method. In order to solve this problem, it would be appropriate to place algorithms based on symbolic operations at the center of agent architectures. Even if the PrediNet and MHDPA modules do not entirely solve these two problems, their solutions will depend on the development of object relations-based methods. In order to reach general intelligence, the most considerable ability brought by deep learning methods is the self-learning capability. Using this ability in symbolic methods would be an essential step.

Moreover, the relational reasoning measuring abilities of the Box-World and Relational-Grid-World environments are shown through the results. However, these environments allow the agent to be taught about direct or indirect object connections instead of cause and effect relationships in the environment. Although this situation is more similar to real life, the environment used can give direct cause-effect relationships to the agent, and train the agent in this regard will contribute to the theoretical studies.

Future studies plan to ensure that the RGW environment, which can be produced procedurally but needs to be controlled manually, can be generated for each episode without the need for manual control. In this way, high variance sampling is provided from the RGW environments, so the performances of the networks will increase. Moreover, RGW object icons can be recreated in order to increase the originality of the environment. Also, doing experiments with different networks (non-relational networks) and comparing the performances of different networks with baselines will increase the reliability of the RGW. In order to do these experiments, there is no need for extra modification on RGW. Also, the RGW environment can be used for real-life modeling problems like maze problems. Apart from the environmental considerations, also there is future work for the agent algorithms. Scalar relational values, which are the output of the PrediNet module, are planned to be used in closed-loop within network optimization. The information obtained from the environment through statistical methods will be processed with symbolic operations. The relationships between objects will be revealed in more precise ways, such as to cause and effect. Revealing such relationships will allow algorithms to recognize different situations instead of principles based on presuppositions forgotten by people in the environment.

REFERENCES

Ahmed, O., Trauble, F., Goyal, A., Neitz, A., Wüthrich, M., Bengio, Y., . . . Bauer, S. (2021). CausalWorld: A Robotic Manipulation Benchmark for Causal Structure and Transfer Learning. *International Conference on Learning Representations*.

Ahmed, O., Trauble, F., Goyal, A., Neitz, A., Wüthrich, M., Bengio, Y., . . . Bauer, S. (2021). CausalWorld: A Robotic Manipulation Benchmark for Causal Structure and Transfer Learning. *International Conference on Learning Representations*.

Barrett, D. G., Hill, F., Santoro, A., Morcos, A. S., & Lillicrap, T. (2018). Measuring abstract reasoning in neural networks. *35th International Conference on Machine Learning.* Stockholm.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 253–279.

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 679-684.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton: Princeton University Press.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *International conference on computers and games.* Italy: Turin.

Driessens, K., & Džeroski, S. (2004). Integrating Guidance into Relational Reinforcement Learning. *Machine Learning*.

Džeroski, S., & Driessens, K. (2001). Relational Reinforcement Learning. *Machine Learning*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., . . . Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *35th International Conference on Machine Learning.* Stockholm.

Garnelo, M., & Shanahan, M. (2019). Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 17-23.

Garnelo, M., Arulkumaran, K., & Shanahan, M. (2016). Towards Deep Symbolic Reinforcement Learning Marta. *arXiv preprint*.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural Computation* .

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-Critic Algorithms. *Advances in neural information processing systems*, 1008–1014.

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, 1106–1114.

Küçüksubaşı, F., & Sürer, E. (2021). Relational-grid-world: a novel relational reasoning environment and an agent model for relational information extraction. *Turkish Journal Of Electrical Engineering & Computer Sciences, 29*(2), 1259-1273.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 436-444.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., . . . Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *33rd International Conference on Machine Learning.* New York.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2019). Playing Atari with Deep Reinforcement Learning. *arXiv preprint*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . King, H. (2015). Human-level control through deep reinforcement learning. *Nature*, 529-541.

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven Exploration by Self-supervised Prediction. *34th International Conference on Machine Learning.* Sydney.

Raven, J. C., & Court, J. H. (1938). Raven's progressive matrices. *Western Psychological Services*.

Reizinger, P., & Szemenyei, M. (2019). Attention-Based Curiosity-Driven Exploration In Deep Reinforcement Learning. *arXiv preprint*.

Rummery, G. A., & Niranjan, M. (1994). On-Line Q-Learning Using Connectionist Systems.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., & Lillicrap, T. (2017). A simple neural network module for relational reasoning. *arXiv preprint*.

Schaul, T., Horgan, D., Gregor, K., & Silver, D. (2015). Universal Value Function Approximators. *32nd International Conference on Machine Learning.* Lille.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 85–117.

Shanahan, M., Nikiforou, K., Creswell, A., Kaplanis, C., Barrett, D., & Garnelo, M. (2020). An Explicitly Relational Neural Network Architecture. *37th International Conference on Machine Learning.* Vienna.

Stanley, K. O., D'Ambrosio, D. B., & and Gauci, J. (2009). A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Faculty Bibliography 2000s*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge: MIT press.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 58–68.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 81–199.

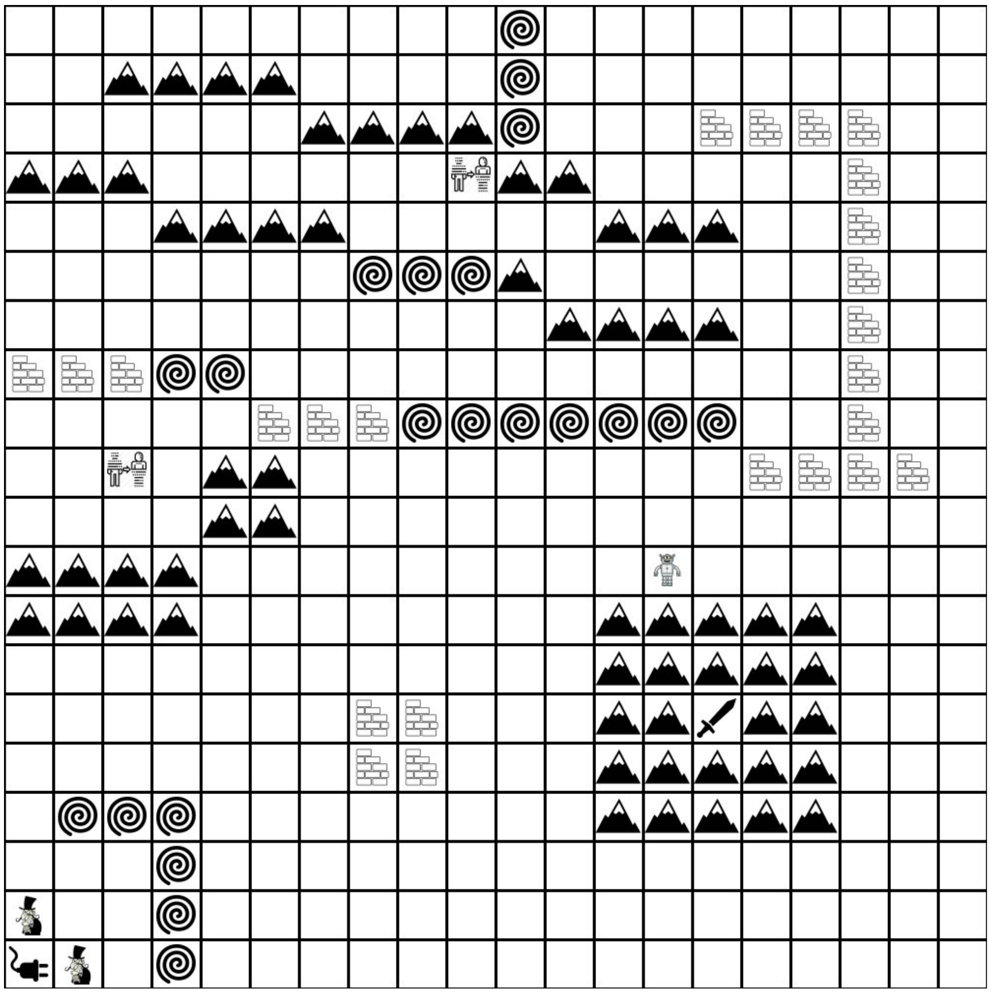
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998-6008.

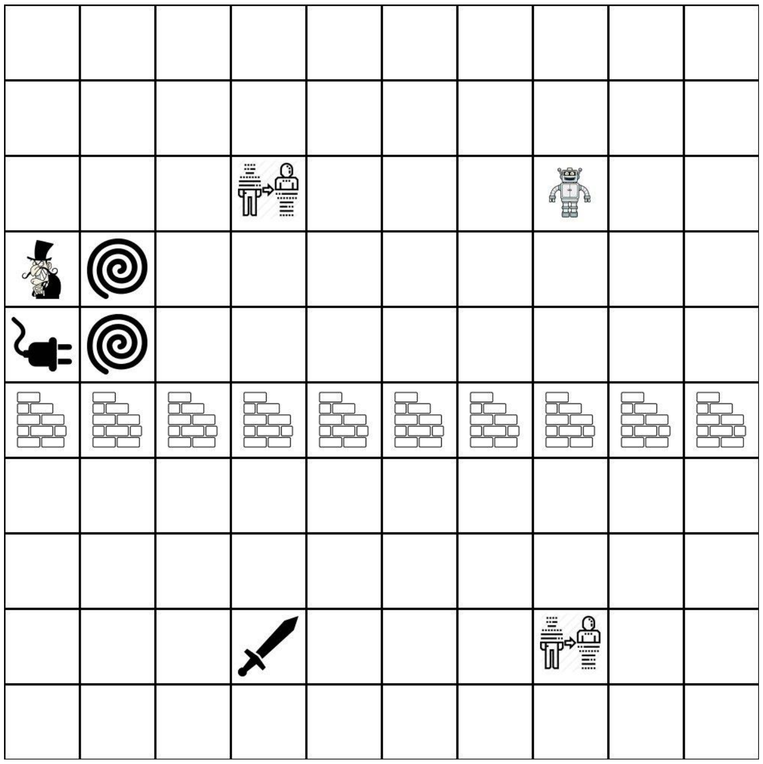
Watkins, C. J., & Dayan, P. (1992). Q-Learning. *Machine Learning*.

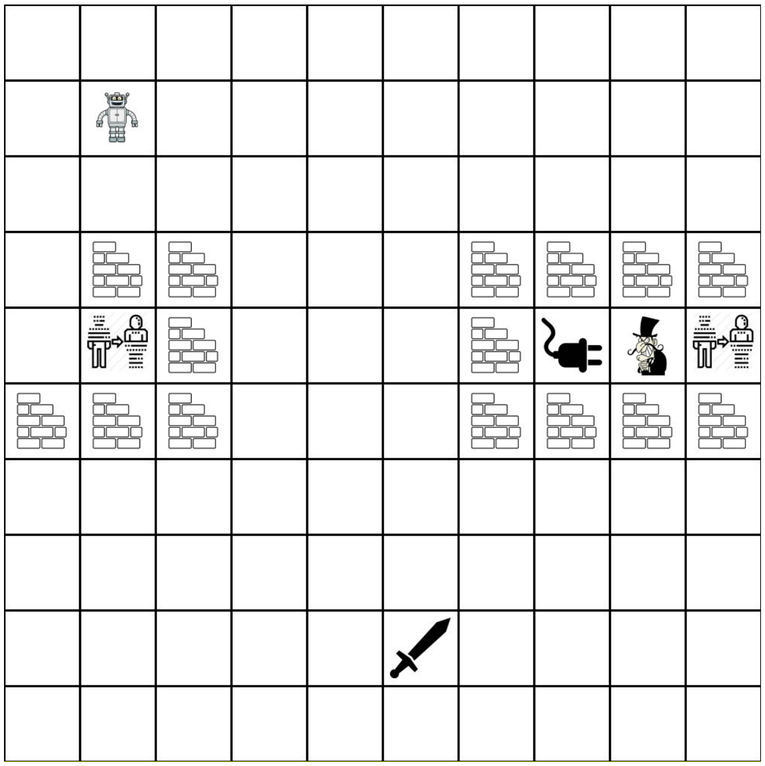
Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., . . . Ba. (2019). Deep Reinforcement Learning With Relational Inductive Biases. *International Conference on Learning Representations.* New Orleans.

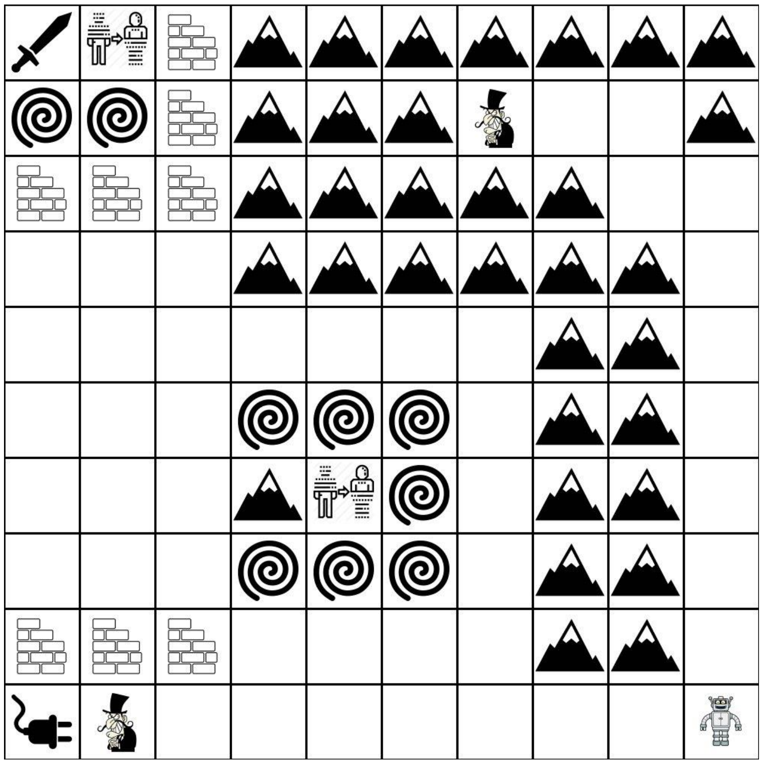
APPENDICES

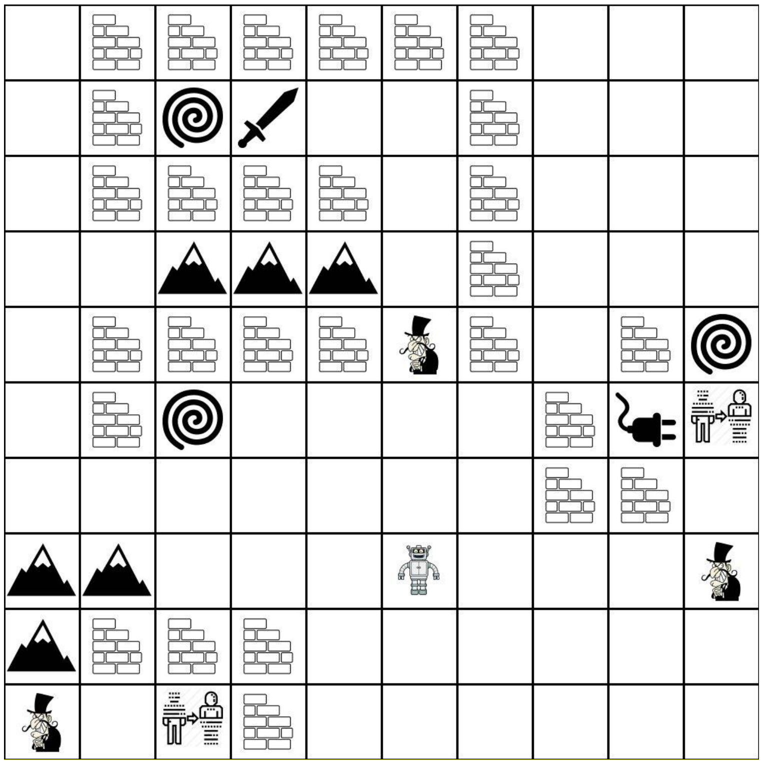
Example Configurations from the Relational-Grid-World Environment











Example Configurations from the Box-World Environment