ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

INGENIERÍA EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

DESIGN OF A MULTI-AGENT ARCHITECTURE USING DEEP REINFORCEMENT LEARNING TO DECREASE THE NUMBER OF INTERACTIONS BETWEEN AGENTS AND ENVIRONMENT BY COMMUNICATING KNOWLEDGE

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE MÁSTER EN COMPUTACIÓN

DAVID ALEXANDER CÁRDENAS GUILCAPI

david.cardenas@epn.edu.ec

Director: Henry Patricio Paz Arias

henry.paz@epn.edu.ec

Enero, 2023

AVAL

Como director del trabajo de titulación “Design of a multi-agent architecture using deep reinforcement learning to decrease the number of interactions between agents and environment by communicating knowledge” desarrollado por David Alexander Cárdenas Guilcapi, estudiante de la carrera de maestría en computación de la facultad de ingeniería de sistemas, habiendo supervisado la realización de este trabajo y realizado las correcciones correspondientes, doy por aprobada la redacción final del documento escrito para que prosiga con los trámites correspondientes a la sustentación de la Defensa oral.

HENRY PATRICIO PAZ ARIAS

Firmado digitalmente por HENRY PATRICIO PAZ ARIAS
Fecha: 2023.04.04 13:45:18 -05'00'

Henry Patricio Paz Arias

DIRECTOR

DECLARACIÓN DE AUTORÍA

Yo, David Alexander Cárdenas Guilcapi, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Firmado electrónicamente por:
DAVID ALEXANDER
CARDENAS GUILCAPI

David Alexander Cárdenas Guilcapi

DEDICATORIA

Dedico este trabajo a mi sobrina Romina. Te adoro, pequeña.

AGRADECIMIENTO

Quiero agradecer a mi familia por su apoyo durante mi carrera estudiantil y profesional, han sido años difíciles y no lo habría logrado sin ellos, especialmente sin la presencia de Romina. También agradecer a mis amigos que me demostraron su apoyo y siempre preguntaban cómo iba todo. También a mi tutor, por su acompañamiento, motivación e inspiración desde que fui su estudiante de la materia de Inteligencia Artificial.

TABLE OF CONTENT

RESUMEN

El presente proyecto de investigación introduce una integración de tres técnicas de inteligencia artificial. El propósito es crear un sistema de comunicación en donde los agentes puedan aprender del ambiente utilizando un algoritmo de aprendizaje profundo y comunicar el conocimiento obtenido. Mediante la comunicación, los agentes pueden tomar ventaja del conocimiento de sus semejantes de manera que el número de acciones incorrectas tomadas en el ambiente se reducen. Aparte del uso de una arquitectura multi agente con roles bien definidos, algunos ajustes se realizaron en el algoritmo *Deep Q Learning*. Uno de ellos es la adición de información a las observaciones almacenadas en la *experience replay*; esta información adicional es una bandera que permite que el agente reconozca un estado relevante de manera que el valor de las recompensas pueda ajustarse durante el entrenamiento de la red neuronal. Otro ajuste es el uso limitado de *ε-greedy* el cual previene que un agente explore un estado que ya ha sido comunicado por sus similares, ya que los estados que son comunicados representan una observación en donde se cometió un error. Estos ajustes demostraron ser efectivos ya que el agente reduce el número de episodios donde se cometen errores en un 86%.

Palabras clave: Aprendizaje por refuerzo profundo, multi agentes.

ABSTRACT

This research project introduces an integration of three artificial intelligence techniques. The purpose of this integration is to create a communication system where agents can learn from the environment using a deep learning algorithm and communicate the knowledge obtained. Through communication, agents can take advantage of the knowledge of its peers in a way that the number of incorrect actions taken in the environment is reduced. Apart from the use of a pair-based multi-agent architecture with well-defined roles, some adjustments were performed in the deep q learning algorithm. One of them is the addition of information to the observations stored in the experience replay; this additional information is a flag which allows the agent to recognize a relevant state so the value of the rewards can be adjusted during the training of the network. In addition, the use of epsilon-greedy is limited to prevent an agent from exploring states that have already been explored and reported by other agents. These states have been identified as having errors and are therefore not worthy of further exploration. These adjustments proved to be effective since the agent reduces the number of episodes where errors are made in around 86%.

Keywords: Deep reinforcement learning, multi-agents.

1. INTRODUCTION

This work presents a combination of multi-agent systems and deep reinforcement learning. A multi-agent system is made of multiple interactive computing elements known as agents. An agent is a computing system with the capacity of taking actions autonomously and interact with other agents [1].

Deep reinforcement learning involves the use of deep learning in the reinforcement learning framework. Deep learning marks a difference with “shallow” learning in the sense that there is one or more hidden layers between input and output layers [2]. For this work, the input is an image which represents a situation an agent inside an environment and the output is the action that the agent should take in that situation. Since the input is an image, a convolutional neural network is used which has a better generalizing ability as compared to other neural networks [3].

The way the agent learns is through reinforcement learning, which has gained significant attention from researchers across multiple fields, including psychology, control theory, artificial intelligence and neuroscience [4]. Reinforcement learning is defined as an artificial intelligence paradigm where an agent learns via interaction where the goal is to maximize a reward signal.

A combination of reinforcement learning and multi-agent systems is seen in [5], where multiple agents try to learn an optimal behavior in a fewer number of episodes compared to a one agent acting by itself. In that work, a multi-agent architecture was designed and it was tested on two environments with different dynamics proving that agents can motivate or discard relevant actions in order to reach their goal. In this multi-agent architecture, pairs of agents with specific roles exchange information. The information within the system consists of relevant states represented as numerical values indicating positions and distances. This representation of a state proved to be effective, however an increase in the number of states makes it more difficult to assimilate the information.

The problem of requiring a higher abstraction level originated the idea of implementing a neural network, specifically, a convolutional neural network to take advantage of its generalizing ability. This implementation will provide the multi-agent architecture with the capacity of taking as input the raw pixels that represent a situation between the agent and its environment. Adding a neural network involves the use of Deep Q-learning. Also, involves a change in the information exchanged through the system which are no longer states in the form of positions and distances but tuples. These tuples contain the information needed by a deep q learning algorithm, as well as an extra element that enables the identification of relevant states.

This work has also fixed an issue regarding the maximum reward obtained in one of the environments which was reported as rebound effect [5]. This observed effect showed that the agent has changed its behavior to achieve the goal of making fewer mistakes, however, the reward value obtained was lower than that of a single agent operating independently. In this work, a special focus is done in this effect with the goal of eliminating it and allowing the agents not only to make no mistakes but also not harm the value of the reward obtained.

Regarding the architecture, some aspects of the multi-agent architecture are kept, like the pair-based structure and the restriction of the use of epsilon-greedy. Other aspects are changed like the way the agents communicate and the information that the agents transmit. On the other hand, the integration of a deep neural network involved a new way of taking advantage of the information transmitted.

The second section of this work presents the theoretical framework that it is based on. The methodology is introduced in the third section, followed by the results and discussion in the fourth section. The fifth section presents the conclusions and future work, and the bibliography can be found in the final, sixth section.

1.1 Research Question

¿How can a pair-based multi-agent architecture work along with deep reinforcement learning to reduce the number of episodes necessary to make no errors without harming the optimal reward?

1.2 General Objective

Design a multi-agent architecture using deep reinforcement learning to decrease the number of interactions between agents and environment by communicating knowledge.

1.3 Specific Objectives

- Design a pair-based multi-agent architecture where all the agents can communicate with each other.
- Design and implement a deep reinforcement learning model using deep q learning algorithm where exploration is the main task.
- Integrate the deep reinforcement learning model with the multi-agent architecture so agents can learn to make no errors in less interactions with the environment.

1.4 Hypothesis

Deep reinforcement learning can be integrated into a pair-based multi-agent architecture to reduce the number of agent-environment interactions without harming the convergence of the results.

2. THEORETICAL FRAMEWORK

The current section introduces the theoretical background this project relies upon.

2.1 Reinforcement Learning

Reinforcement learning (RL) refers simultaneously to a problem, a class of solutions methods that work well on the class of problems and the field that studies these problems and their solution methods [4]. RL involves how to map situations into actions to maximize a numerical reward signal without being told which actions to take, through trial and error. It attempts to mimic the way humans learn new things not from a teacher but from interaction with the environment [6].

The use of reinforcement learning is wide, since it's been involved in areas like engineering, neuroscience, psychology, mathematics and economics. On the practice RL-systems have been able to make humanoid robots walk, defeat the world champion at Go, and play many different Atari games better than humans, among others. In all the systems where RL is involved, the basic idea is capturing the most important aspects of the problem to provide an agent with the capability of interact with its environment and achieve a goal.

In the reinforcement learning problem, the learner is called the *agent* and everything outside the agent, is called the *environment.* These two interact continually, the agent selects actions, and the environment responds to those actions and presents new situations to the agent. In a more specific way, agent and environment interact in a sequence of discrete time steps, $t =$

$0,1,2,3$. At each time step t , the agent receives some representation of the environment's state $S_t \in S$ where S is the set of possible states, and based on that, selects an action $A_t \in A(S_t)$ where $A(S_t)$ is the set of actions available in state S_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward R_{t+1} and finds itself in a new state S_{t+1}. This process is illustrated in Figure 1.

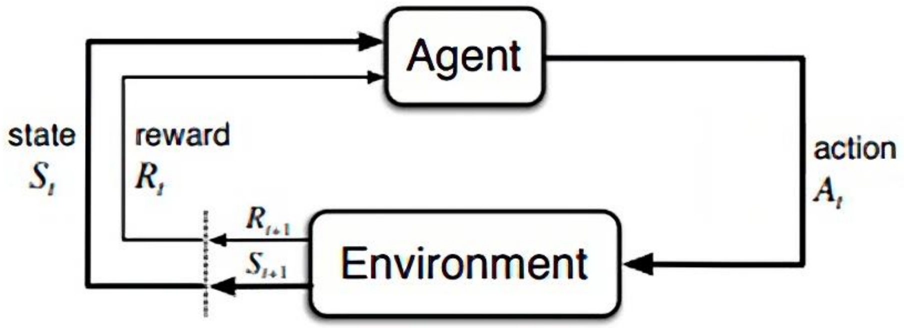


Figure 1 Agent-Environment interaction

A state capable of retaining all the relevant information is said to have the Markov property. To define this Markov property, it is necessary to assume that there are a finite number of states and reward values. A response of the environment at time $t+1$ to the action at time t may depend on everything on everything that has happened earlier. For that case, the dynamics of the environment can be defined as in Equation 1 [2].

$$Pr\,\{S_t = s',\, R_t = r | S_0,\, A_0, R_1, \dots S_{t-1},\, A_{t-1}, R_{t-1}\} \qquad (1)$$

Equation 1 Dynamics of an environment

For all s', r and all possible values of $S_0,\, A_0, R_1, \dots S_{t-1},\, A_{t-1}, R_{t-1}$.

When the environment's response at t depends only on the state and action at $t-1$, the state has the Markov property and the dynamics of the environment can be defined as in Equation 2.

$$p(s',\, r\,|s,a) \doteq Pr\,\{S_t = s',\, R_t = r|S_{t-1} = s,\, A_{t-1} = a\} \qquad (1)$$

Equation 2 Dynamics of an environment that meets the Markov property

For all s', $s \in S$, $r \in R$ and $a \in A(s)$. The function p defines the dynamics of the MDP and Pr represents a discrete probability distribution. When Equation 1 is equal to Equation 2 for all s', r and $S_0,\, A_0, R_1, \dots S_{t-1},\, A_{t-1}, R_{t-1}$, the environment and task are also said to have the Markov property. In a situation where an environment has the Markov property, its one-step dynamics allow to predict the next state and expected next reward given the current state and action. As a whole, the one-step dynamics, state and actions of an environment defines a finite Markov Decision Process (MDP) as long as the learning task satisfies the Markov property.

Given the dynamics specified in Equation 2, and assuming an environment as a finite MDP, anything else can be computed about the environment. Equation 3 shows the state-transition probabilities which are obtained by computing the probability of transitioning from one state to another given a particular action.

$$p(s'|s,a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s,\; A_{t-1} = a\} \qquad (3)$$

Equation 3 State-transition probabilities

Equation 4 shows the formula for the expected rewards for state-action pairs which is obtained by evaluating the expected reward of a state-action pair. This expected reward is the weighted average of the expected rewards for all possible next states, being the weights, the transition probabilities from one state to the next one under certain action.

$$r(s,\, a) \doteq E[R_t | S_{t-1} = s,\, A_{t-1} = a] \qquad (4)$$

Equation 4 Expected rewards for state-action-pairs

Equation 5 shows the expected rewards for state-action-next-state triples. It can be obtained by summing all possible next states weighted by the probability of transitioning to each next state and then multiplying each term by the reward associated in the triplet.

$$r(s,\, a,\, s') \doteq E[R_t \mid S_{t-1} = s,\, A_{t-1} = a,\, S_t = s'] \qquad (5)$$

Equation 5 Expected rewards for state–action–next-state triples

An agent receives a reward at each time step, which is a simple number. Informally, the agent's goal is to maximize the total reward it receives, which means maximizing not the immediate reward but cumulative reward in the long run which is known as the return. Formally, defined in Equation 6.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} \dots + R_T \qquad (6)$$

Equation 6 Expected return

T represents the final step, a concept that makes sense in applications where there is a natural notion of final time step, which means that the agent-environment interaction breaks naturally in subsequences called episodes. Each episode ends in a particular state called the terminal state followed by a reset to a pre-defined starting state. One more important concept related to the return is the discount rate, which is represented by γ, a parameter with values $0 \le \gamma \le 1$. This discount rate determines the present value of future rewards. A reward received k time steps in the future, is worth only γ^{k-1} times what it would be worth if it were received immediately. Through this concept of discounting, an agent tries to select actions in a way that the sum of the discounted rewards it receives over the future is maximized which means that it will select A_t to maximize the expected discounted return, represented by the following equation:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad (7)$$

Equation 7 Discounted return

To map a state to a probability of selection certain action at each time step, the agent has a policy, denoted π where $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. These time steps not necessarily refer to fixed intervals of time, they can refer to arbitrary successive stages of decision-making and acting.

Reinforcement learning algorithms involve estimating value functions of states, which estimate how good the expected return will be for the agent in a given state. In other words, this value function estimates how good a state is in terms of its potential for long-term rewards.

The value function of a state s under a policy π denoted $v_\pi(s)$ is the expected return when starting in s and following π thereafter, Equation 8 defines the value function formally.

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \big| S_{t=s}], for\ all\ s \in S \qquad (8)$$

Equation 8 State-value function for policy π

In a similar way, the value of selecting action a in state s under a policy π, $q_\pi(s, a)$, which is the expected return starting from s, taking action a and thereafter following policy π, is called the action value function for policy π and is illustrated in Equation 9.

$$q_\pi(s, a) \doteq E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_{t=s}, A_{t=a}] \quad (9)$$

Equation 9 Action-value function for policy π

To solve a reinforcement learning task is necessary to find a policy that achieves the maximum reward possible over the long run which means that an optimal policy must be found. A policy π is defined to be better than or equal to policy π' if its expected return is greater than or equal to that of π' for all states; $\pi \geq \pi'$ if and only $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \epsilon S$. There is always at least one policy that is better than the others and there are even cases where there is more than one optimal policy and all of them are denoted by π_* [2]. Optimal policies share the same state-value function, which is called the optimal state-value function, and the same action-value function which is called optimal action-value function, both are illustrated in Equations 10 and 11 respectively.

$$v_*(s) \doteq \max v_\pi(s) \quad (10)$$

Equation 10 Optimal state-value function

$$q_*(s, a) \doteq \max q_\pi(s, a) \quad (11)$$

Equation 11 Optimal action-value function

2.2 Q Learning

Q Learning is a Temporal Difference (TD) algorithm, TD is an approach to reinforcement learning where the agent incrementally updates estimates of a value function using observed rewards and the previous estimates of that value function. Q Learning estimates the optimal Q-function by iteratively updating its estimates Q after each interaction with the environment. Q learning performs off-policy learning; meaning that it learns about a different policy than the one generating the interactions with the environment, which is called the behavior policy. The policy the algorithm ends up learning is the optimal policy.

2.3 Epsilon-Greedy

One specific challenge in reinforcement learning is the trade-off between exploration and exploitation [4]. The agent must exploit what it already knows to obtain a reward, but it also must explore in order to make better action selections in the future. One strategy to deal with this dilemma is called epsilon-greedy. With this strategy, the agent takes a random action with probability ε and it takes the best action known up to that time step with probability $1\text{-}\varepsilon$. This process is described in Equation 12.

$$\pi(s) = \begin{cases} a\ random\ action\ if\ epsilon < \varepsilon \\ \arg max_a Q(s, a)\ otherwise \end{cases} \quad (12)$$

Equation 12 Epsilon-greedy [7]

One important aspect is that exploration must be prioritized when the agent does not have enough information about the environment and once the agent has enough information, it must exploit its knowledge [7]. Thus, a variation of epsilon-greedy appears, where a decay is introduced, which means that the value of epsilon will decay across the life of an agent. Specifically, a real value less than 1 is multiplied by epsilon in every episode, which is also known as exponential decay.

2.4 Deep Learning

In the context of RL, the field of deep learning is about approximating functions in high-dimensional problems where tabular methods cannot find exact solutions [8]. Deep learning uses deep neural networks to find approximations for large, complex high-dimensional environments, such as in images and speech recognition. Deep neural networks consist of many layers of neurons where different types of connections are used.

2.5 Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of artificial network which has deep feed-forward architecture which can learn highly abstracted features of objects especially spatial data and can identify them more efficiently [3]. A deep CNN model consists of a finite set of processing layers that can learn various features of input data with multiple levels of abstraction. The first layers learn and extract the low-level features and the deeper layers learn and extract the high-level features.

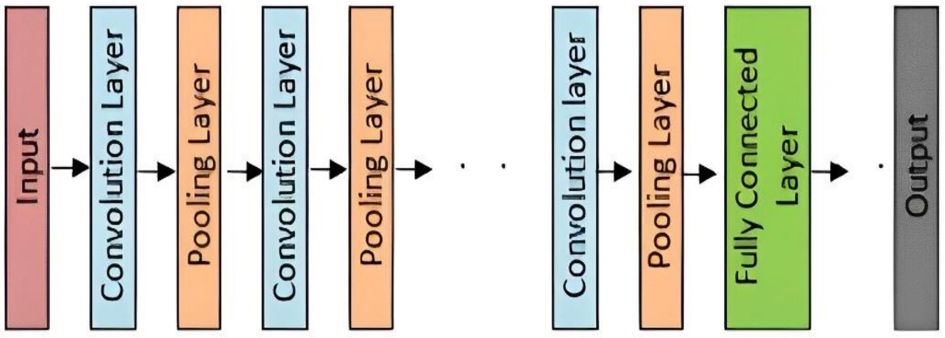


Figure 2 Conceptual model of a CNN [10]

A Convolutional layer is the most important layer of any CNN architecture. In this layer, a set of convolutional kernels can be found, these kernels are also called filters that are convolved with the input to generate an output feature map. A kernel is a grid of discrete values, where each value is known as the weight of the kernel. When the training starts, the values of the kernel are random numbers, once the epochs make the agent learn, the values of the kernel change and is capable of extracting meaningful features. The output feature map is obtained through a convolution operation between the input and the filter (Figure 3). Pooling layers are used to sample the feature map. There are different pooling techniques, but the most popular and mostly used is Max Pooling whose principle is illustrated in Figure 4.

Another important concept is the activation function, which decides whether a neuron will fire or not for a given input by producing the corresponding output. In CNN architecture, non-linear activations layers are used after each learnable layer.

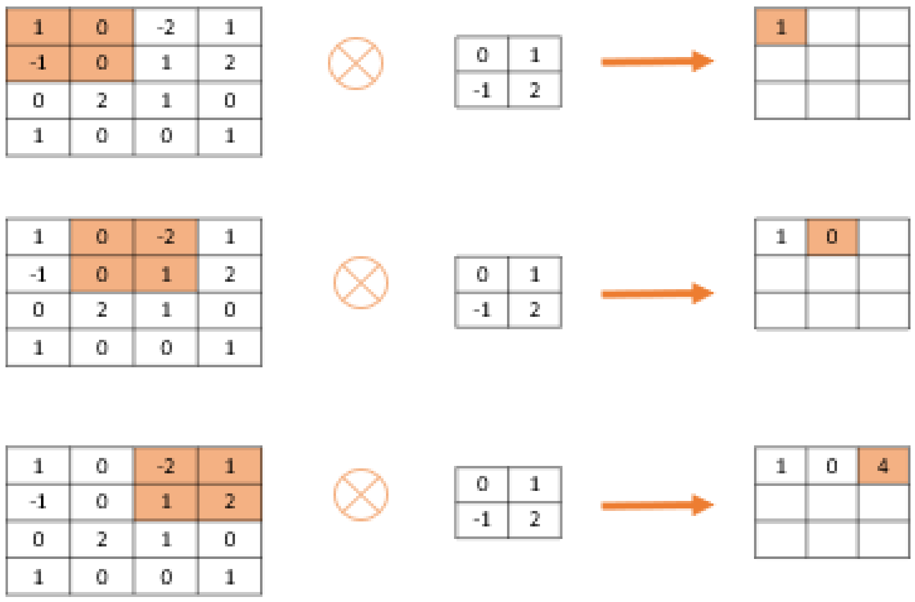


Figure 3 Convolutional process illustrated

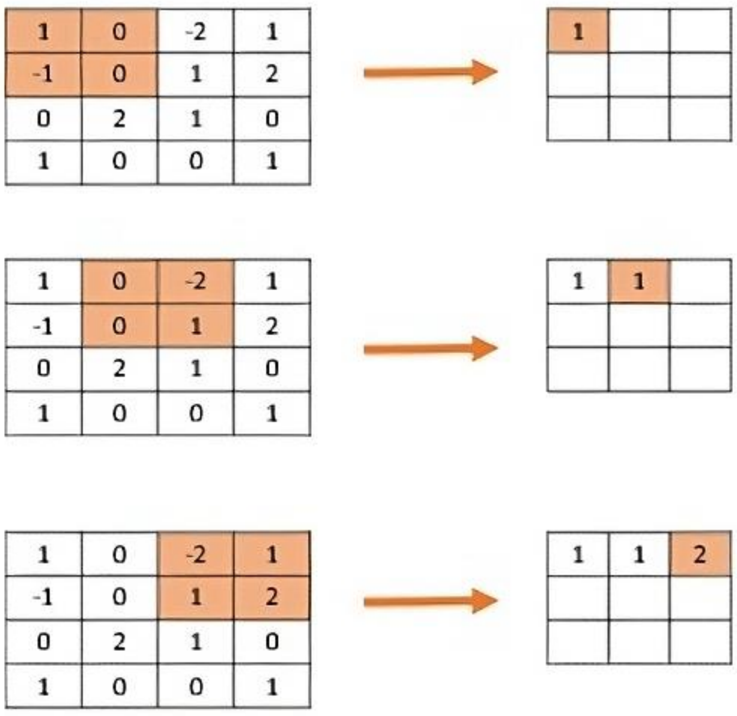


Figure 4 Max pooling process illustrated

2.6 Deep Reinforcement Learning.

Deep Learning has had a significant impact on many areas in machine learning since it can automatically find compact low-dimensional representations of high dimensional data [9]. The progress in RL has similarly been accelerated by DL; the use of DL algorithms within RL defines the field of deep reinforcement learning (DRL).

Deep reinforcement learning can learn to solve large and complex decision problems where there is no solution yet, but an approximating trial-and-error mechanism exists through which a solution can be obtained out of repeated interactions with the problem.

2.7 Deep Q Learning

Deep Q Learning was introduced in the work of Mnih, where a reinforcement learning algorithm is connected to a deep neural network which works directly on RGB images and efficiently process training data using stochastic gradient updates [10], the DQN algorithm is shown in Figure 5. Basically, a network takes preprocessed pixel images as inputs and outputs a vector containing Q-values for each valid action.

The main features of DQN are the use of a target network and the use of experience replay. The target network is a separate neural network used to generate the Q-values and it has the same structure as the online network. Its weights remain fixed for a certain number of time steps until they are updated to match the weights of the online network. The online network is the primary neural network which is used for action selection and updates its weights at each time step [10]. An experience replay [12] was used since learning directly from consecutive samples is inefficient due to strong correlations between the samples.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode = 1, $M$ do
        Initialize state $s_t$
        for t = 1, $T$ do
                With probability $\varepsilon$ select a random action $a_t$
                Otherwise select $a_t = max_a Q^*(s_t, a;\ \theta)$
                Execute action $a_t$ and observe reward $r_t$ and state $s_{t+1}$
                Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
                Set $s_{t+1} = s_t$
                Sample random minibatch of transitions $(s_t, a_t, r_t, s_{t+1})$ from $D$
                Set $y_j = \begin{cases} r_j & for\ terminal\ s_{t+1} \\ r_j +\ \gamma maxa'Q(s_{t+1}, a';\ \theta) & for\ non-terminal\ s_{t+1} \end{cases}$
                Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^{\ 2}$
        end for
end for
```

Figure 5 Deep Q Learning algorithm [11]

2.8 Multi-agent systems

There is no universally accepted definition of the term agent [1].The only general consensus is that autonomy is central to the notion of agent. A definition provided by Wooldridge and Jennings say: An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. Some ideas in common with the definition of Russell and Norvig [13] define an agent as a flexible autonomous entity capable of perceiving the environment through the sensors connected to it and act on the environment through actuators.

Some features of an agent are situatedness, autonomy, inferential capability, responsiveness, pro-activeness and social behavior. The last one stablishes that even though an agent's decision must be free from external intervention, it must still be able to interact with the external sources when the need arises to achieve a specific goal. An agent should be able to share its knowledge and help other agents to solve a specific problem. Agents should be able to learn from the experience of other communicating entities that can be humans or other agents in the network. This last idea brings to the table the definition of a Multi-Agent System (MAS).

In a Multi-Agent system, a group of loosely connected autonomous agents act in an environment to achieve a common goal. This goal is achieved by cooperating, competing, sharing, or not sharing knowledge with each other. MAS have been adopted in many application domains because of the beneficial advantages offered that include the following ideas [14]:

- Increase of the speed due to parallel computation and asynchronous operation.
- When one or more agents fail, the system does not suffer of a representative degradation which means that reliability and robustness are increased.
- Reduced cost since individual agents cost much less than a centralized architecture.
- Reusability, a direct effect of an agent modular structure which means that an agent can be replaced by another or moved to a different system.

3. METHODOLOGY

The following subsections describe the way this project was build; it was divided into three specific parts. The first one focuses on the reinforcement model, the second one on the multi-agent architecture and the third and last one, on how to combine the former parts.

3.1 Reinforcement learning model design

Deep Q learning was the algorithm selected, not only by the off-policy characteristic but also for its use in different areas with good results. Epsilon-greedy with exponential epsilon decay was chosen as the behavior policy, this decay was stablished in 0.9998 and the starting value for epsilon was 1. There is not a minimum value for epsilon, which means that after certain numbers of epochs the value will be 0, meaning that the agent will fully exploit its knowledge.

The environment where the agents were tested is inspired by the work of [5] where the agents are placed inside a delimited area and must move to the furthest point and will be penalized if that point is crossed. The environment in the form of a square presents a challenge for the agent to search for the nearest border starting at a random initial point. This objective of the agent is considered an exploration task, where it tries to find the optimal route to each border. This task is suitable to be divided among multiple agents where each agent has the responsibility of finding the optimal route to one or more of the borders.

For this work, this area is a grid with 21 units per side, agents appear in a central zone and try to move to the outside of the grid. There is an internal border placed 2 units from the border to the inside, this internal border represents a set of furthest points the agent is allowed to step on. A graphic of the environment is showed in Figure 6 and its characteristics are described in Table 1 following the properties described by Wooldridge [1].

The border in color red represents the internal border. When an epoch started, the agent could start its interaction with the environment at any point in the blue zone. The representation of a point is determined by its x and y positions having as the origin the top left corner, the direction of x and y is also shown in Figure 6.

To move from their origin position, agents have 9 actions available, which include 8 movements and a ninth action where the agents stay in their position. The actions are illustrated in Figure 7 where the arrows represent the directions the agent can follow and the "stay" actions means that the agent won't move. This action was added with the purpose of preventing any issue that can harm the reward like the rebound effect reported in [5].

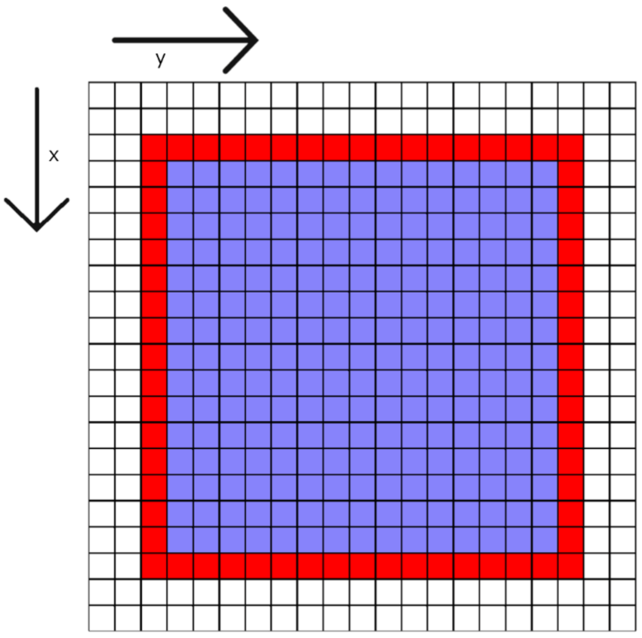


Figure 6 Representation of the environment

The maximum number of movements was set at 21, which is the same as the length of one side of the environment grid. An episode is over when this number is reached or if the agent moves past the internal border, which will be known as a "fall".

Criteria	**Feature of the environment**	**Description**
Accessibility	Accessible	Agents can obtain detailed and precise information from the environment.
Determinism	Deterministic	Every agent's action has a known effect.
Dynamism	Static	The dynamics of the environment do not change.
Continuity	Discrete	There is a fixed number of actions and states

Table 1 Environment characterization

Any movement where the agent doesn't "fall" has an initial reward of 100, which was a value found through experimentation, this reward was called the movement award. The definitive value of the reward is determined by how far the agent is to the internal border, since the internal border has 4 sides, the distance to each one of them is determined and the minimum of these distances is taken into consideration. Equations 13 to 16 represent how the distances are measured; *agentT* is the distance of the agent to the top border, *agentD*, the distance to the inferior order, *agentL*, the distance to the left border and *agentR*, the distance to the right border. The formula to obtain the step reward of an agent is shown in Equation 17.

Without these equations, the agent would lack the incentive to move towards the border. It may opt for a conservative movement strategy that keeps it in a central position while still making the maximum number of movements. To encourage the agents to move towards the border, a bigger reward was required to be obtained in the positions closer to the border. A unit is added to Equation 16 in order to prevent any division by 0, since when an agent is on the border, the minimum distance is 0.

$$agentT = abs(agent's\ x\ position - superior\ border) \qquad (13)$$

Equation 13 Distance to the top border

$$agentD = abs(agent's\ x\ position - inferior\ border) \qquad (14)$$

Equation 14 Distance to the inferior border

$$agentL = abs(agent's\ y\ position - left\ border) \qquad (15)$$

Equation 15 Distance to the left border

$$agentR = abs(agent's\ y\ position - right\ border) \qquad (16)$$

Equation 16 Distance to the right border

$$stepReward = \frac{100}{min(agentT, agentD.agentL, agentR) + 1} \qquad (17)$$

Equation 17 Step reward equation

The other value of reward involved is when the agent "falls" this reward has a value of –1600. This number was determined by the number of movements an agent would have to take when it starts in the extreme of the blue zone and wants to get to the internal border of the opposite side, which is 16, by -100, which is the reward for a movement. This formula is shown in Equation 18.

$$number\ of\ steps\ to\ reach\ opposite\ site = 16$$

$$fall\ reward = number\ of\ steps\ to\ reach\ opposite\ site * -100 \qquad (18)$$

Equation 18 Fall reward equation

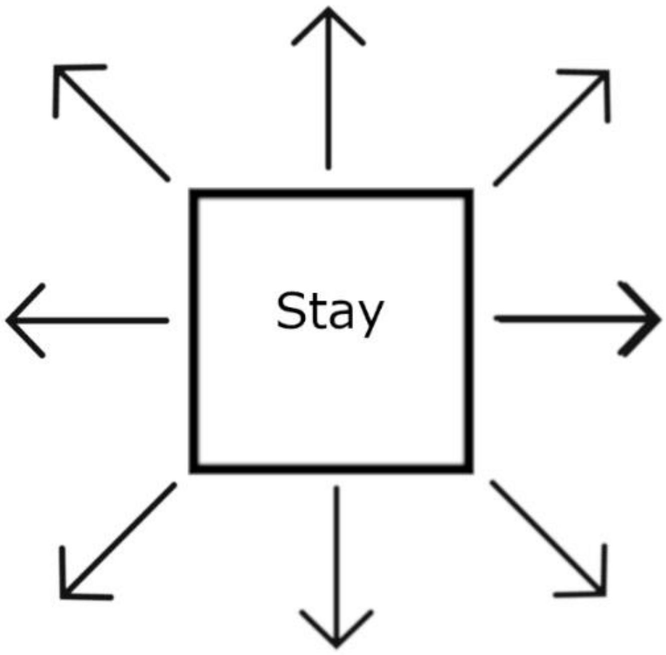


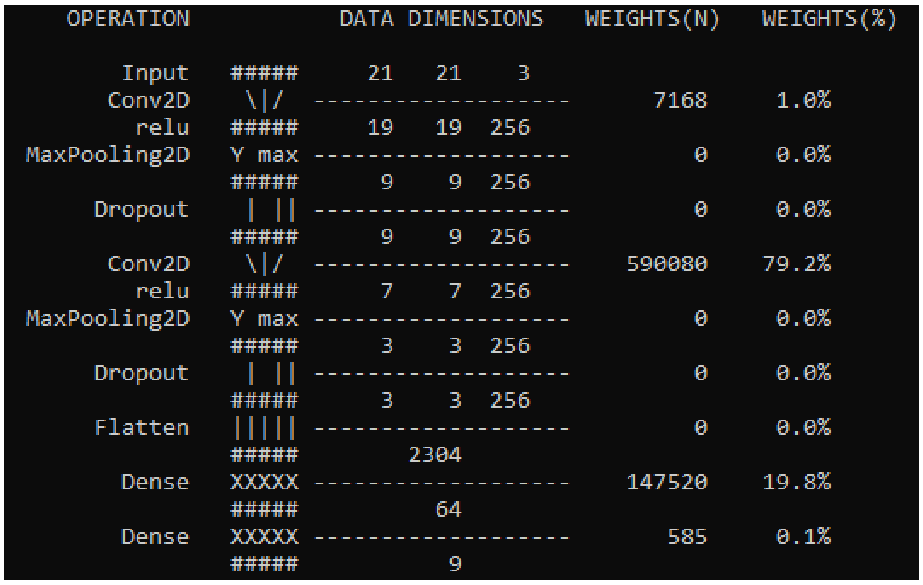
Figure 7 Representation of the actions available for the agent

The difference with the work introduced in [5] is that since Q-learning was used, a Q-table exists, and it’s the one responsible for determining the action when the agent exploits its knowledge. Now in this paper, deep Q learning is the algorithm used, and the mapping between actions and values are performed by a model, trained by a deep network.

Since each Explorer agent of the architecture had a copy of the neural network within, the architecture was constrained in terms of complexity, since running several agents at once would have reached the maximum processing capabilities. To achieve this, the number of layers was minimized. The first layers of the architecture are made of two blocks that contain convolutional, pooling and dropout layers to extract the features. Subsequently, a flatten layer was introduced to reduce the dimensions of the extracted features. Finally, a couple of fully connected layers were incorporated to classify the features and produce an output that determines the action to be taken.

The dimensions of the input layer is 21x21x3 since as it was seen in Figure 6, the length is 21 and since it’s a rgb image, the number of channels is 3. On the other hand, the dimension of the output layer is 9 since is the number of available actions. After experimenting with some nonlinear activation functions, *relu* was chosen due to the superior performance showed in achieving the desired results.

The architecture of the network is described with the help of Figure 8 and 9. Figure 8 shows a summary of the neural network used, and Figure 9 shows a more graphical representation.



```
   OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

       Input   #####     21   21    3
      Conv2D    \|/  -------------------      7168     1.0%
        relu   #####     19   19  256
MaxPooling2D   Y max -------------------         0     0.0%
               #####      9    9  256
     Dropout    | || -------------------         0     0.0%
               #####      9    9  256
      Conv2D    \|/  -------------------    590080    79.2%
        relu   #####      7    7  256
MaxPooling2D   Y max -------------------         0     0.0%
               #####      3    3  256
     Dropout    | || -------------------         0     0.0%
               #####      3    3  256
     Flatten   ||||| -------------------         0     0.0%
               #####       2304
       Dense   XXXXX -------------------    147520    19.8%
               #####         64
       Dense   XXXXX -------------------       585     0.1%
               #####          9
```

Figure 8 Convolutional Neural Network summary

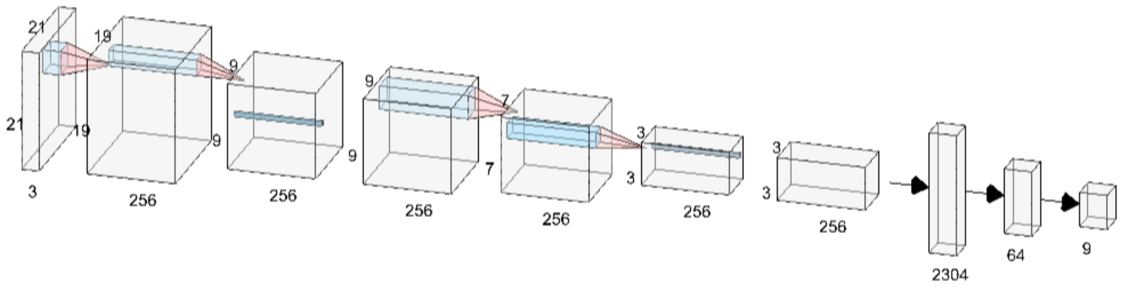


Figure 9. Convolutional Neural Network graphical representation

3.2 Multi-agent architecture design.

Just as in the work introduced in [5], centralized planning and decentralized execution paradigm is used, along with the pair-based concept.

The agents that are part of each pair have different roles and based on this role they are given a name. The agent in charge of interacting with the environment was called an “Explorer”, on the other hand, the agent in charge of receiving the information from the “Explorer” was called an “Accumulator”. Each pair was identified by the number at the end of their names. An “Explorer” is the one who performs an action, receives a reward and trains the model, but by itself it’s unable to know about the actions performed by the other “Explorers”, that’s the importance of the “Accumulator”. An “Accumulator” is in charge of receiving the information from all the “Explorers” in the system and make it available for its correspondent “Explorer”. An illustration of this architecture is shown in Figure 10.

An Explorer has the ability to send two types of messages, the first is an Inform and is sent when it makes a mistake in the environment, this message is sent to all the “Accumulators”. The other message is a “Request” and is done to its “Accumulator” only. An “Accumulator” can send messages only to its correspondent “Explorer” and is the response of the request started by said “Explorer”.

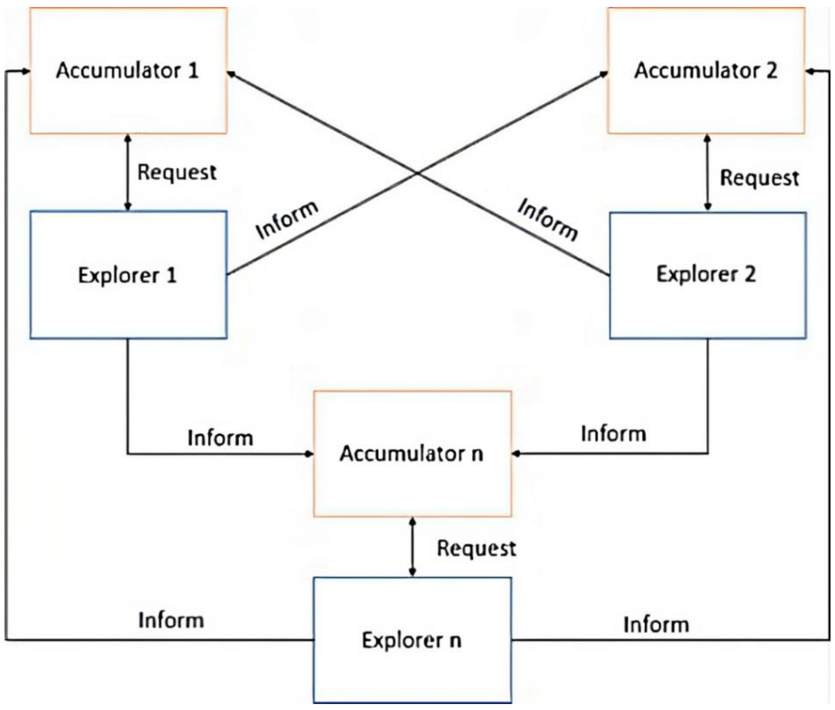


Figure 10 Multi-agent architecture

This model of communication relies on a peer system which means that the agents along the system have the exact same features, these features according to the work presented in [15], are characterized for this work in Table 2. Regarding the feature called “Interaction-specific features”, there are a number of sub features which are described in Table 3.

Feature	**Description regarding this work**
Degree of decentralization	The pair of agents learn in parallel
Interaction-specific features	See Table 3.
Involvement-specific features	Goal attainment could be executed by any pair of agents and all the pairs of agents contain the same type of agents.
Goal-specific features	Agents try to reduce the number of episodes required to learn a policy where no mistakes are made and this goal does not create conflict along agents.
Learning method	Learning by discovery
Learning feedback	Reinforcement learning

Table 2 Features of the agents involved in the architecture

Interaction-specific features	**Description according to this work**
Level of interaction	Information exchange
Persistence of interaction	Long-term
Frequency of interaction	High
Pattern of interaction	Different strategies of communication along Explorers and Accumulators
Variability of interaction	Fixed

Table 3 Interaction-specific features of the agents involved in the architecture

3.3 Integration of the reinforcement learning environment with the multi-agent architecture

The third and last phase of this project was to create a successful integration of the multi-agent architecture with the reinforcement learning system. The goal was for the agents not to only learn about the environment but also communicate and take advantage of that communication. It was expected that the agents were capable of reaching a policy where errors were not made, and if the returns obtained by this policy were the maximum available, it can be considered an optimal policy.

The important aspects of this integration are what and when agents communicate. To understand what is communicated, it's necessary to refer to the tuples that are part of the replay memory of the agent. For this integration, information was added to the tuple. This extra information is a flag which says whether the current state that is being sent is a fallen state or not. Each tuple was referred to as an observation. These observations flow throughout the system, allowing the agents to take advantage of the insights obtained from their peers.

On the other hand, related to when the agents communicate, Figure 11 shows a diagram of it; it can be seen the moments where agents send the messages. The first moment where an agent sends a message is when the Explorer starts an episode and makes a request to its Accumulator. In that request, it sends a list of observations that the Explorer previously received and for instance, are already known for the Explorer; this list is called the known observations list (KOL). This list is sent with the purpose of filtering the observations that will be in the response of the Accumulator. The Accumulator receives the KOL from its Explorer couple and since it has been receiving the observations of the other Explorers of the system, uses the KOL to filter the observations. Consequently, redundant information is not sent, and the Explorer won't have repeated observations.

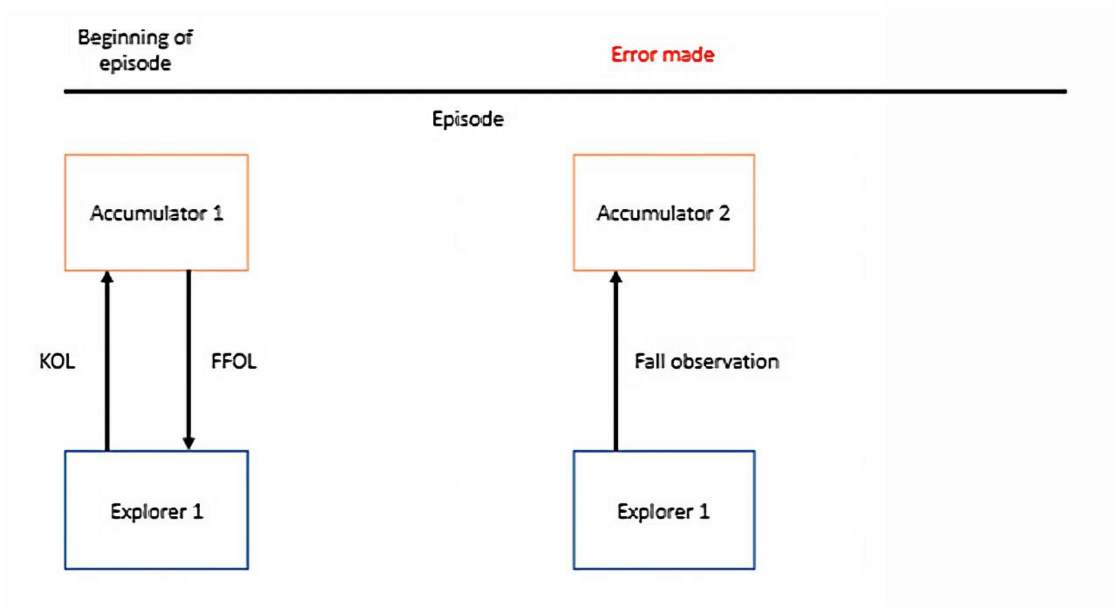


Figure 11 Communication moments

The second moment of communication happens when an agent "falls" and sends the observation to all the accumulators except for its own couple. Accumulators receive this observation and add it to a list called the fall observations list (FOL). The observations received that are added to this list comes from any Explorer of the system as shown in Figure 12. This list is the one that gets filtered when the accumulator receives the KOL from its Explorer couple. This filter consists in deleting from FOL the observations that are contained in KOL, preventing the Explorer from receiving repeated observations. The filter list is called the filtered fall observations (FFOL) which is sent back to the Explorer as response which is a process illustrated in Figure 13.

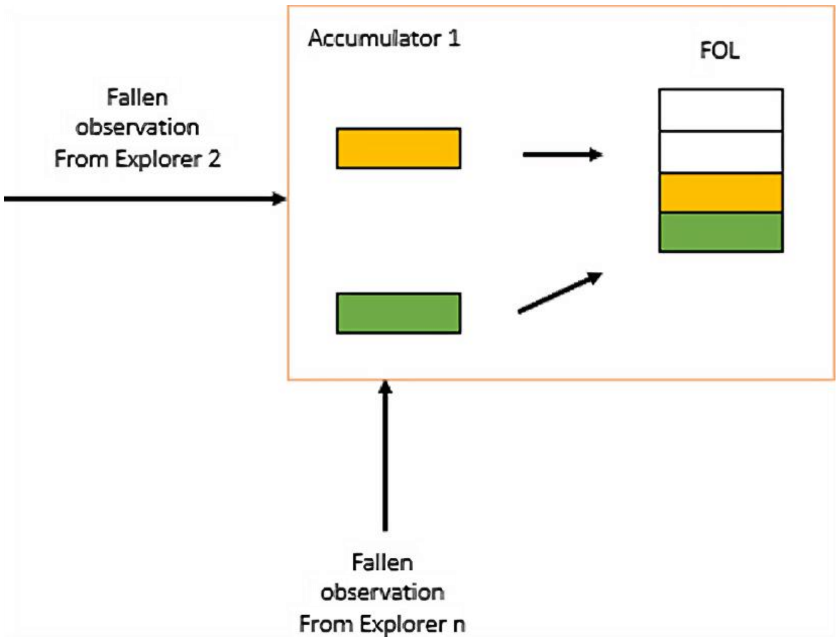

Accumulator 1
FOL
Fallen observation From Explorer 2
Fallen observation From Explorer n

Figure 12 Accumulator forming its KOL list

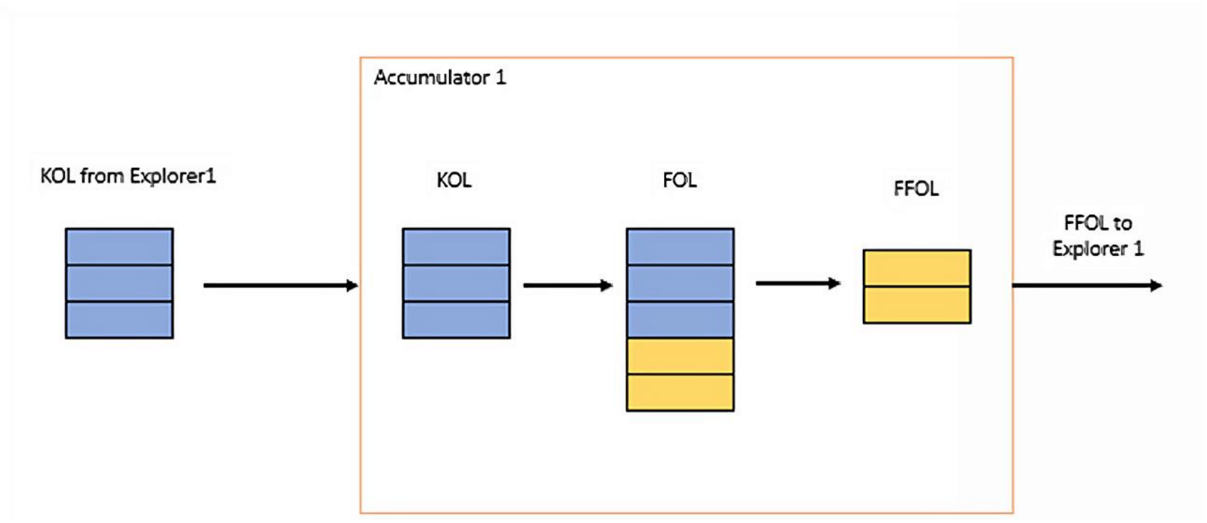


Figure 13 Filter inside accumulator

Once the Explorer receives the FFOL it needs to interpret the information that has received. Precisely, these observations were part of the minibatch that was sampled from the experience replay to train the neural network during that episode. These observations are considered in the minibatch immediately as they are received but are also added to the experience replay for a possible future use. This means that the number of observations the agent will be trained with, will be bigger than the stablished size of 64. Which is the standard size of a batch obtained from the experience replay. The observations received by the explorer will also be added to KOL as shown in Figure 14. Now that the observations were part of the sample to train the network, it was necessary to assimilate the info in these observations, and that’s where the extra information that was added to tuple played a role.

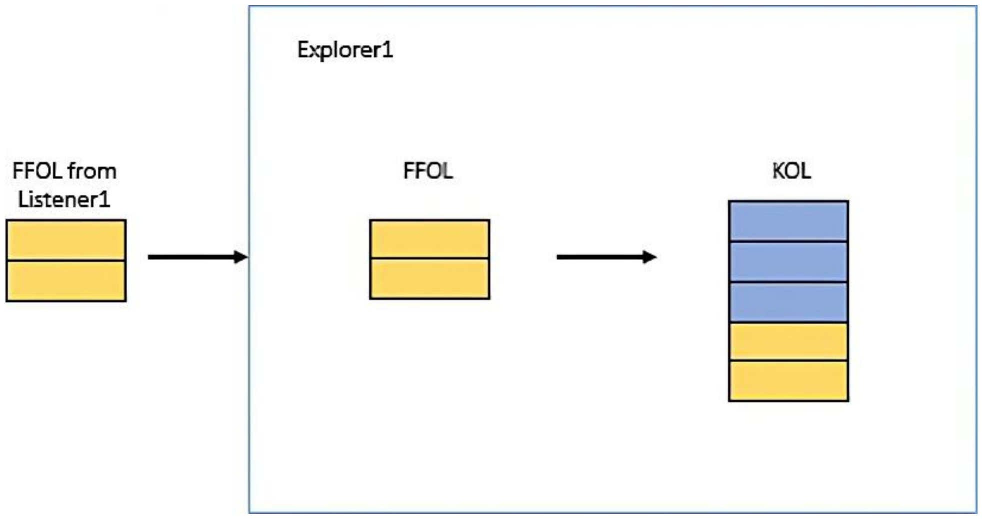


Figure 14 Explorer forming KOL

Since the network is going to train how to map a state to an action, the value for the action that keeps the agent in the same position was increased, specifically, its value will be the opposite of the fall reward, 1600. It's a process that happens every time the fallen state flag of the tuple has a true value. By doing that, the neural network will train for that state with a big negative reward for the action that led the agent to fall and with a big positive value for the action that keeps the agent in the same position. The objective of this change in the agent is to motivate the agent to stay in the same position right on the border and not trespass this border. The perimeter of the environment is 68 (Figure 6). By sharing learning, an agent can access a wider range of observations beyond its own. Through this, an agent can learn more efficiently and in fewer interactions compared to if it were only observing the states of the perimeter independently.

Another aspect in the explorer regards the application of epsilon-greedy. This application of epsilon-greedy is limited in the sense that it won't be applied to states that are included in KOL. Which means that once that the q values for a state have been manipulated (Figure 14), this mentioned process would not be applied to that state again. Instead, a straight exploitation is performed, as seen in Figure 15.

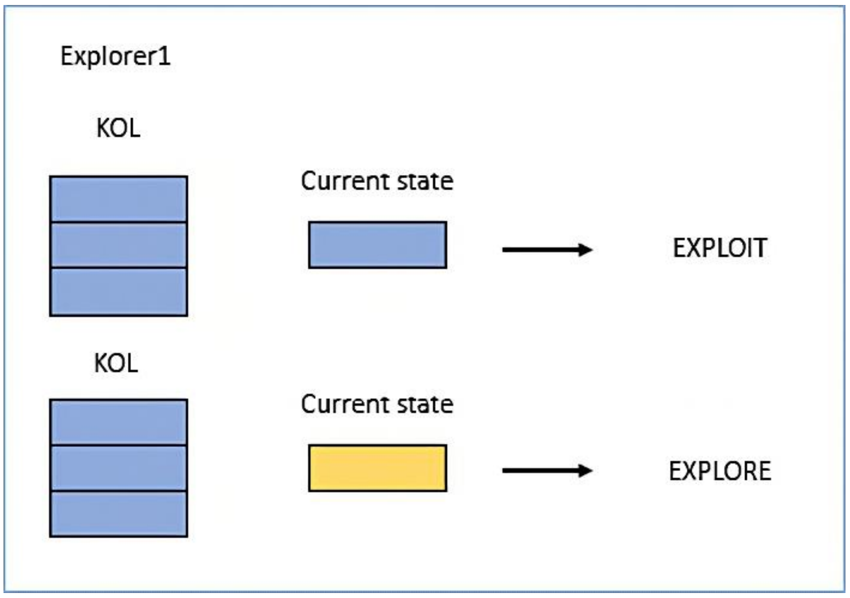
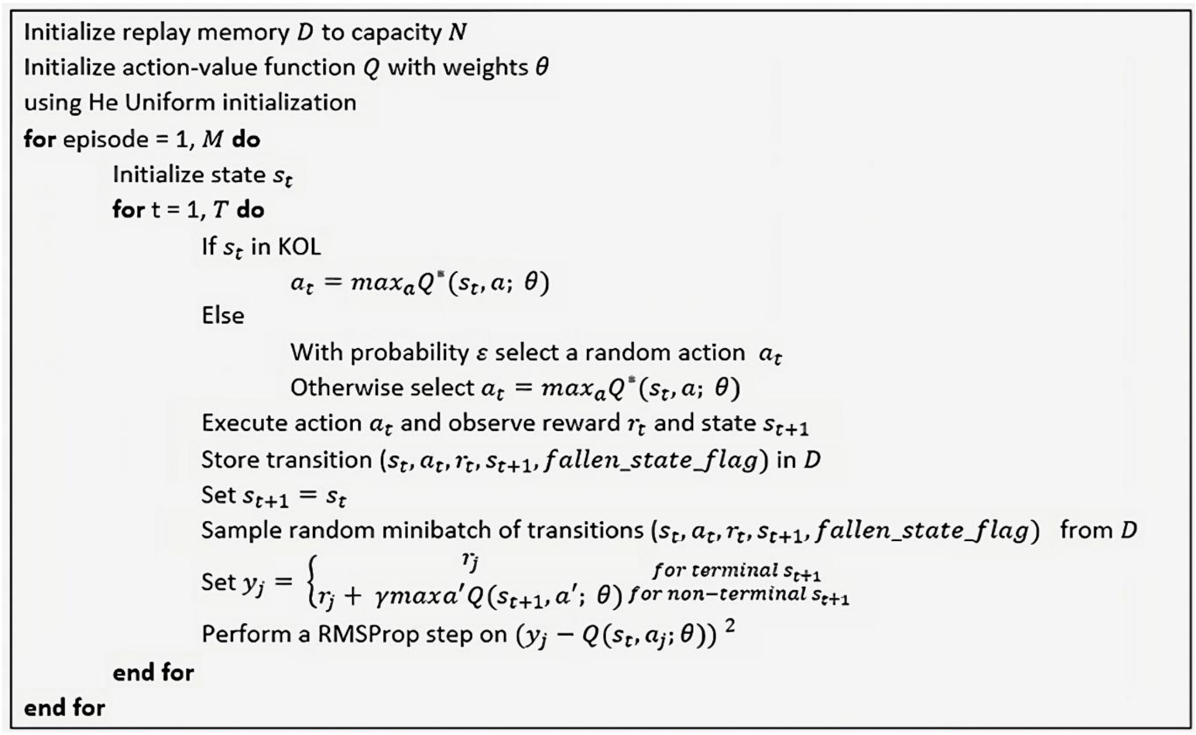


Figure 15 Epsilon-greedy limitation performed by the Explorer agent

The way the explorers handle the messages and apply the RL algorithm introduces a variation in the Deep Q learning algorithm apart from the initializer which is He uniform [16] and optimizer which is RMSProp [17]. This mentioned variation can be seen in Figure 16. Once the agent reached a final state a counter was increased, once the counter got to 5, the target network was updated with the weights of the online network. The value of 5 to perform the update was found based on the favorable results observed during experimentation. Following the same principle, the value of the discount factor was obtained, which was 0.95.



```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with weights $\theta$
using He Uniform initialization
for episode = 1, $M$ do
    Initialize state $s_t$
    for t = 1, $T$ do
        If $s_t$ in KOL
            $a_t = max_a Q^*(s_t, a;\ \theta)$
        Else
            With probability $\varepsilon$ select a random action $a_t$
            Otherwise select $a_t = max_a Q^*(s_t, a;\ \theta)$
        Execute action $a_t$ and observe reward $r_t$ and state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1}, fallen\_state\_flag)$ in $D$
        Set $s_{t+1} = s_t$
        Sample random minibatch of transitions $(s_t, a_t, r_t, s_{t+1}, fallen\_state\_flag)$ from $D$
        Set $y_j = \begin{cases} r_j & for\ terminal\ s_{t+1} \\ r_j + \gamma maxa' Q(s_{t+1}, a';\ \theta) & for\ non{-}terminal\ s_{t+1} \end{cases}$
        Perform a RMSProp step on $(y_j - Q(s_t, a_j; \theta))^2$
    end for
end for
```

Figure 16 Variation of Deep Q learning

Since there is an extra item in the transitions that are stored in the experience replay, this work is placed along a couple of works which follow this strategy. A description of these, is seen in Table 4.

Name	Description
Lenient Multi-Agent Deep Reinforcement Learning [18]	Adding a leniency value to the transition allows to determine whether the sample is considered or not
Stabilizing Experience Replay for Deep Multi-Agent Reinforcement Learning [19]	Add information to help disambiguate the age of the data in the form of the number of the iteration during training
Present work	Add a flag that helps characterizing a relevant state

Table 4 Summary of works under the same principle

To compare the performance of the agents using the multi-agent architecture with a unique agent, decrement percentage formula was used, and the metric to consider was the number of episodes an agent needs to learn to not trespass the internal border of the environment.

4. Results

To quantify what would be the improvement of using the multi-agent architecture, first, one only agent is placed in the environment with the purpose of observing to three certain metrics.

The first is the average reward, which is computed every 50 episodes; in the same interval, the number of errors made by the agent is accounted as well. The third is the accuracy of the model.

4.1 Results with no communication

Figure 17 shows the number of errors made by the agent, this metric represents the number of times an agent has “fallen” or trespassed the internal border of the environment. For the agent that acted by itself in the environment, it’s observed that this number increases during the early stages of the training, specifically the number increases until around 5000 episodes. After that, it is observed how the number of errors decreases until obtaining a flat line, which means no errors, at around 45000 episodes.

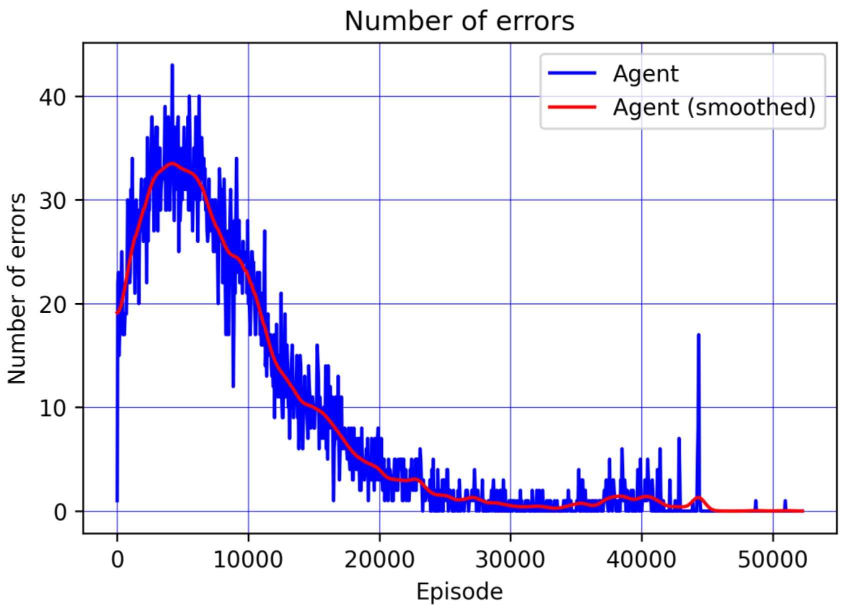


Figure 17 Number of errors made by the lonely agent

Figure 18 shows that around 45000 episodes the accuracy was 0.85. The accuracy shows values higher than 0.9 at around 16000 episodes, it is observed a decrease in the performance after the mark of the 41000 episodes, however, it handles to recover performance getting to be placed in the same range of values around 0.9. Regarding the average reward, Figure 19 shows that the agent stabilizes this reward at around 25000 episodes obtaining its maximum values at around 45000 episodes.

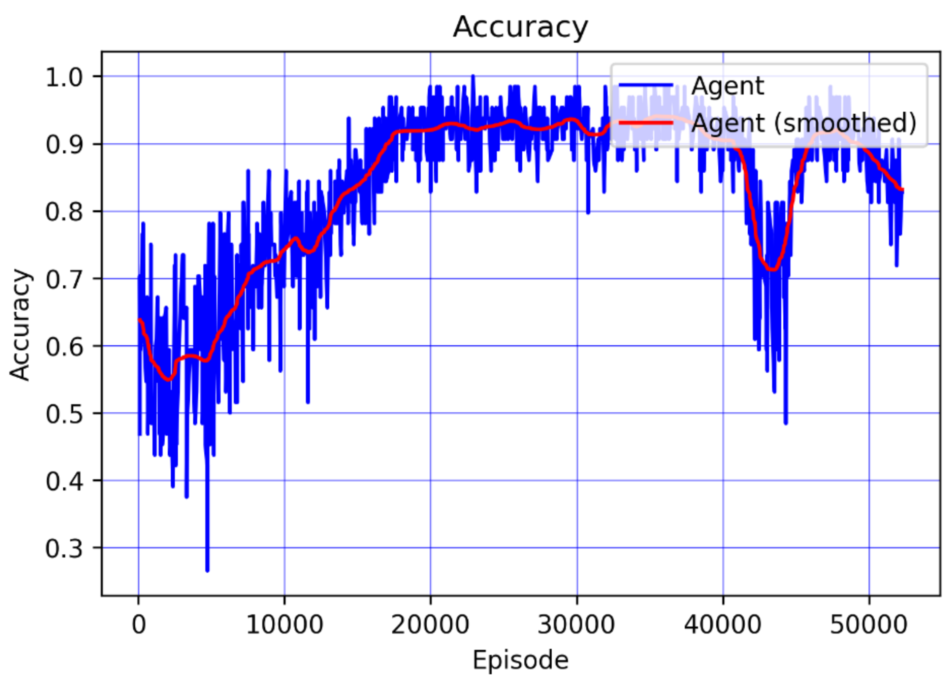


Figure 18 Number of errors made by the lonely agent

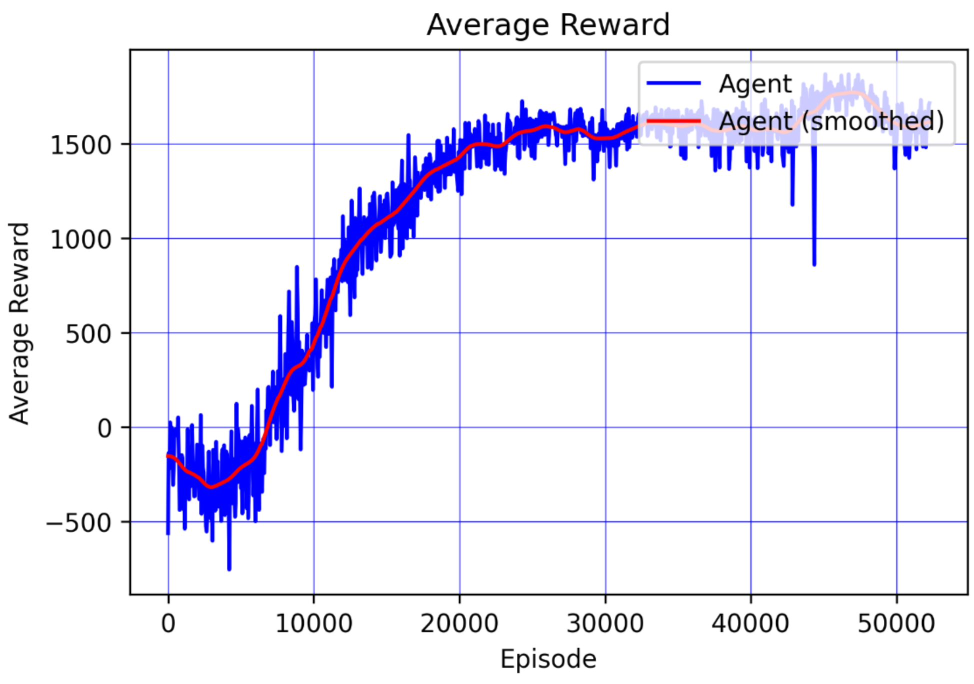


Figure 19 Average reward values obtained by the lonely agent

4.2 Results with communication

To avoid any confusion with the pair-based architecture concept, a constraint was put in place to ensure that the number of pairs of agents could not be two. Additionally, due to processing limitations, the number of pairs could not exceed 3. Thus, the reported results were obtained using 3 pairs of agents.

For 3 pairs of agents, Figure 20 shows that the explorer 1 obtained 0 errors at around 3900 episodes and continues to make 0 errors for the subsequent episodes. A different behavior is seen for gents 2 and 3. In the case of agent 2, a similar behavior to agent 1's is seen until gets around 5750 episodes when it starts to make errors again, however, after less than 2000 episodes later, precisely, at 6900 episodes, it gets back to a behavior where no errors are made.

Explorer 3 shows a different behavior, first, it shows a similar behavior to the one performed by explorer 2, then starts to make errors at around 4900 episodes and gets to make no errors at around 5400 episodes. The difference is that at around 7900 episodes, it starts to show a behavior where during some intervals it makes no errors, but during other intervals it makes several errors even obtaining episodes with more than 20 errors.

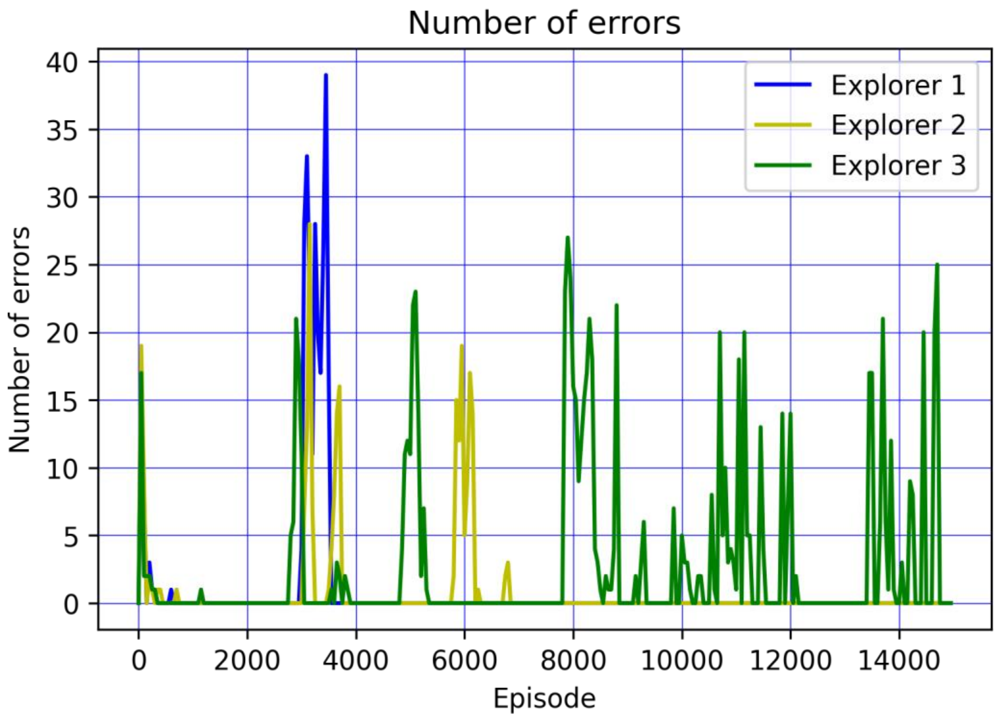


Figure 20 Number of errors made by the explorers using the multi-agent architecture

Figure 21 shows the average reward obtained by the explorers of each pair. It is observed that the explorers get the maximum average reward at around 10000 episodes with values around 18000. It is also observed that two of the agents, precisely, agents 2 and 3 show a kind of a negative spike around 1000 episodes later, however, agent 2 obtains the maximum reward again a few hundreds of episodes later and onwards. That is not the case for agent 3 since after around 7000 episodes, the average rewards show a considerable decrease on its value.

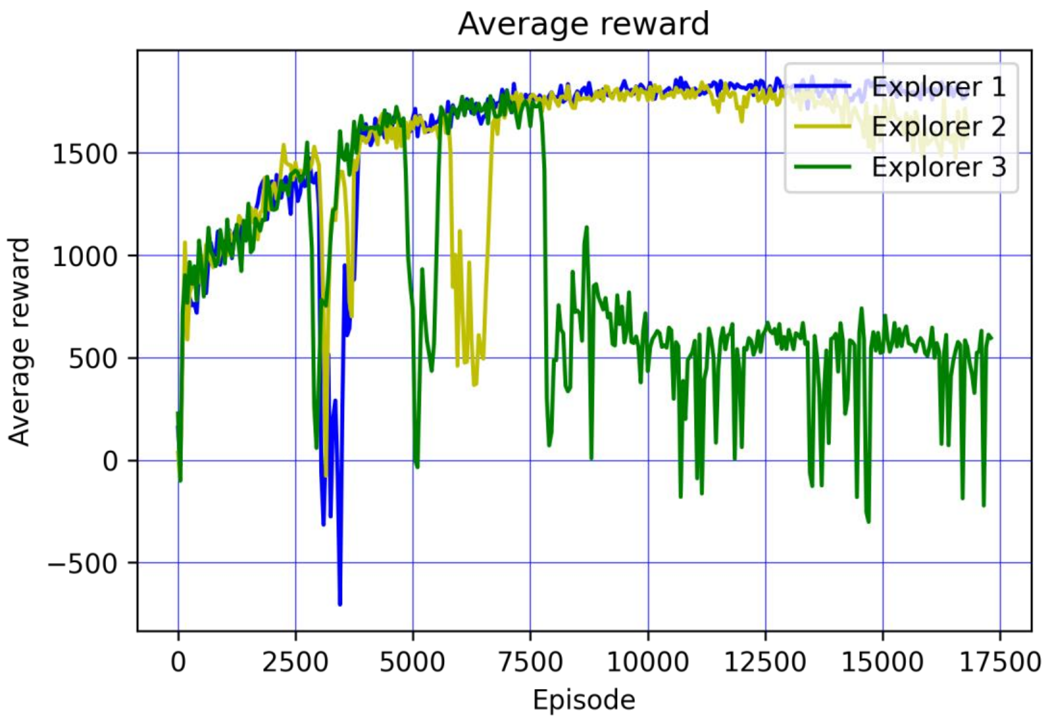


Figure 21 Average reward of the explorer agents using the multi-agent architecture

There are moments where the average reward reaches its maximum value which is 18000, but for other episodes the average reward is considerably lower compared with the values obtained by the remaining explorers of the system. Similar results were observed comparing the number of errors.

Comparing Figure 21 with Figure 19 for the best Explorer agent, it is observed that the rewards obtained by the Explorer agent are not smaller than those obtained by the independent agent. This suggests that the Explorer agents were able to achieve the maximum possible reward, as opposed to the results presented in [5] where the independent agent performed better in this specific metric. This effect was documented as a consequence of a called *rebound effect*. Since the maximum values of the reward were obtained, the resulting policies can be considered optimal.

In Figure 22, it's seen that the number of episodes that the agent makes no error can be found at around 5000 episodes, the accuracy is not over 0.8, however it starts to get the maximum values at around 16000 episodes.

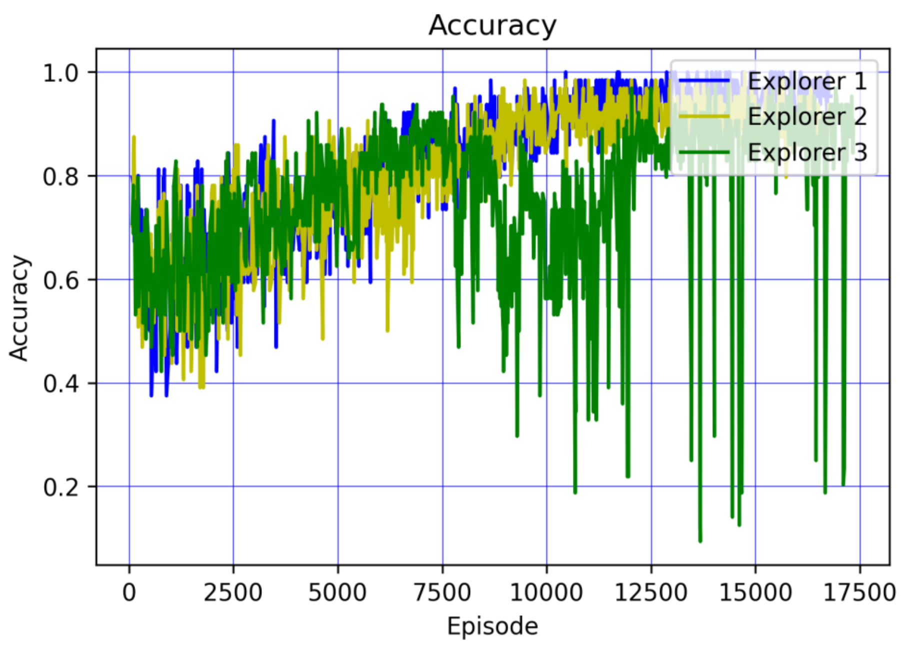


Figure 22 Accuracy of the explorer agents using the multi-agent architecture

4.3 Discussion

After comparing the results of an independent agent and 3 Explorer agents, it was observed that there is a decrease regarding two metrics. These metrics are the number of episodes to reach a policy where no errors are made and the number of episodes where the maximum value of the reward is reached. These metrics are accuracy, maximum value of reward and the number of episodes where the agents reduced their errors to zero. It's important to remark that to make this comparison, the best of the explorers is the one taken in consideration. An explorer is considered the best if is the one that has obtained a policy where no errors are made in the smaller number of episodes.

This best explorer managed to obtain a policy where no errors are made in the environment in around 3900 episodes. On the other hand, the independent agent achieved the same in around 45000 episodes. This represents a decrease of 86.6%. Considering the maximum value of the reward, the best explorer agent reached this value at around 10000 episodes and the independent agent reached this value at around 45000 episodes. These numbers represent that the Explorer agent achieved a better performance than the independent agent since it has managed to decrease the number of episodes by 77.78%. These two metrics show that the Explorer agent has an advantage over the independent agent.

The last metric shows a parity between the Explorer agent and the lonely agent. The metric consists of the number of episodes required to reach the best value of accuracy. Both, the best Explorer agent and the lonely agent managed to reach the best value of accuracy at around 1600 episodes. The comparison of the three metrics can be seen in Table 5.

In Table 6, it's observed a comparison of the results obtained by the experimentations of this work with its "shallow" learning counterpart [5] for 3 pairs of agents. Since the metric that considers the accuracy of the model did not apply to the previous work, it is not included in the comparison. Comparing the results of the experiments with no communication and the experiments using the multi-agent architecture, is observed that a decrement exists in both works. However, the work that used Q-learning instead of Deep Q-learning shows a larger decrement, to be precise 97.7% compared to 86.6%

On the other hand, regarding the metric of episodes needed to obtain the maximum values of the reward, it is observed that the deep learning approach shows a larger decrement, to be precise, 77.78% to 61.1%. It is important to notice that since in this work not all the agents managed to obtain a policy where no errors were made, the midrange formula could not be applied as in the work introduced in [5]. It should be noted that these results only apply to the specific metrics considered and do not necessarily generalize to other performance criteria.

Metric	No Communication	With communication (best agent)
Episodes to get the best value of accuracy	16000	16000
Episodes to get the maximum value of reward	45000	10000
Episodes to make no errors	45000	3900

Table 5 Summary of results

Metric	Work introduced in [5] No communication	Work introduced in [5] using its multi-agent architecture with 3 pair of agents	This work no communication	This work using its multi-agent architecture with 3 pair of agents
Episodes to get the maximum value of reward	45102	Around 17500	45000	10000
Episodes to make no errors	45102	1030 *	45000	3900

Table 6 Comparison with the results reported in the work introduced in [5]

* Average between the value obtained by the agent that needed the fewer number of interactions and the one that needed the bigger number of interactions, midrange formula.

5. Conclusions and Recommendations

This section introduces the conclusions based on the objectives introduced in former sections. Also, some recommendations to be considered in possible future works on the same line of research.

5.1 Conclusions

The multi-agent architecture proposed in this work has achieved a decrease in the number of episodes necessary to obtain a policy where no errors are made without harming the value of the maximum reward. Since the maximum rewards have been obtained, it can be considered an optimal policy.

The deep reinforcement learning model provided the agents with an exploration task where agents must not only explore the environment but also find the limits in it. The necessary actions have been provided in order to prevent any issue during the obtention of the reward like the rebound effect.

A deep reinforcement model has been designed to provide the agents with an exploration task that involves not only moving around a delimited zone but finding the limits of the zone. The actions provided to the agent were effective in achieving this objective, as no rebound effect was observed during the experiments.

The absence of the rebound effect is associated with a reduction in the number of episodes needed to obtain the maximum reward, as the agent is now capable of taking an action that allows it to remain in its position and receive a larger reward at the end of an episode.

Comparing the deep learning approach with the shallow learning approach in the number of episodes to obtain a policy where no errors are made, it was observed that there is a difference of about 10% in the decrement. Which is an effect that can be attributed to the increase of complexity of the algorithm and the increased number of actions available to the agent in this work.

A multi-agent architecture has been redesigned to take care of the new information that flows through the environment as a consequence of the use of a deep learning algorithm. The pair-based approach has proven to be effective handling not only information in the form of numbers but also in the form of big arrays which represent an image.

The integration of the multi-agent architecture with a variant of the deep reinforcement learning algorithm has permitted the agents to reach a policy where no errors are made in fewer interactions. The multi-agent architecture enables agents to leverage the knowledge of their peers by incorporating additional information into the tuples stored in the experience replay. Specifically, the identification of a state where errors are made, has allowed to train the neural network with the appropriate inputs and outputs so errors can be reduced, and at the same time the reward obtained is not harmed.

5.2 Future Work

During training, not all the Explorer agents managed to obtain an policy where no errors are made. As future work, this anomaly can be assessed with the objective of finding the reasons or exact component of the architecture that causes the issue and for that, further experimentation will be required.

Regarding scalability, the number of agents were limited due to processing capabilities, for that reason, increasing the number of agents in order to find new optimizations remains to be seen. Another possible area for further experimentation is changing the primary objective the agents. That would require a different design regarding the actions and rewards of the Explorer agents in order to generate a working model and consequently, a successful integration with the pair-based architecture.

References

[1] M. Wooldridge, 'An Introduction to Multi Agent Systems', 2nd ed. John Wiley & Sons, 2009.

[2] Y. Li, ‘Deep Reinforcement Learning: An Overview’, CoRR, vol. abs/1701.07274, 2017..

[3] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, ‘Fundamental Concepts of Convolutional Neural Network’, in Recent Trends and Advances in Artificial Intelligence and Internet of Things, V. E. Balas, R. Kumar, and R. Srivastava, Eds. Cham: Springer International Publishing, 2020, pp. 519–567.

[4] R. Sutton and A. Barto, Reinforcement learning An Introduction, 2nd ed. The MIT Press, 2018.

[5] Cárdenas Guilcapi, D.A., Paz-Arias, H., Galindo, J. (2020). Design and Implementation of a Multi Agent Architecture to Communicate Reinforcement Learning Knowledge and

Improve Agents' Behavior. In: Rodriguez Morales, G., Fonseca C., E.R., Salgado, J.P., Pérez-Gosende, P., Orellana Cordero, M., Berrezueta, S. (eds) Information and Communication Technologies. TICEC 2020. Communications in Computer and Information Science, vol 1307. Springer, Cham. https://doi.org/10.1007/978-3-030-62833-8_32

[6] X. Han, A mathematical introduction to reinforcement learning, 2018.

[7] A. Maroti, 'RBED: Reward Based Epsilon Decay', CoRR, vol. abs/1910.13701, 2019.

[8] A. Plaat, 'Deep Reinforcement Learning', CoRR, vol. abs/2201.02135, 2022.

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, 'A Brief Survey of Deep Reinforcement Learning', CoRR, vol. abs/1708.05866, 2017.

[10] V. Mnih et al., 'Playing Atari with Deep Reinforcement Learning', CoRR, vol. abs/1312.5602, 2013.

[11] Jafari, R., Javidi, M.M. 'Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning', SN Appl. Sci. 2, 259 (2020). https://doi.org/10.1007/s42452-020-2012-0.

[12] L.-J. Lin, 'Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching', Machine Learning, 1992, pp. 293-321.

[13] P. Balaji and D. Srinivasan, "An Introduction to Multi-Agent Systems", Innovations in Multi-Agent Systems and Applications 1, pp. 1-27, 2010.

[14] N. A. Vlassis, 'A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence', in A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence, 2007.

[15] Sandip Sen and Gerhard Weiss, 'Learning in Multiagent Systems', Multiagent systems, The MIT Press, 1999.

[16] K. He, X. Zhang, S. Ren and J. Sun, 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,' 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026-1034, doi: 10.1109/ICCV.2015.123.

[17] Tieleman, T. and Hinton, G. (2012) Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude. COURSERA: Neural Networks for Machine Learning, 4, 26-31.

[18] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, 'Lenient Multi-Agent Deep Reinforcement Learning'. arXiv, 2017.

[19] J. Foerster et al., 'Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning'. arXiv, 2017.

[20] T. Brys, 'Reinforcement Learning with Heuristic Information', 2016.

[21] F. Sultana, A. Sufian, and P. Dutta, ‘Advancements in Image Classification using Convolutional Neural Network’, in 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), 2018.