university of groningen / faculty of science and engineering / bernoulli institute

MASTER'S THESIS

Performing a piece collecting task with a Q-Learning agent

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

FACULTY OF SCIENCE AND ENGINEERING

UNIVERSITY OF GRONINGEN

June 26, 2019

Author:
G. (Guillem) Casado Rubert
S3909115

Supervisor:
dr. M.A. (Marco) Wiering

Abstract

Since the early days of Artificial Intelligence (AI), researchers have tried to design intelligent machines capable of performing specific tasks with few instructions. In the 1950s, Machine Learning (ML) appeared and proposed that the goal might not be to design intelligent machines but machines that are able to learn from data. In the field of ML, Reinforcement Learning (RL) focused all the efforts on designing machines, referred to as agents, which are able to learn not from external data but from data derived from the own machine's experiences. The key concept of RL is to force agents to learn by providing them rewards depending on the outcome of each of its experiences. Many studies have proposed different approaches to RL systems and found applications in the industrial and manufacturing domain such as supply chain management, robot navigation and control and chemical reaction optimization.

The main aim of this thesis is to design an agent with a behaviour based on Reinforcement Learning, capable of performing tasks which could be extrapolated to activities and processes in an industrial environment. Specifically, the studied activity is the navigation control of a robot tasked to collect pieces placed in a two-dimensional environment. The algorithm used to guide the agent's learning process is one of the most known and used RL methods, Q-Learning. An Artificial Neural Network (ANN) structure, the MultiLayer Perceptron (MLP), is used to approximate the values used by the agent to decide which action to take in each situation. The experiments are designed in order to validate the capability of the agent to perform the task and compare the effect and results of several improvements implemented.

The results of the experiments validate the capacity of the agent to perform the task with acceptable results but indicate the agent is able to collect all the pieces in different environment configurations only when the improvements are implemented. These improvements are the addition of an experience replay memory and the observation strategy thanks, to which the agent knows what is it surrounded by. During the experimentation, comparisons between environment configurations and task complexity are done.

Acknowledgements

First and foremost, I would like to begin by thanking dr. Marco Wiering for accepting to be my supervisor and for the encouragement and kind support he has given to me throughout my Master's Thesis. His guidance, advice and ideas were the key to succeed in the fulfilling challenge this project has been. I would also like to particularly thank the several discussions and meetings we had regarding issues related to this thesis, as well as those unrelated to it but still linked to theoretical concepts of Reinforcement Learning.

Additionally, I would like to thank my home university, *Universitat Politècnica de Catalunya*, and the University of Groningen for giving me the opportunity to do my thesis abroad and seize the qualities of this experience.

Finally, I have to express my utmost gratitude to my family for their unconditional support and the confidence they always provide me to achieve all my goals.

The realization of this Master's thesis would not have been possible without all of them.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
MDP	Markov Decision Process
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
MLP	MultiLayer Perceptron
MSE	Mean Squared Error

1 Introduction

Artificial Intelligence was originally defined, amongst many other definitions, as the science capable of enabling machines do things that would require intelligence if done by humans [1]. At the beginning, in the early 1950s [2], it was a field treated both as a philosophical and a mathematical matter and researchers theorized about its promising future impact. While AI (Artificial Intelligence) was still an academic discipline, a new subfield was born, Machine Learning. This subfield posits that machines are not only meant to do things that would require intelligence if done by humans [1], but to learn from experience, learn by example and learn by analogy [2].

On the one hand, the success of implementing these last mentioned ideas is more than proven. Some examples or fields in which they have been applied are cancer prediction [3], environmental hazard detection [4], facial expression recognition [5], robotic motion control [6], etc. Therefore, the possibilities of Machine Learning are almost unlimited.

On the other hand, the industry has always tirelessly worked so as to optimize its processes and to reduce the resources needed. As a result, a combination of the tools and mechanisms provided by AI and Machine Learning, and the needs of the industry sector, would match perfectly. This is the reason why this thesis is focused on learning, analyzing and applying a specific Machine Learning field, Reinforcement Learning. The intention is to work on these three tasks (learning, analysis and application) from two points of view simultaneously. Firstly, from the point of view of research and acquisition of new knowledge. But, secondly, from the point of view of its possible implementation into real processes and situations in fields such as industrial engineering and robotics.

1.1 Research Background

In this section, the research background is discussed. It is structured based on the scope of each presented field. Therefore, Artificial Intelligence is discussed firstly, Machine Learning, secondly, and finally we introduce the main field in which this thesis is based, Reinforcement Learning.

1.1.1 Artificial Intelligence

The main goal of Artificial Intelligence is, as mentioned previously, to be able to create and design machines capable of doing things that would require intelligence if done by humans.

The first thoughts related to AI being applied to physical machines or *computers* date from the early 1950s when Alan Turing crystallized ideas about the possibility of programming an electronic computer to behave intelligently in his seminal paper in the philosophy journal *Mind* [7]. However, AI has gone through different phases or stages, some more optimistic and promising than others.

The era of great ideas and great expectations in the 1960s, with advances like the *Advice taker* [8], a program to search for solutions to general problems of the world,

made by John McCarthy. The disillusionment and funding cutbacks in the early 1970s made AI researchers realize most of the problems they were trying to face were too broad and too difficult [7].

The first expert systems, capable of performing at a human expert level, were developed in the 1970s, such as *MYCIN* [9], *PROSPECTOR* [10] and *DENDRAL* [11]. In the 1980s/90s, expert systems technology was massively applied in several areas [7].

1.1.2 Machine Learning

Machine learning (ML [12]) is a subset of Artificial Intelligence. Often, ML and AI are mistaken for one another because both of them are entitled to study how to make machines *think*. However, ML focuses on a more specific goal. Instead of simply making intelligent machines follow static program instructions, ML explores the construction and study of algorithms that enable the computer to learn from and make predictions on data. The difference is to *learn* how to make data-driven predictions or decisions. One of the most widely quoted definitions of ML is:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [12]*

There are three types of ML problems or learning strategies:

- Supervised Learning: The computer is presented with example inputs and their desired outputs, given by a *teacher*, and the goal is to learn a general rule that maps inputs to outputs [13].
- Unsupervised Learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end [13].
- Reinforcement Learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly informing it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent [13].

All of these three subfields of ML currently often use *Neural Networks* as the tool to design and establish the machine intelligence and learning strategy. Neural Networks will be more deeply discussed in Chapter 2.

During the early days of AI, when it was still only an academic discipline, some researchers were interested in having machines learn from data. That was the moment in which, theoretically, Machine Learning was born. Back then, the first neural network theories commenced to appear, but they were forgotten because of the deficiency of results they were providing. When the neural network theories started to flourish again in the mid-1980s, ML was reborn and reorganized as a separate field [2].

1.1.3 Reinforcement Learning

After briefly describing AI and ML, the last and more specific field guiding this thesis is presented. Reinforcement Learning (RL) is a subset of Machine Learning, therefore, a subset of Artificial Intelligence. As mentioned in the last subsection, RL does not learn from externally collected data but from data derived from the agent's experience. The theoretical description of RL will be presented in Chapter 2.

In order to understand the history and origin of Reinforcement Learning we have to know it came from different threads, from the learning theory of trial and error and from the problem of optimal control. Intertwining these two threads we get to what is known as the modern RL approach the origin of which is dated from the early 1980s [14].

There have been exciting achievements accomplished by RL such as autonomous helicopter flight [15] in 2003 or control for a biped robot walking on uneven surfaces [16] in 2006. But, first and foremost, in 2016 a group of RL experts from *Google DeepMind* managed to develop a program able to beat professional *Go* players for the first time [17]. After *DeepBlue* was able to beat the chess world champion Gary Kasparov in 1997, *Go* was the next milestone for AI as it is considered the most challenging of classic games.

Due to the great potential Reinforcement Learning offers, there are no boundaries for its possible applications and future impact in all types of fields.

1.2 Thesis Goals and Research Questions

The main goal of this thesis is defined as follows:

Evaluate the possibility of designing and implementing an agent with a behaviour based on Reinforcement Learning, capable of performing tasks that could be extrapolated to activities and processes in an industrial environment.

This broad goal can be split into more specific questions that assist with the evaluation of the results with regard to the already described overarching goal. The research questions enquired at the beginning of this thesis are the following and are sorted from general to specific:

- Is a Reinforcement Learning agent able to learn behaviour policies only by interacting with the environment and show results comparable to those of a programmed industrial robot?
- Despite the relative simplicity of the Q-Learning Algorithm, can this algorithm be used to learn the optimal policy?
- What adaptability do Artificial Neural Networks, such as MultiLayer Perceptrons, provide and how do they perform in this type of problems?
- Which is the minimum information the environment should provide to the agent in order to accomplish the proposed task?

1.3 Thesis Outline

This thesis is organized as follows. In Chapter 2, the theoretical background about Reinforcement Learning and Neural Networks is provided. Additionally, more concrete aspects such as the *Q*-learning algorithm and the Exploration vs. Exploitation trade-off are discussed. The aim of this chapter is to clarify all these concepts and establish all knowledge before describing the proposed system. In Chapter 3, firstly, a description of the designed environment is presented. Then, the *Q*-Learning agent, its structure and all its specifications are explained and discussed. In Chapter 4, all experiments are shown and described. All the parameters from the environment, agent and experimentation are referenced in this chapter. Chapter 5 provides and discusses the results derived from the experiments described in the previous section. Lastly, in Chapter 6, the Research Questions are answered based on the results and discussion from the previous chapter. In addition, future improvements are proposed and discussed.

2 Theoretical Background

2.1 Introduction

In this chapter, the fundamental theoretical background is provided in order to set the basics to be able to contextualize and understand what is discussed in subsequent chapters. This theoretical background begins with a brief introduction to Reinforcement Learning. Following this first section, Reinforcement Learning theory is extended with the description of Markov Decision Processes. Moreover, the specific RL method used in this thesis, *Q*-Learning, which describes how an agent can learn to behave based on a policy and on the interaction with the environment, is disclosed. Next, we discuss the need to implement Artificial Neural Networks in *Q*-learning and the process to do it. Lastly, the Exploration vs. Exploitation dilemma affecting *Q*-Learning is presented.

2.1.1 Reinforcement Learning Definition

Reinforcement Learning (RL) is based on the idea of learning by interacting, discovering and experiencing. The learner, technically known as the *agent*, has the goal to behave in the way that would maximize the reward received from the *environment* after taking an action [18]. The simplest scheme of Reinforcement Learning, shown in Figure 2.1, consists of the two mentioned elements and shows the interaction between the environment, the world, and the agent, the learner. When the agent takes an action, the environment changes its state providing the agent with information representing this new state. The agent also receives a reward signal which evaluates the appropriateness of the action taken. This reward does not have to necessarily be the result of the last action taken, but can be the result of the combination of the previous actions in the experienced states.

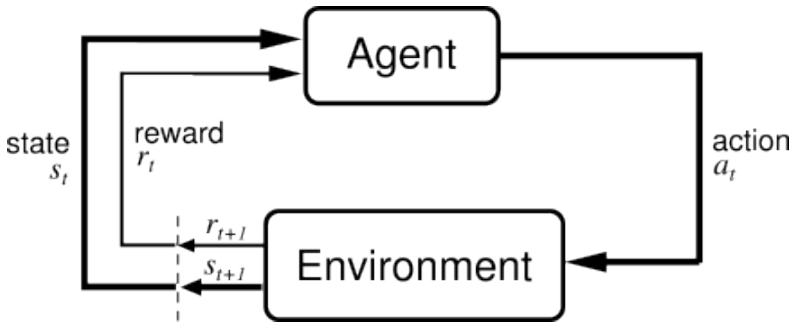


Figure 2.1: Schematic representation of the RL agent and the environment interaction. Taken from [14].

The main goal of Reinforcement Learning is to make the agent able to find the optimal policy to follow, so as to reach the primary goal, obtaining the maximum cumulative reward [14].

2.1.2 Markov Decision Processes

Markov Decision Processes (MDPs) are a way to formalize, mathematically, sequential decision-making processes, where actions not only affect immediate rewards, but also future states and rewards [14]. This formalization follows the *Markov property*, which means that the future must be independent of the past given the present. In other words, knowing the action taken, every state must contain sufficient information in order to be able to determine the next state.

Almost all Reinforcement Learning problems can be formalized as MDPs. That is the reason why, in an important number of cases, the Reinforcement Learning problem is modelled as the finite and discrete-time version of this formalization.

A discrete MDP would be modelled as the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$[18]–[21] where:

- $\mathcal{S}$ is a finite set of states ($s_i \in \mathcal{S}$)
- $\mathcal{A}$ is a finite set of actions ($a_i \in \mathcal{A}$)
- $\mathcal{P}$ is a state transition function $\mathcal{P}(s, a, s')$ or $\mathcal{P}^a_{ss'}$ where $0 \leq \mathcal{P}(s'|s, a) \leq 1$ and represents the probability of transitioning from state s to s' if action a is taken.
- $\mathcal{R}$ is a reward function $\mathcal{R}(s, a)$ or $\mathcal{R}^a_s$ and provides the agent with the reward for performing action a while being in state s.
- $\gamma \in [0, 1]$ is the discount factor with which the agent will be encouraged to look for long-term rewards or not.

Assuming the MDP is correctly defined, the next step is to find the *rules* which will assist the agent to take the optimal action in each situation. These sets of *rules* are called policies. The policy is what will guide the agent in its decision-making process. In order to determine and evaluate a policy two functions are used. The first one is called the *Value Function* or *State-Value Function* defined as in equation 2.1. The *Value Function* represents how good it is to be in the state s while following the policy π.

$$V_\pi(s) = \mathbb{E}[G_t|s_t = s] \tag{2.1}$$

Where G_t is the total discounted reward from time-step t as shown in equation 2.2.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

The second function is called the *Action-Value Function*, also known as *Q-function* and it is defined in equation 2.3.

$$Q_\pi(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a] \tag{2.3}$$

In this case this function represents how good it is to take action a while being in state s and following policy π.

As the goal is to find the optimal policy for the agent, using the Bellman Expectation Equation defined in equations 2.4 and 2.5, the optimal value function and action-value function can be defined as shown in equations 2.6 and 2.7.

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(\mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} V_\pi(s')) \tag{2.4}$$

$$Q_\pi(s) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a \in \mathcal{A}} \pi(a'|s') Q_\pi(s', a') \tag{2.5}$$

$$V_*(s) = \max_a Q_*(s, a) = \max_a (\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_*(s')) \tag{2.6}$$

$$Q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_*(s') = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a') \tag{2.7}$$

This last two equations are known as the Bellman Optimality Equations [18]–[20].

2.2 Q-Learning

2.2.1 Q-Learning Definition

Most of the Reinforcement Learning algorithms designed to learn how to behave in MDP environments are based or inspired on the principles stated in the previous section. In this section, the algorithm on which the entire project is based is presented.

Q-Learning [22] is one of the most often used RL algorithms. It learns to predict the quality, in terms of expected cumulative reward, of an action in a specific state (*Q*-value) [23]. Moreover, as it is a model-free reinforcement learning algorithm [22], [24], the agent does not have a model representation of the environment, it simply learns and acts without knowing the changes being caused in the environment. The methods in which an environment model is known are called model-based. In this case, the agent knows approximately how the environment is going to evolve. This is the reason why model-based methods focus on planning while model-free ones focus on learning [14].

The main aim of *Q-Learning* is to determine the optimal *Q*-function ($Q^*(s_t, a_t)$) for the agent. The actions taken by the agent will be decided by the behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$. There will also be considered an alternative successor action decided by the target policy $A' \sim \pi(\cdot|S_t)$. As *Q*-Learning learns from one policy and acts according to another different policy, it is also classified as an Off-Policy Learning algorithm. The goal is to update $Q(s_t, a_t)$, at each step, towards the *Q*-value obtained from the target policy π [14]. Then, the update is done following equation 2.8.

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R_s^a + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{2.8}$$

Where $\alpha \in [0, 1]$ is the learning rate. The learning rate specifies how much the *Q-function learns* from each update. A representation of this method is shown in figure 2.2.

Combining what has been described about the Bellman Optimality Equations and the basics of the *Q*-Learning algorithm we can obtain the following conclusion. Let $Q^*(s, a)$ be the optimal expected cumulative reward received by being in state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$. If each action would be taken in each state an infinite number of times, the *Q*-values from the *Q*-function would converge to $Q^*(s, a)$ with a probability of 1 [18], [25]–[27].

An explicit procedure to implement the *Q*-learning algorithm is provided in Algorithm 1.

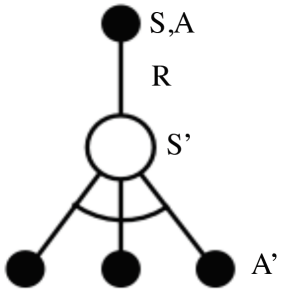


Figure 2.2: Q-Learning Algorithm basic representation. Taken from [14].

Algorithm 1 Q-Learning Algorithm

initialize $Q(s, a)$ arbitrarily
for each episode **do**
  get initial state s
  repeat
    select a using policy derived from Q (e.g., ϵ-greedy)
    take action a, observe next state s' and obtain reward r
    update $Q(s, a)$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  until s is terminal state
end for

A method to store dynamically the calculated and optimized Q-values is by a Table Look-up. This table represents the Q-function by having an entry $Q(s, a)$ for each state-action pair s, a and updating its values according to the result of the Q-Learning updates. The dimension of this table grows as the state space does too. Due to this, when the complexity of the problem faced or the environment in which the agent is interacting increases, the Table Look-up method becomes unfeasible.

2.2.2 Function Approximation for Q-Learning

In order to scale up the Q-Learning method to be able to solve larger MDPs, the use of Function Approximators is one of the best solutions. There are different types of Function Approximators, e.g. Linear Combinations of Features, Decision Trees, Nearest Neighbour, etc. But the most used in RL are Artificial Neural Networks (ANN) also simply known as Neural Networks.

Artificial Neural Networks are a tool that aims to simulate the physiological behaviour of the human brain in terms of information processing [28]–[30]. As in the human brain, the basic component of these networks is the artificial neuron.

The structure of a neuron, as shown in Figure 2.3, can be treated as the integration of four parts. A group of inputs and weights, a transfer function, e.g. sum, an activation function, e.g. sigmoid function and a bias [28], [30]–[33]. In mathematical terms, a neuron k is described by equations 2.9 and 2.10.

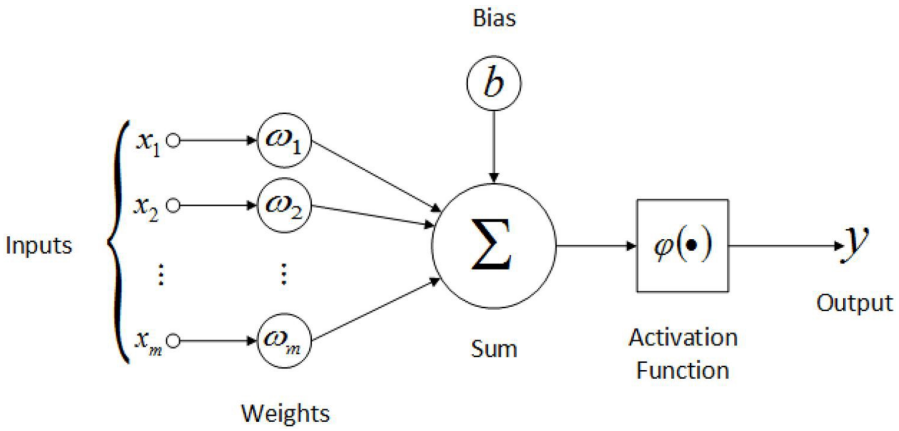


Figure 2.3: Nonlinear model of an artificial neuron. Taken from [34].

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.9}$$

$$y_k = \phi(u_k + b_k) \tag{2.10}$$

Where $x_1, x_2, ..., x_m$ are the input signals, $w_{k1}, w_{k2}, ..., w_{km}$ are the synaptic weights of neuron k, u_k is the linear combined output due to the input signals, b_k is the bias, $\phi(\cdot)$ is the activation function and y_k is the output signal of the neuron.

There are multiple types of ANN architectures based on the problem to solve, e.g. image classification, speech recognition, etc. These types are, for instance, Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), etc. An example of an ANN is shown in Figure 2.4

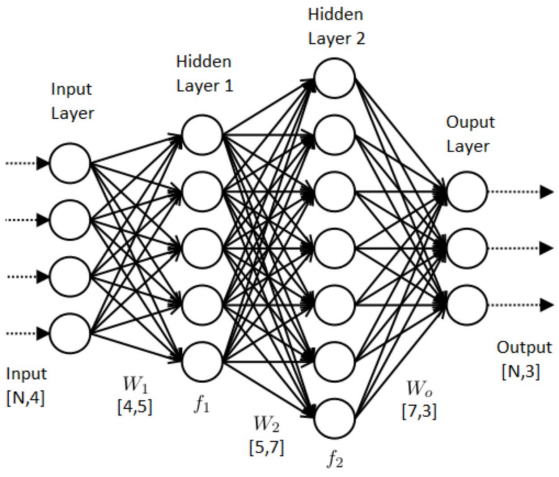


Figure 2.4: Artificial Neural Network example structure. Taken from [35].

In this example, there is one input layer with 4 neurons, two hidden layers with five and seven neurons respectively and an output layer with three neurons. All four layers are fully connected in a forward propagation way. This is the type of structure used in this project. But instead of two hidden layers, only one is used. The reason for this is that an ANN with a single hidden layer containing a large enough finite number of

non-linear neurons can approximate any continuous function on a compact region of the network's input space to any degree of accuracy [14], [36]. This type of structure is one of the simplest ones and it is commonly called *MultiLayer Perceptron* (MLP) [37].

As mentioned previously, MLPs can be used to approximate the Q-function from the Q-Learning algorithm. Theoretically, the inputs of the MLP might contain the information related to the agent's state and the action taken so as to determine the derived Q-value as the only output ($Q(s,a)$). However, another configuration is possible and proved to perform better. This configuration defines as many outputs as actions the Q-Learning agent is able to perform [38]. Then, the MLP provides the Q-value of each action to its respective output neuron. Moreover, the main idea is to train the MLP, changing its parameters, until its behaviour is similar to the expected Q-function. In order to accomplish this, a squared-error measure between the value provided by the MLP and the one provided by the Q-Learning algorithm update can be used as cost function:

$$error_a = (y_a - Y_a)^2 \tag{2.11}$$

Where y_a is the output value provided by the MLP, Y_a is the target value by the Q-Learning algorithm update and a is the action associated to each output of the MLP. Therefore, the error can be reformulated as:

$$error_a = (Q(s,a) - (R + \gamma \max_{a'} Q(s',a'))^2 \tag{2.12}$$

Then, common gradient descent techniques (like the *back-propagation* learning rule) can be applied to adjust the weights of the MLP so as to minimize this error. The back-propagation method [39] is the main algorithm used to train ANNs towards the minimization of the error previously mentioned. This method is based on the idea of propagating the error measured in the output layers through the previous layers. These changes depend only on the relative contribution of each of the neurons to the error received from the next layer. The back-propagation method is described in mathematical terms in equations 2.13 to 2.17 [40].

The error of one update i is computed as the sum of the squared errors of each output a, where A is the total number of output neurons:

$$E_i = \sum_{a=1}^{A} error_a \tag{2.13}$$

While the total error or loss is computed as the average of the errors of each update i when more than one update is used to adjust the network weights, where n is the total number of update errors:

$$E = \frac{1}{n} \sum_{i=1}^{n} E_i \tag{2.14}$$

In order to reduce this total error, the weights of the network are updated by a small adjustment. This adjustment is defined by the gradient of the error and proportional to a factor α called learning rate, a factor performing a similar job to the one described in section 2.2.1. :

$$\mathbf{w} \leftarrow \mathbf{w} - \Delta\mathbf{w} \tag{2.15}$$

$$\Delta \mathbf{w} = \alpha \nabla E(\mathbf{w}) \tag{2.16}$$

$$\nabla E(\mathbf{w}) = (\frac{dE(\mathbf{w})}{dw_1}, \frac{dE(\mathbf{w})}{dw_2}, \cdots, \frac{dE(\mathbf{w})}{dw_N}) \tag{2.17}$$

Where $\mathbf{w}$ is the set of all weights, $\nabla E(\mathbf{w})$ is the gradient of $E(\mathbf{w})$ and w_i is the weight between two specific neurons.

This adjustment rule is applied after each Q-Learning algorithm update [41], [42] and it is repeatedly applied until E fails to descend.

2.2.3 Experience Replay

Approximating the Q-function with an ANN such as the MLP when the state space is complex is a positive and proven solution. Despite this, it has some drawbacks. An issue usually affecting this type of MLP implementations into Q-Learning systems is the correlation of the data extracted from the agent's experience. Gradient descent techniques, used in the learning process of the MLP, do not work well when the data used is in some way correlated. In the case of Q-Learning, as the agent learns from the data it is experiencing, there is an important temporal/chronological correlation in it. Experience Replay, a method firstly described in 1992 [43], is a method that alleviates this correlation [44].

The implementation of *Experience Replay* into the Q-Learning algorithm with ANN approximation is described in Algorithm 2 [44]. The aim of this algorithm is to store each $<s_t, a_t, r_t, s_{t+1}>$ transition set experienced by the agent at each step into what is called the *Replay Memory*. When the replay memory is filled with sufficient transitions, the agent will random uniformly sample a minibatch of the experiences from this replay memory. This set of transitions are used to train the Q-function approximator (the MLP in our case). If the replay memory is full, the newest transition experienced will substitute the oldest one. The reason why the temporal correlation is broken with this method is because the Q-function approximator is learning from randomly collected experiences from the replay memory and not from the transitions the agent is experiencing at each step.

2.2.4 Exploration/Exploitation Trade-off

All Reinforcement Learning agents learn from experience. Therefore, the decisions an agent makes in terms of selecting an action determine which type of experience they obtain and what they learn from it. Then, it is sometimes important to avoid the best actions, according to the agent, so as to get to new states that could, potentially, provide the highest cumulative reward in the future. If an agent always selects an action based on the highest Q-value (greedy strategy), it will probably find a suboptimal solution or not find any. Following a policy only based on the best actions (those with the highest Q-value) is called *Exploit*, while ignoring those best actions and select another action in order to discover possible new states and rewards is called *Explore* [45]. This is the reason why it is necessary to have a balance between exploring and exploiting. This need of balance is commonly known as the *Exploration/Exploitation dilemma* or *Exploration/Exploitation trade-off*.

Algorithm 2 Q-Learning with Experience Replay and ANN approximation

initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
initialize $Q(s, a; \theta)$ with random weights θ
for each episode **do**
  get initial state s
  repeat
    select a using policy derived from Q (e.g., ϵ-greedy)
    take action a, observe next state s' and obtain reward r
    store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    if s_{j+1} is terminal state **then**
      set $y_j = r_j$
    else
      set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta))$
    end if
    perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    $s \leftarrow s'$
  until s is terminal state
end for

One of the most used strategies is the *epsilon greedy* or ϵ*-greedy* method. This exploration strategy is based on the use of a designated value (ϵ) to calculate the probability of taking a random action (exploration) over the action with the maximum Q-value (exploitation) [14], [46]. The ϵ parameter will be set between zero and one, $\epsilon \in [0, 1]$. The procedure the strategy follows is:

1. Generate a random number n from the uniform distribution [0, 1]
2. If $n < \epsilon$
   (a) Select a random action $a \in \mathcal{A}$
   (b) Otherwise, select the action with the highest Q-value ($argmax_{a \in \mathcal{A}} Q(s, a)$)

However, the ϵ parameter is not the only adjustable parameter in relation to the ϵ-greedy strategy. There is a variation of this method called *Decaying* ϵ*-greedy* [14]. Decaying ϵ-greedy is based on the same idea as regular ϵ-greedy, but in this case, another parameter is used, the *Decaying Rate*. This decaying rate is applied to the ϵ parameter so as to decrease exploration in favour of exploitation as the agent learns.

3 Proposed Method

In the previous chapter, Reinforcement Learning and the Q-Learning algorithm were described in more detail. Possible scaling problems when the state space is too large were also presented and in this last case, the possibility of using MLPs to solve the problem was explained.

In this chapter, we describe and show the entire design process of the components needed to end up with the model of an agent capable of performing the task of collecting pieces. First of all, a description of the task the agent must develop is presented. Subsequently, the first component to be described in greater detail is the environment, its elements, the mechanisms by which the simulations are ruled and the reward system used.

Finally, the core part of this thesis is detailed. Presenting the design and implementation of the agent capable of collecting pieces within a simulated discrete environment. To describe this agent, we disclose its structure, as well as each of the improvements applied to it so as to reach the final goal.

3.1 Task Description

The main goal of this thesis is to design and implement an agent capable of accomplishing a specific task. This task consists of the four following subtasks or subobjectives:

- Stay within the limits of the environment.
- Avoid the walls placed in the environment.
- Collect pieces placed around the environment.
- Reach the goal location bringing the pieces previously collected.

Therefore, in order to consider the task as performed, the agent requires to accomplish the four subobjectives mentioned previously. Moreover, only performing the task described is not the only goal of this thesis because it can be completed in several ways, some more optimal or completed than others. The optimality and completion of the solution found by the agent depends on the following criteria:

- Number of pieces collected, out of the total, and brought to the goal location.
- Number of steps needed by the agent to complete the task.

To clearly evaluate the aspects mentioned previously, we have designed five different scenarios with different criteria for fulfillment. All of them are shown in Table 3.1, as well as the task evaluation result which would be assigned to each of them. The asterisk symbol ($*$) represents a *don't care* value of the task evaluation criteria in that scenario.

Task evaluation criteria	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Goal location reached	✗	✓	✓	✓	✓
Pieces collected	$*$	None	Some	All	All
Minimum number of steps	$*$	$*$	$*$	✗	✓
Result	Failed	Failed	Partially Performed	Suboptimally Performed	Optimally Performed

Table 3.1: Evaluation of the task performance based on the fulfillment criteria.

3.2 Environment

The environment design is one of the most crucial parts when experimenting with RL. In Subsection 2.1.1, its relevance and the information it provides to the agent were discussed. The main duty of the environment is to simulate the inputs the agent would receive in a real or non-simulated situation and modify itself according to the outputs of this agent.

For the purpose of this thesis, the design consists of a limited two-dimensional grid-based environment in which the agent must perform its task while interacting with the different elements contained in it. Each position in the grid is considered as a state [47].

3.2.1 Elements

The elements defining the environment along with the two dimensional grid are: the walls, the pieces and the goal location. To represent the configuration of the elements inside the grid and for it to be used by the Q-Learning agent, the grid and the elements are mapped into a three-dimensional integer matrix. This matrix consists of three layers, each of which represents the configuration of each element in the grid. The first layer represents the configuration of the walls, the second layer represents the target location and the third layer represents the configuration of the pieces. The dimension of each layer is the same as the dimension of the grid.

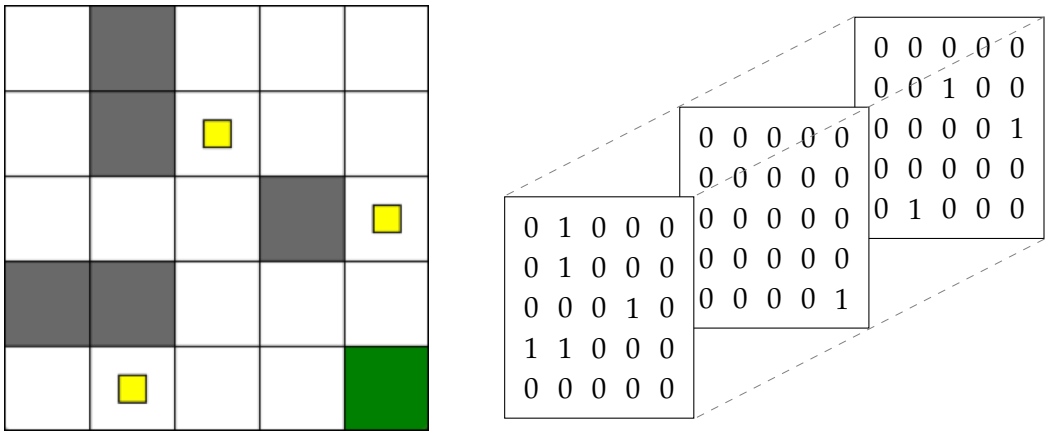
In an $M \times N$ grid, for every matrix element $E(x, y, z)$ where $x \in \{0, 1, \ldots, M-1\}$, $y \in \{0, 1, \ldots, N-1\}$ and $z \in [0, 2]$, the value of the element $E(x, y, z)$ will be 1 or 0 according to:

$$E(x,y,0) = \begin{cases} 0 & \text{if } E(x,y,z) \text{ is not occupied by a wall} \\ 1 & \text{if } E(x,y,z) \text{ is occupied by a wall} \end{cases}$$

$$E(x,y,1) = \begin{cases} 0 & \text{if } E(x,y,z) \text{ is not the goal location} \\ 1 & \text{if } E(x,y,z) \text{ is the goal location} \end{cases}$$

$$E(x,y,2) = \begin{cases} 0 & \text{if } E(x,y,z) \text{ is not occupied by a piece} \\ 1 & \text{if } E(x,y,z) \text{ is occupied by a piece} \end{cases}$$

In Figure 3.1, the representation of a simple 5×5 environment example is presented. Figure 3.1a shows the visual representation of this configuration in which the grey and green squares represent the walls and goal location respectively, and the smaller yellow squares represent the pieces. Figure 3.1b illustrates the equivalent matrix representation of the environment example.



(a) Visual representation

(b) Matrix representation

Figure 3.1: Environment representation example.

3.2.2 Simulation Mechanisms

Besides the structure and the elements of the environment, there are a set of rules governing the simulations and determining the dynamics of the agent-environment interaction.

First of all, the agent only has four possible actions to perform, namely: move up, move down, move left and move right. Then, as all four possible actions are movements, by performing any of them, the agent will move to the adjacent cell in that direction. The action decision making is handled by the Q-Learning system which will be disclosed in the next section. In terms of the simulation, a step is considered as the choice and performance of any action. However, there are two situations in which the agent does not perform the movement derived from a selected action. When the action selected by the agent leads to a wall or to the limits of the environment (grid), the agent's movement is not applied but it is still considered as a step done.

Regarding the interaction between the agent and the pieces placed around the environment, a piece is considered collected when the agent performs an action leading it to a position where that piece is located. After the agent performs the action, the piece disappears from the environment modifying both its visual and matrix representation.

Finally, a simulation terminates when the agent is able to reach the goal location regardless of the number of pieces collected or when the set limited number of steps have been performed.

3.2.3 Reward System

What makes Reinforcement Learning versatile and powerful is the reward philosophy. Giving the agent positive rewards whenever it performs well and punishing it, giving negative rewards, whenever it does something it is not supposed to. This is the reason why in any RL design or implementation, the reward system design acquires a great relevance and importance.

In our case, the underlying reward function for the system is described in Equation 3.1. There are four clearly distinguishable cases. On the one hand, if the agent reaches the goal location it receives a reward of $100 \cdot N_i$ units where N_i is the number of pieces

collected until the final step i. The aim of the way this reward is computed is to encourage the agent to collect more pieces before moving to the goal location. When the agent collects a piece it receives a reward of 100 units. On the other hand, if the agent selects an action that moves it towards a wall or out of the limits of the environment, it receives a punishment of -10 units.

As we want our agent not only to pick all the pieces and carry them to the goal location but also to do it in the least number of steps possible, every step the agent takes is penalized by a constant value called *energy cost* ($\mathcal{E} = -1$).

$$\mathcal{R}(s,a) = \begin{cases} 100 \cdot N_i + \mathcal{E} & \text{if the agent reaches the goal location} \\ \mathcal{E} - 10 & \text{if the agent moves towards a wall or the grid limits} \\ 100 + \mathcal{E} & \text{if the agent collects a piece} \\ \mathcal{E} & \text{otherwise} \end{cases} \tag{3.1}$$

3.3 Q-Learning Agent

While the environment is a static component with its dynamics previously defined, the only component capable of adapting and learning through simulations is the agent. As discussed in Section 2.2, the dynamics of the agent, as well as its learning process are guided by the Q-Learning algorithm. The two-dimensional grid, the configuration of its elements and the agent are referred to as *world*.

All Reinforcement Learning agents need their own representation of the world so as to know and understand its situation relative to the environment at all times. This own representation is referred to as the agent's state. The agent's state of the firstly designed base agent was defined simply by its coordinate position in the grid. That was the only information, besides the reward, the agent was receiving from the environment. The learning process of this base agent was exactly the same as the one described in Algorithm 1. Therefore, the agent was learning from the information it was experiencing at each step while taking that step. Finally, all the Q-values the Q-function was supposed to provide and which had to be updated in each step were stored in a table.

In order to complement agent's perception of the environment, two improvements were implemented, namely vision and a collected pieces counter. In relation to the learning process, two other improvements were also implemented, experience replay, which allowed the agent to learn from past experiences, and the substitution of the Q-table by an MLP which approximates every Q-value given a certain state. The final agent is the implementation of these four improvements into the base agent.

3.3.1 Vision

Intuitively, giving the agent the ability to know what is it surrounded by, assists it to select better actions and anticipate positive or negatives rewards. Vision is represented by a small agent centered grid. The grid dimension is adjustable. Figure 3.2 shows the 3×3 vision setting.

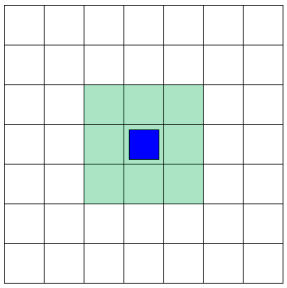
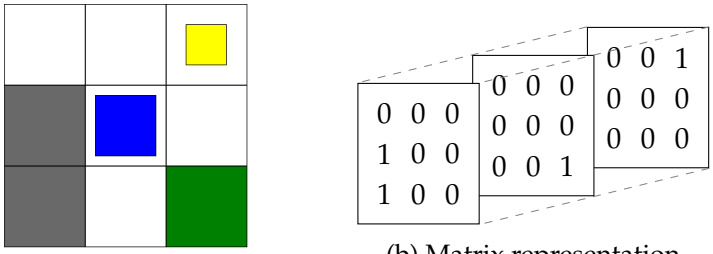


Figure 3.2: 3×3 example vision illustration. The agent is represented by the blue square. The tiles the agent is observing are highlighted with green.

To implement it, the agent receives a three-dimensional matrix representation of this vision. The structure of the vision matrix is the same as the one representing the environment. Figure 3.3 illustrates the visual and matrix representation of a possible vision scenario in which the agent is observing the three types of environment elements.

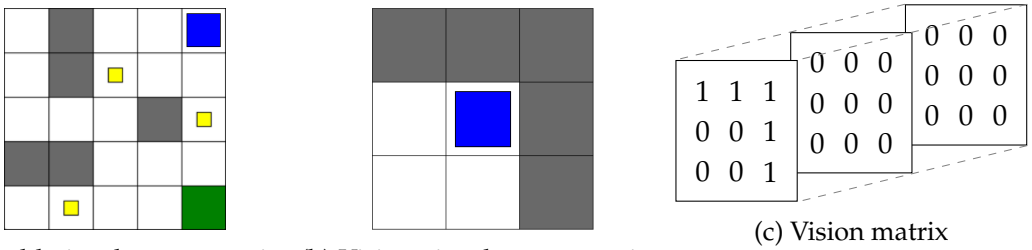


(a) Visual representation

(b) Matrix representation

Figure 3.3: 3×3 Vision representation with the agent surrounded by walls, a piece and the goal location.

The matrix derived from the vision is the information the agent receives from the environment and with which it can analyze the state it is in and which action to take better. However, there are situations where the vision grid is out of bounds. These situations occur when the agent is located in tiles such as the environment grid limits. In these cases the representation of the vision treat the out-bounded tiles as walls. Figure 3.4 exemplifies one of these cases.



(a) World visual representation (b) Vision visual representation

(c) Vision matrix representation

Figure 3.4: 3×3 Vision representation with the agent located in the top right corner.

Figure 3.4a represents an arbitrary world configuration in which the agent is located in the top right tile, Figure 3.4b represents the derived vision and Figure 3.4c describes the derived matrix from the agent's vision.

3.3.2 Piece Collecting

If the agent does not collect any piece before reaching the goal location, the task is evaluated as failed. If the agent collects some of them but not all, it is evaluated as partially performed. This is the reason why improving the agent's ability in terms of counting the number of pieces collected and left to collect is an important issue.

To provide all this information to the agent, a one-dimensional integer array is defined. The characteristics of this array and how it is modified depending on what the agent is experiencing are as follows:

- The length of the list is the total number of pieces placed in the grid.
- Each element of the array represents one specific piece.
- Initially, all the elements are initialized as zeros.
- When the agent collects a piece, the value of the corresponding element is changed to one.
- When the value of an element is equal to one, this means the corresponding piece has already been collected by the agent. Otherwise, the value of the element is equal to zero.
- Every time the simulation starts or the environment is reset, the array is redefined and filled with zeros.
- When all the elements have a value of one, the agent has collected all the pieces.

3.3.3 MLP Structure

Due to the complexity and dimension of the state space, a function approximator is needed to model the Q-function. Furthermore, the use of a function approximator provides the system versatility and robustness in front of state space complexity variations. The function approximator implemented in the Q-Learning system is the MultiLayer Perceptron, which has been broadly described in Subsection 2.2.2.

The structure of the MLP designed for our agent consists of three fully connected layers each of which has 32, 50 and 4 neurons, respectively. The first layer corresponds to the input layer and it has been designed so as to be able to compute all the state information given by the agent. The second layer, also referred to as hidden layer, is designed with the trial and error method until its performance is good enough. The third layer, output layer, is determined by the number of actions the agent is able to perform. The activation functions used by the hidden layer and the output layer are the *sigmoid* activation function $f(x) = \frac{1}{1+e^{-x}}$ and *linear* activation function $f(x) = x$. All the weights are initialized by the *Glorot uniform* initializer also referred to as *Xavier uniform* initializer. The optimizer used to process and apply the changes during each learning cycle is the *Adam* optimizer which is an improved and more complex version of the traditional stochastic gradient descent algorithm. Lastly, the loss function used to compute the error between the value outputted by the MLP and the one derived from the Q-Learning algorithm update is the mean squared error (MSE). Figure 3.5 shows a schematic representation of the described MLP and some of the relevant characteristics mentioned previously.

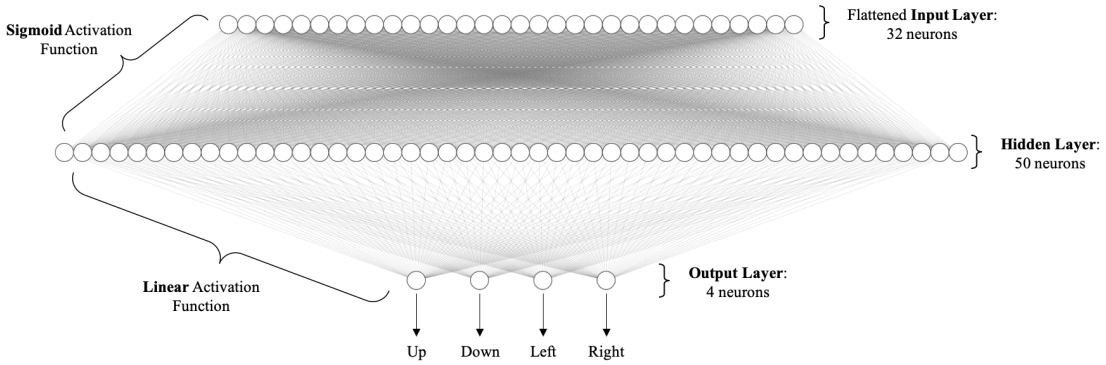


Figure 3.5: MLP structure and description of its parts.

After describing the global structure of the MLP and its characteristics it is necessary to discuss in more detail the specific structure of the input layer and the information it contains as well as the preprocessing needed. Figure 3.6 describes the transformation of the information provided by the agent's experience into the flattened input array given to the MLP.

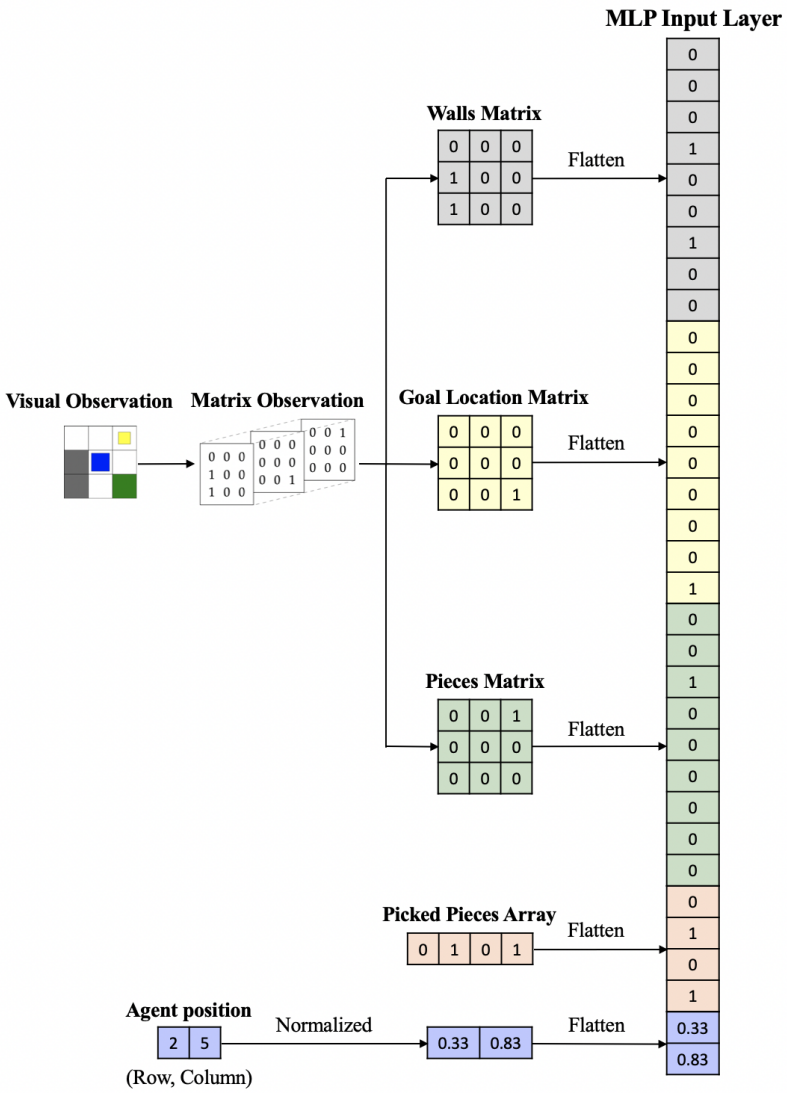


Figure 3.6: MLP input layer design and description.

The experience information is the agent's vision, the array representing which pieces have been collected and the agent's position. Regarding the agent's vision, as discussed in Subsection 3.3.1, it is a three-dimensional array with three layers containing the agent's vision in relation to the walls, goal location, and pieces. Then, these three layers are split into three independent matrices and each matrix is flattened as a vector. The same vectorizing procedure is done with the array of collected pieces. Lastly, the row and column coordinates of the agent's position are normalized with respect to the grid dimensions. The normalization procedure is showed in equation 3.2.

$$x_{norm} = \frac{x}{M-1} \quad y_{norm} = \frac{y}{N-1} \tag{3.2}$$

Where x and y are the row and column coordinates of the agent's position respectively, M is the total number of rows and N is the total number of columns of the environment grid.

The normalization is done because MLPs have a better performance when their inputs are normalized. This array is also vectorized. Concatenating all resulting vectors, the MLP input layer is obtained. All the transformations applied to the agent's information in order to generate the MLP input layer will be referred to as the preprocessing stage.

3.3.4 Experience Replay

In Subsection 2.2.3, experience replay has been discussed. As mentioned there, if the Q-Function is approximated by an MLP and the agent is learning directly from the current data it is experiencing in each step, this data has a considerable temporal correlation. This correlation inhibits the MLP learning process, due to the gradient descent techniques used.

For the sake of the agent's learning process, an experience replay memory was designed. This memory has the following characteristics:

- The memory size is limited to a maximum value in order not to store all the transitions experienced by the agent (10^4 transitions).
- It can sample a batch of stored transitions to provide them to the agent (100 transitions).
- When the memory is full, the oldest transition is substituted by the newest one.
- The memory will only provide a batch of sampled transitions when the maximum size has been reached.

The information stored in the memory as a transition consists of all the information related to the agent's current state and next state. This information is the agent's vision, pieces collected array and agent's position from the current and next state; the action taken to transition from the current state to the next state; the reward obtained due to performing this transition and a value determining if the simulation has terminated or not. Figure 3.7 shows a simple representation of it.

Current Observation	Current Position	Current Pieces Picked Array	Action	Reward	Next Observation	Next Position	Next Pieces Picked Array	Terminated

Figure 3.7: Structure of the transition sets stored in the experience replay memory.

3.3.5 Simulation Cycle

Lastly, the process that the system performs to simulate an episode is described in Algorithm 3. This process will be referred to as the simulation cycle. The simulation cycle is run as many times as the number of episodes set for an experiment.

Algorithm 3 Q-Learning system simulation cycle for one episode

initialize environment E
initialize replay memory $\mathcal{D}$ with capacity $\mathcal{N}$ and minibatch size n
initialize agent $\mathcal{X}$ and its MLP with random weights θ
initialize initial state s, initial vision o and initial pieces array p from environment E

initialize step counter $i = 0$
initialize terminated flag λ = False
initialize step-limit T
initialize ϵ-greedy exploration rate ϵ

while $\lambda \neq$ True **do**
  preprocess s, o and p to create the MLP input layer
  perform a forward propagation with the created input layer and get the Q-values of the agent's current state, set ρ as a random probability $0 \leqslant \rho \leqslant 1$
  if $\rho \leqslant \epsilon$ **then**
    select a random action a
  else
    select action $a = argmax_{a \in \mathcal{A}}\, Q(s, a)$
  end if
  perform action a to get next state s', next vision o', next pieces array p', reward r and terminated flag λ
  store transition (s, o, p, a, r, s', o', p', λ) in $\mathcal{D}$
  if size($\mathcal{D}$) $\geqslant \mathcal{N}$ **then**
    sample random minibatch of transitions (s_j, o_j, p_j, a_j, r_j, s'_j, o'_j, p'_j, λ) from $\mathcal{D}$
    for each transition set **do**
      preprocess s_j, o_j, p_j to create the current state MLP input layer
      preprocess s'_j, o'_j, p'_j to create the next state MLP input layer
      if $\lambda =$ True **then**
        set $y_j = r_j$
      else
        set $y_j = r_j + \gamma \max_{a'} Q(s'_j, a'; \theta)$
      end if
      perform a learning step on $(y_j - Q(s_j, a_j; \theta))^2$
    end for
  end if
  $s \leftarrow s'$
  $o \leftarrow o'$
  $p \leftarrow p'$
  if $i = T$ **then**
    set λ = True
  end if
  set $i = i + 1$
end while

4 Experimental Setup

In this chapter we describe the design and setup of the experiments performed so as to evaluate the method proposed in the last chapter. First of all, the hyperparameters used in the experimentation and their values are determined. Then, we present the different configurations of the method and the environment to compare and analyze its performance such as the use of the agent's vision, the dimension of the environment grid or the number of pieces to be collected.

4.1 Parameter Optimization

The parameter tuning is a transcendental step in all RL problems. All parameters were described in chapter 2. Their optimization is done based on a mix of theoretical knowledge and trial and error. In order to do it, the different possible combinations of these parameters are tested with the same environment configuration and complexity, a 7×7 grid with agent's vision and four pieces to be collected. The visual representation of the mentioned environment is shown in Figure 4.1.

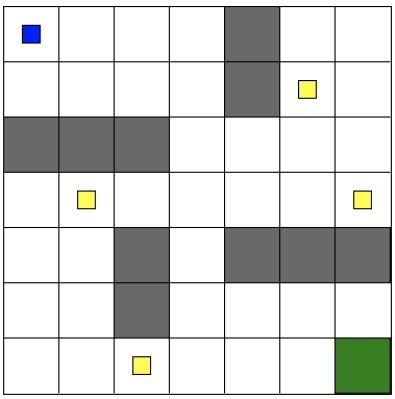


Figure 4.1: Environment configuration used for parameter optimization.

The values tried for each parameter and the reason to select them are the following:

- Learning Rate (α): 0.001, 0.005, 0.01, 0.05
  - If the learning rate is set too low, the learning process would be too slow, but if the value is too high, it can cause a divergent behaviour leading to not finding any solution. As we prefer finding a solution than accelerating the learning process the values tried are relatively low.
- Discount Factor (γ): 0.9, 0.95, 0.975, 0.99
  - Our purpose is to make the agent behave in a way that it will try to obtain the maximum cumulative reward rather than instantaneous rewards. Then, high discount values are tried.
- Initial Epsilon (ϵ): 0.8, 0.9, 1

  - The epsilon value determines how random is the action decision making of the agent. Then, as we want the agent to start exploring the environment instead of following the learned policy, this parameter is tried with high values.

- Epsilon Decay Rate: 0.95, 0.975, 0.99, 0.995, 0.999
  - Similarly to the epsilon value, this parameter affects how much is the epsilon decreased after each episode. In order to provide sufficient exploration and episodes for the agent to learn, the parameter is tried to obtain a slow decrease.

- Epsilon Minimum Value: 0, 0.1, 0.2, 0.3
  - This parameter is more related to the testing part of the experimentation. For this reason, low values are required because we do not want the agent to continue exploring but to follow the policy learned.

The final parameter values are determined based on performance indicators like accuracy, convergence time and final cumulative reward. The values which perform the best are shown in Table 4.1

Learning Rate	Discount Factor	Epsilon	Epsilon Decay Rate	Epsilon Minimum Value
0.005	0.95	1	0.995	0.2

Table 4.1: Chosen parameter values for the final experimentation.

4.2 Method Experimentation and Comparison

In this section we describe the specifications of the experiments done to analyze the performance of the solutions depending on the different configurations and features of the agent and the environment.

4.2.1 Environment Configurations

To compare the agent's performance with regard to the environment complexity we designed four different scenarios with different grid sizes and number of pieces to collect. The four scenarios will be referred to as *Simple with 1 piece*, *Simple with 3 pieces*, *Complex with 2 pieces*, *Complex with 4 pieces*. The simple environment refers to a 5×5 grid and the complex one to a 7×7 grid. All four scenarios are shown in Figure 4.2.

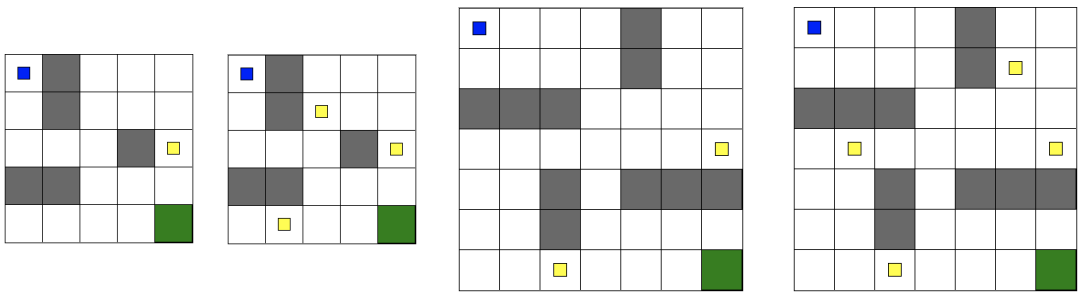


Figure 4.2: Visual representation of the four different environment scenarios used in the final experiments.

4.2.2 Agent's Vision

Another feature added to the agent, and one that requires testing, is the agent's observation or vision. To do it, it is necessary to analyze its influence by testing the agent's performance with and without the vision feature implemented. This comparison is done by running the simulations in all of the previously mentioned environment configurations. A 3×3 vision grid is used, therefore, the agent only knows what is immediately in its surroundings.

4.2.3 Final Experiments Setup

As a summary of what has been presented in this section, the different configurations and setups of the experiments done are shown in Table 4.2. The results of these experiments will be shown and discussed in the next chapter as well as the comparisons between them.

Experiment Number	Environment Size	Number of Pieces	Vision
1	5×5	1	Yes
2	5×5	1	No
3	5×5	3	Yes
4	5×5	3	No
5	7×7	2	Yes
6	7×7	2	No
7	7×7	4	Yes
8	7×7	4	No

Table 4.2: Final experiments setup, where the conditions for each of them are described.

Moreover, all experiments are run under the same conditions and restrictions. First of all it is necessary to understand the terminology used to describe these conditions. Each time the agent performs an action and, therefore, its state changes, will be referred to as a *step* and each time the agent learns from a transition stored in its experience replay memory will be referred to as a *learning step*. Then, a step implies 100 learning steps due to the experience replay mini-batch sampling size. When the agent manages to reach the goal location or exceeds the maximal number of *steps*, will be referred to as having completed an *episode*. Running the number of *episodes* set will be referred to as a *simulation* and running the number of *simulations* set will be referred to as an *experiment*.

Given the definitions of the terminology used to present the results of the experiments, its configurations and restrictions used are set according to Table 4.3.

Simulations per experiment	Episodes per simulation	Max. number of steps per episode	Learning steps per step
10	500	500	100

Table 4.3: Configuration and settings for the final experiments.

5 Results and Discussion

In the previous chapter we defined the experiment configurations and setups as well as each of the scenarios the agent has to solve. The complexities of the environment and the task are the main differences between the scenarios. But, not only the environment and the task are modified between experiments. The agent is also modified between experiments. It is tested having access to the information provided by its vision and not having access to it. In this chapter, we present the results of these experiments and discuss the performance of the agent with respect to each of the variations both in the environment and the task.

The conditions the experiments are governed by were described at the end of the last chapter. Each experiment consists of 10 simulations and a limit of 500 episodes per simulation. These conditions are applied to each of the scenarios. Before running each simulation, the environment is reset to the same initial configuration. The same way, the agent is reset and the weights of the MLP are reinitialized.

The structure of this chapter is the following. Firstly, the results are shown and described, and secondly, we discuss and interpret them. In each of the two parts, the simple environment is treated first and then the complex environment is treated.

5.1 Final Results

Before showing the results of the experiments, the exploratory strategy used requires further description, even if it has already been presented in the previous chapters, so as to understand the evolution of the agent's performance in each of the tested scenarios.

The exploratory strategy used is a slight modification of the *Decaying Epsilon-greedy* strategy which, in turn, is a variation of the traditional *Epsilon Greedy* strategy, as discussed in Section 2.2.4. Moreover, in Section 3.3.4 we discussed the need to fill the experience replay memory of the agent before starting the learning process. Combining this issue with the Decaying Epsilon-greedy method we obtain our own exploratory strategy.

The exploration process is divided into three parts each of which has a specific mission. The first is dedicated to maintain the full-randomness of the action decision making until the experience replay memory is full. We take advantage of that because, then, the first samples the agent learns from are completely random. The second part follows the Decaying Epsilon-greedy strategy according to which the exploratory rate decreases based on the set rate, 0.995. Finally, the epsilon rate decrease ceases at a minimum exploratory set value, 0.2, in order to test the solution found by the agent. All the values mentioned were described in the last chapter. The exploratory rate evolution for all of the experiments is shown in Figure 5.1.

Then, the aim of this section is to show the results of each of the experiments, analyze them and discuss the different behaviours and performances of the agent depending on

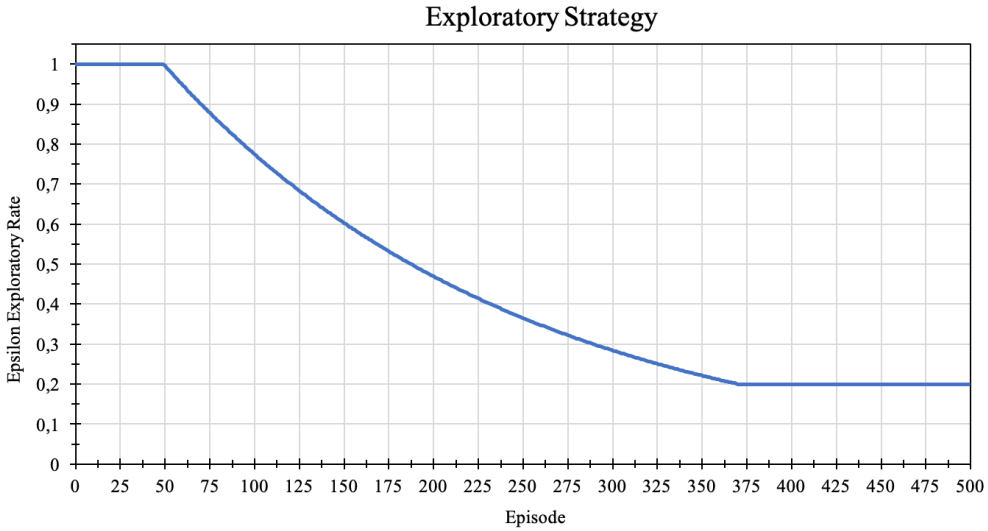


Figure 5.1: Exploratory rate evolution over the 500 episodes of a simulation.

the configuration and the conditions of itself and the environment. The selected indicator to describe the results of the experiments is the mean cumulative reward received by the agent over all the simulations of an experiment. The evolution of this indicator over the episodes of the simulations, as well as its final value, is shown. The average performance time needed by the agent to perform a complete simulation for each scenario is also presented.

5.1.1 Simple Environment

First, we study the results of the Q-Learning system in the simple environment (5×5 grid). The experiments run in this environment consist in collecting 1 or 3 pieces by an agent with and without vision. The evolution of the total cumulative reward received by the agent averaged over the 10 simulations performed in each scenario is presented in Figure 5.2. In addition, the final averaged value is shown in Table 5.1 as well as its standard error and the average time needed by the agent to complete the 500 episodes.

The first noticeable issue is the absence of the results for the scenario *3 Pieces without Vision*. The absence is due to the non-convergence of the agent's total cumulative reward. The reason for this non-convergence is discussed in the next section. In contrast, the other three scenarios are represented in the chart. The difference between the *3 Pieces with Vision* scenario and the other two is caused by the rewards received for collecting two more pieces. To recall the reward system, collecting one piece provides a reward value of 100 and when reaching the goal, the reward received is 100 times the number of pieces collected. An aspect to be highlighted is the difference in performance between the *1 Piece with Vision* and *1 Piece without Vision* scenarios. While the scenario with vision rapidly converges to the final value, the one without it is slower and more unstable. The instability can be seen through the evolution of the mean cumulative reward but mostly from its standard error. In the three scenarios, at the end of the simulations, the agent has learned an optimal policy with which it knows how to collect all the pieces and carry them to the goal location. Based on the results from Table 5.1, the scenario with no vision takes longer to finish the 500 episodes than the one with it.

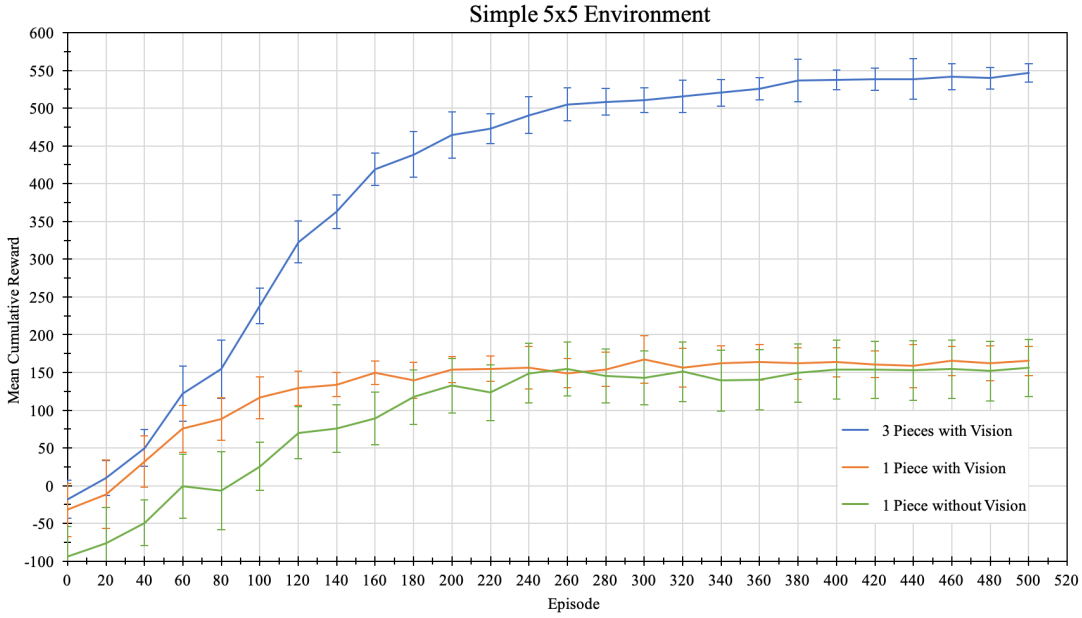


Figure 5.2: Mean cumulative reward evolution for each of the scenarios run in the simple environment over the 500 episodes of the simulations. The bars represent the standard error.

	1 Piece without Vision	1 Piece with Vision	3 Pieces without Vision	3 Pieces with Vision
Mean Cumulative Reward	156.1	165.4	-	546.8
Standard Error	37.9	19.2	-	17.4
Average Performance Time [s]	492	388	-	538

Table 5.1: Mean final cumulative reward, standard error and average performance time of the system in each of the scenarios run in the simple environment.

5.1.2 Complex Environment

The factor with a major effect in the performance of the system is the environment complexity. Therefore, if the agent can learn an optimal policy to collect all the pieces and carry them to the goal location, the experiments will be considered successful. To recall conditions of the experiments performed in the complex environment, the environment dimension is a 7×7 grid with a more complex wall distribution and more pieces to be collected, 2 and 4. Next, the results in these conditions are presented.

Similarly to the last subsection, the evolution of the mean cumulative reward for each scenario is shown in Figure 5.3 and the final values of the experiments are presented in Table 5.2. Analyzing the mean cumulative reward evolution, it can be seen how the agent receives different cumulative rewards in each of the scenarios tested. While the agent is receiving the maximum possible cumulative reward in the scenarios *4 Pieces with Vision* and *2 Pieces with Vision* in an acceptable convergence time, it is not able to reproduce this performance in the *2 Pieces without Vision*. Furthermore, equivalently to the simple environment experiments, the agent is not capable of finding a valid policy in the *4 Pieces without Vision*. The possible reasons causing these issues are discussed in the next section.

The final experiment values illustrate the behaviour differences between the different scenarios. In this case, the standard error and the average performance time from the *2 Pieces without Vision* are again greater than the ones from the *2 Pieces with Vision*.

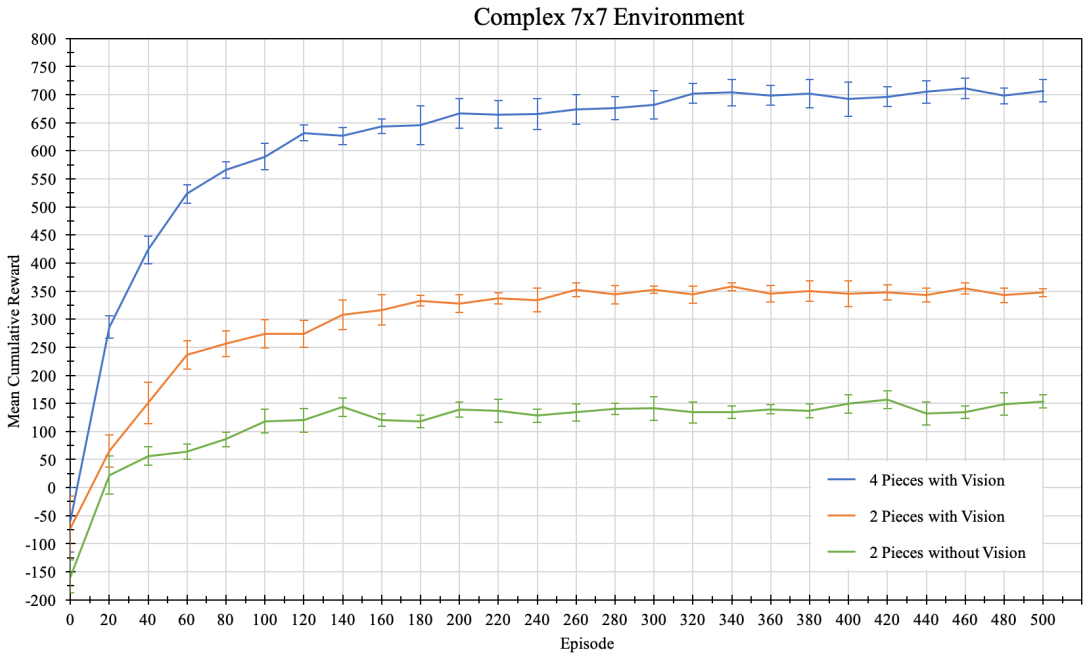


Figure 5.3: Mean cumulative reward evolution for each of the scenarios run in the complex environment over the 500 episodes of the simulations. The bars represent the standard error.

	2 Pieces without Vision	2 Pieces with Vision	4 Pieces without Vision	4 Pieces with Vision
Mean Cumulative Reward	153.5	346.2	-	707.0
Standard Error	12.0	6.9	-	19.5
Average Performance Time [s]	849	751	-	917

Table 5.2: Mean final cumulative reward, standard error and average performance time of the system in each of the scenarios run in the complex environment.

5.2 Results Discussion

The experiments results are discussed in this section. The discussion is based on the comparison between the agent's behaviour depending on the environment complexity and the access to vision. Moreover, other experiments tried did not provide sufficiently acceptable results in order to be shown in the previous section. The referenced experiments were designed to compare the effect of experience replay in the agent's performance. Based on these experiments we verified the agent is not able to find a correct policy due to the correlation between the transitions it is learning and the action decisions generated by the MLP.

5.2.1 Simple Environment

In the *1 Piece without Vision* scenario the final mean cumulative reward indicates the agent has been able to collect the piece and carry it to the goal location. However, the analysis of the standard error of this indicator over the episodes demonstrates the instability of the policy learned by the agent. The standard error is caused by the learning process and the randomness, derived from the exploratory strategy, of the action decision making procedure. But in this case, the standard error is high, compared to the *1 Piece with Vision*, which means the policy learned by the agent is not completely defined because it is not always able to collect the piece. Furthermore, the evolution of the mean

cumulative reward complements this conclusion because its convergence in *1 Piece without Vision* scenario is slower compared to the *1 Piece with Vision*. The difference between the average performance times of both scenarios also confirms this conclusion. All the mentioned deficiencies with regard to the agent in the *1 Piece without Vision* scenario are the reason why it is not capable of finding a policy to follow in the *3 Pieces without Vision* scenario. The agent is not able to find a specific policy because it is prevented by the instability of its learning process owing to not having vision to predict the possible outcome of the action taken.

In the *3 Pieces with Vision* scenario the agent manages to receive the maximum possible cumulative reward which means it has collected the three pieces before getting to the goal location.

5.2.2 Complex Environment

In scenario *2 Pieces without Vision*, similarly to the scenario *1 Pieces without Vision* in the simple environment, the agent manages to find a specific policy. But in this case, the policy found is not optimal due to not collecting the two pieces placed in the environment, therefore the task is partially performed. Increasing the complexity of the environment increases the difficulty for the agent to learn the optimal policy as well. In the complex environment the agent without vision is not able to perform the complex version of the task, collecting four pieces, in the *4 Pieces without Vision* scenario neither.

In the scenarios in which the vision feature is implemented, the agent always learns a policy to collect all the possible pieces. These results verify the adaptability to more complex environments and tasks of the agent with vision. Nonetheless, on the on hand, the convergence towards the maximum reward in the scenario *4 Pieces with Vision* is slow. The reason for this to happen is because the agent rapidly learns it has to collect four pieces to receive the maximum reward, as after 40 episodes the mean cumulative reward is greater than 400, but not the optimal path to follow in order to accomplish it, as until the episode 300 the mean cumulative reward continuously increases. On the other hand, the convergence in the *2 Pieces with Vision* scenario is faster, as the order in which the agent has to collect the pieces is more evident than in scenario *4 Pieces with Vision*.

5.2.3 Final Conclusions

To terminate the analysis and discussion of the results, the most relevant conclusions extracted from them are presented.

First of all, as mentioned in the beginning of this section, adding experience replay to our agent is indispensable in order for it to learn a good behaviour from its own experience. Furthermore, the vision feature clearly assists the agent in its action decision making process. If these two elements are correctly implemented the agent is able to optimally perform in environments and complete tasks with different complexities.

Then, from the point of view of the main aim of this thesis, the results are evaluated as successful.

6 Conclusion and Future Work

In this thesis, we presented the design and implementation of an agent capable of performing a piece collecting task in an environment with obstacles and carrying them to a desired goal location. The base of this process has been the Q-Learning algorithm combined with the approximation of the Q-function through a MultiLayer Perceptron. Moreover, we discussed the need to provide vision to the agent as well as forcing it to learn from past randomly sampled experienced through an experience replay memory.

To evaluate the performance of the designed system, we proposed different scenarios and configurations that allowed us to compare the effect of the factors and features implemented, such as the agent's vision and the environment complexity. The results discussed in the last chapter indicated how important these features are with regard to the agent's capacity to understand what is surrounding it and act accordingly. Even though the experiments performed could only explore some of the power of our system, they showed the great possibilities RL methods and its derived algorithms can offer. In this last chapter we answer the research questions raised at the beginning of this thesis.

Finally, one of the future aims of this thesis is to be able to design a system capable of performing real tasks related to manufacturing or industrial processes. In that sense, the system presented proves it is possible if working more in this direction. This is the reason why we conclude this thesis discussing the possible future improvements applicable to our system to get closer to the mentioned goal.

6.1 Answering Research Questions

Question 1: Is a Reinforcement Learning agent able to learn behaviour policies only by interacting with the environment and show results comparable to those of a programmed industrial robot?

If we treat this question from a theoretical point of view, there is no need to develop an answer to it because it has already been answered in previous approaches [48]. But the real aim of this question is to verify and test the complexity involved in the entire process of designing and implementing an RL system.

Based on the results shown in the last chapter, an RL agent can perform simple tasks similarly to an industrial robot. However, time to train itself and learn how to perform them is required, thus, it can be considered a drawback compared to the existing programmed robots. In contrast, the fact of learning from its own experience provides the RL agent a versatility completely out of reach for industrial robots. It is difficult to ensure an RL agent that can perform a task at the level of a specifically programmed robot for a specific task, but the RL agent would be able to keep performing at a high level. If the task is modified in a way it can still use the experience previously achieved and manage to find new policies able to guide it to find optimal solutions.

Question 2: Despite the relative simplicity of the Q-Learning Algorithm, can this algorithm be used to learn the optimal policy?

In Section 2.2 we disclosed the Q-Learning Algorithm and how it can be used to make the agent learn which behaviour policy provides better results and optimal performance. But this question has the same verification purpose as the first one. Then, the designed system, based on the Q-Learning Algorithm, can learn a specific behaviour policy with which it can select the optimal action so as to reach the maximum cumulative reward. However, implementing the Q-Learning Algorithm, without any other feature or improvement, would not be sufficient for complex tasks in complex environments. For instance, if the state space complexity is too high, a function approximator, e.g. Artificial Neural Networks, providing the Q-values for each of the states the agent can be in is required.

In conclusion, the Q-Learning Algorithm can be used to learn behaviour policies able to guide an agent while performing a task as long as it is combined with other RL concepts, methods and features that complement it and stabilize its training and learning process.

Question 3: What adaptability do Artificial Neural Networks, such as MultiLayer Perceptrons, provide and how do they perform in this type of problems?

The use of ANNs in Reinforcement Learning is one of the keys to the success of this field. ANNs offer almost an infinite amount of possibilities while facing all types of RL problems, but designing and understanding them is still a challenge in some cases.

The preliminary experiments of this thesis were run in an environment without pieces to collect. The only objective was to guide the agent to the goal location while avoiding the walls. The agent's training and learning process was also based on the Q-Learning Algorithm, but instead of implementing a Q-function approximator, a Q-table where the Q-values for each state were stored was used. The agent completed the task, but when the environment got more complex and the piece collecting task was added, the Q-table became a limitation and the computational cost was an insurmountable issue. Then, we implemented one of the simplest ANN structures, the MLP, to overcome the problem. At first it was a heavy workload to adapt the system to the new Q-function approximator and adjust its parameters to obtain the same results as with the Q-table solution. But when the implementation and parameter tuning was completed, the MLP was able to adapt to almost every change in the complexity of the environment.

Hence, ANNs such as MLPs clearly provide the adaptability needed to solve an RL problem. Moreover, this adaptability can be even greater if MLPs are combined with other methods like experience replay, as has been done in this thesis.

Question 4: Which is the minimum information the environment should provide to the agent so as to accomplish the proposed task?

The amount of useful information an agent can obtain from the environment depends on two factors. The first one is the capacity we have to extract the information from the environment in real world applications. The second one is the importance and relevance of the gathered information with regard to the agent's task.

On the one hand, since our system has been developed in a simulated environment, the first factor has no effect because as we design the environment we can add and extract all the necessary information. On the other hand, the second factor does need a rigorous analysis because we only want to provide the essential information to the agent. After analyzing the results of the experiments with and without some of the information we are able to obtain from the environment, the conclusions are clear. The position in reference to the environment grid is required by the agent, therefore, the coordinates of its position are indispensable. In relation to the piece collecting task, it is necessary for the agent to know which pieces have already been collected to not move and try to collect them again. Finally, as discussed in the previous chapter, the discovery of this thesis is the relevance of the agent's vision in order to adapt to more complex tasks and environments. The agent is capable of partially and suboptimally performing in simple tasks but when performing in more complex environments, vision is required so as to predict and anticipate its actions.

6.2 Future Work

The results shown and discussed in the last chapter allowed us to answer all the research questions originally proposed, not only from the research point of view but also from the industrial point of view. In this chapter we provide other suggestions so as to extend the capacities of the proposed system and help to make it more viable in real industrial processes.

6.2.1 Deep Reinforcement Learning

During the research stage of this thesis various RL methods and algorithms were analyzed in order to be implemented and improve the performance of the system. Some of them were implemented and some others were not due to a deficiency of time or to not giving better results. For instance, experience replay and agent's vision were two methods which improved our system while others like adding a *Target Q-Network* [49] did not.

Moreover, a concept being thoroughly studied and used nowadays is *Deep Reinforcement Learning*. It was not used in our system because the complexity of the problem did not require it. But, if we want to get the system closer to solve real world problems, it would be absolutely necessary to work with it. A possible approach would be to use *Deep Q-Networks*, of which its good performance has been sufficiently proven in problems like classic Atari games [44]. This would definitely help to handle environments with a higher complexity and would allow us to design more versatile agents.

6.2.2 Environment Configurations and Dynamic Environment

With the proposed system and sufficient computational power and time, we suggest to increase the complexity of the environment the agent would face by increasing the environment dimensions as well as locating the walls and placing the pieces in a more difficult way such as in a real situation.

In addition, it would be interesting to study how the agent reacts to a dynamic environment in which each time it starts the task the walls and pieces are differently organized or in which the pieces move over time. This last suggestion would need

the use of *Deep RL*, mentioned before, to assist with the agent's learning process but it would definitely be an enormous step towards real world applications.

6.2.3 Implementation into Physical Robot in Real Environment

Finally, as a closing of the entire thesis, we propose to implement the system with the improvements and features previously mentioned in a real robot. To do so, the robot must meet a number of requirements. It must have sensors dedicated to know if it is hitting a wall or an obstacle. It has to be able to collect the pieces it finds in its way. To do this, we suggest two possibilities. The first one is to programmatically collect the object whenever it finds one. The second one is to add a *collect* action to the agent's action space and force the agent to learn it does not only have to get to the pieces but to perform the action of collecting them too. The last requirement is to have sufficient computational power to run and process the system.

References

[1] M. A. Boden, *Artificial Intelligence and Natural Man*, ser. Computer science/psychology. Basic Books, 1977.

[2] M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*, 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[3] J. A. Cruz and D. S. Wishart, "Applications of machine learning in cancer prediction and prognosis", *Cancer informatics*, vol. 2, 2006.

[4] M. Kubat, R. C. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images", *Machine learning*, vol. 30, no. 2-3, pp. 195–215, 1998.

[5] M. S. Bartlett, G. Littlewort, M. Frank, C. Lainscsek, I. Fasel, and J. Movellan, "Recognizing facial expression: machine learning and application to spontaneous behavior", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, IEEE, vol. 2, 2005, pp. 568–573.

[6] J. Peters, "Machine learning for motor skills in robotics", *KI-Künstliche Intelligenz*, vol. 2008, no. 4, pp. 41–43, 2008.

[7] B. G. Buchanan, "A (very) brief history of artificial intelligence", *AI Magazine*, vol. 26, no. 4, p. 53, 2005.

[8] J. McCarthy, "Programs with Common Sense", in *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, London: Her Majesty's Stationary Office, 1959, pp. 75–91.

[9] E. Shortliffe, "Computer-based medical consultations: MYCIN", *Artificial Intelligence - AI*, vol. 388, 1976.

[10] R. Duda, J. Gaschnig, and P. Hart, "Model design in the PROSPECTOR consultant system for mineral exploration", in *Readings in Artificial Intelligence*, Elsevier, 1981, pp. 334–348.

[11] B. Buchanan, G. Sutherland, and E. A. Feigenbaum, *Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry*. Stanford University Stanford, 1968.

[12] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[14] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st. Cambridge, MA, USA: MIT Press, 1998.

[15] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, "Autonomous Helicopter Flight via Reinforcement Learning", in *Advances in Neural Information Processing Systems 16*, S Thrun, L. K. Saul, and B Schölkopf, Eds., MIT Press, 2004, pp. 799–806.

[16] S. Wang, J Braaksma, R. Babuska, and D Hobbelen, "Reinforcement Learning Control for Biped Robot Walking on Uneven Surfaces", in *IEEE International Conference on Neural Networks - Conference Proceedings*, 2006, pp. 4173–4178.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search", *Nature*, vol. 529, p. 484, 2016.

[18] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[19] R. Bellman, "A Markovian Decision Process", *Journal of Mathematics and Mechanics*, 1957.

[20] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[21] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[22] C. J.C. H. Watkins, "Learning from Delayed Rewards", PhD thesis, King's College, Cambridge, UK, 1989.

[23] A. O. Wiehe, N. S. Ansó, M. M. Drugan, and M. A. Wiering, *Sampled Policy Gradient for Learning to Play the Game Agar.io*, 2018. arXiv: `1809.05763 [cs.AI]`.

[24] C. J.C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[25] J. N. Tsitsiklis, "Asynchronous Stochastic Approximation and Q-Learning", *Machine Learning*, vol. 16, no. 3, pp. 185–202, 1994.

[26] T. Jaakkola and M. I. Jordan, "On the Convergence of Stochastic Iterative Dynamic Programming Algorithms", *Neural computation*, vol. 6, pp. 1185–1201, 1994.

[27] F. S. Melo, "Convergence of Q-learning: A simple proof", *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2007.

[28] S. Yang, T. O. Ting, K. L. Man, and S.-U. Guan, "Investigation of Neural Networks for Function Approximation", *Procedia Computer Science*, vol. 17, pp. 586–594, 2013.

[29] C Zhang, L Yu, and W Wu, "Study on fitness data processing based on neural network information processing", in *2012 International Conference on Systems and Informatics (ICSAI2012)*, 2012, pp. 295–298.

[30] Ying Zhao, Jun Gao, and Xuezhi Yang, "A survey of neural network ensembles", in *2005 International Conference on Neural Networks and Brain*, vol. 1, 2005, pp. 438–442.

[31] F Heimes and B van Heuvelen, "The normalized radial basis function neural network", in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, vol. 2, 1998, 1609–1614 vol.2.

[32] J. Dr, A. Deb, and S. Chandra, "Algorithm for building a neural network for function approximation", *Circuits, Devices and Systems, IEE Proceedings -*, vol. 149, pp. 301–307, 2002.

[33] R Setiono and A. Gaweda, "Neural network pruning for function approximation", vol. 6, 2000, 443 –448 vol.6.

[34] J. Bapu, *The Artificial Neural Networks Handbook: Part 4*, 2018. [Online]. Available: `https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e` (visited on Apr. 25, 2019).

[35] J. Bapu, *The Artificial Neural Networks Handbook: Part 1*, 2018. [Online]. Available: `https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4` (visited on Apr. 25, 2019).

[36] G Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[37] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[38] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning", no. 9, 2015.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1", in, D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., Cambridge, MA, USA: MIT Press, 1986, ch. Learning I, pp. 318–362.

[40] D Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*, 1994.

[41] M Riedmiller, "Concepts and Facilities of a Neural Reinforcement Learning Control Architecture for Technical Process Control", *Neural Computing & Applications*, vol. 8, no. 4, pp. 323–338, 1999.

[42] M. Riedmiller, "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method", in *Machine Learning: ECML 2005*, J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328.

[43] L.-J. Lin, "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching", *Mach. Learn.*, vol. 8, no. 3-4, pp. 293–321, 1992.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning.", *Nature*, vol. 518, no. 7540, pp. 529–33, 2015.

[45] M. Wiering, "Explorations in Efficient Reinforcement Learning", PhD thesis, University of Amsterdam, 1999.

[46] A Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for Q-learning in random stochastic mazes", in *2016 IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2016) Proceedings, 6-9 December 2016, Athens, Greece*, United States: Institute of Electrical and Electronics Engineers (IEEE), 2017.

[47] H. E. Alagha, *Communicating Intention in Decentralized Multi-Agent Multi-Objective Reinforcement Learning*, Master's thesis, University of Groningen, 2019.

[48] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey", *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[49] H. van Hasselt, A. Guez, and D. Silver, *Deep Reinforcement Learning with Double Q-learning*, 2015. arXiv: `1509.06461 [cs.LG]`.