UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Q-learning to develop an IoT network controlled by an agent

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniera en Tecnologías de la Información

Autora:

Cabascango Anrango Gissel Vanessa

Tutor:

Ph.D. - Armas Arciniega Julio Joaquín

Urcuquí, octubre 2022

Autoría

Yo, **GISSEL VANESSA CABASCANGO ANRANGO**, con cédula de identidad 1716247943, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, octubre 2022.

GISSEL VANESSA CABASCANGO ANRANGO

Firmado digitalmente por GISSEL VANESSA CABASCANGO ANRANGO
Fecha: 2023.02.09 19:05:53 -05'00'

Gissel Vanessa Cabascango Anrango

CI: 1716247943

Autorización de publicación

Yo, **GISSEL VANESSA CABASCANGO ANRANGO**, con cédula de identidad 1716247943, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, octubre 2022.

GISSEL VANESSA CABASCANGO ANRANGO

Firmado digitalmente por GISSEL VANESSA CABASCANGO ANRANGO
Fecha: 2023.02.09 19:06:54 -05'00'

Gissel Vanessa Cabascango Anrango

CI: 1716247943

Dedication

"Dedicated to my family, especially to Rafael Cabascango and Matilde Anrango, my parents and the pillars of my life."

Gissel Vanessa Cabascango Anrango

Acknowledgment

First, I want to make a special acknowledgment to God, who has never left me on this long road called life. For your blessings and lessons, thank you, God. Second, I thank my family, whom I love and treasure above all else and who are always there in any circumstance. I thank my parents, who have given me unconditional support throughout my academic career and life. Thank you, daddy, for sharing everything you have learned in your whole life and inspiring us to reach very high. Thank you, mommy, for teaching us to love God and guiding us along his path. I thank my siblings Jany, Richard, Zuly, Tefy, and Steeven for sharing many moments with me and accepting me as I am. To my little Avril, thank you for coming to this home and rejoicing with us every day since March. You have made us very happy from the moment we knew you were on your way. I want to thank my friends and colleagues Susy, Liz, Cinthy, Fer, Oso, and Fabian, with whom I shared many fun and bad moments at the university and who hold a special place in my heart.

I thank Yachay Tech University for giving me the opportunity to learn from excellent teachers who inspire and motivate me not to give up. Finally, I want to especially thank my thesis tutor Oscar Chang, who patiently guided me and shared all his knowledge to develop this project. In addition, I want to thank Professor Julio Armas who was pending the approval of the pre-professional practice hours and the thesis.

Gissel Vanessa Cabascango Anrango

Resumen

Hoy en día la interacción entre los sistemas informáticos y las personas es más común y frecuente a nivel mundial. Como resultado, las técnicas de inteligencia artificial (IA) se han vuelto cada vez más relevantes y necesarias en la vida cotidiana, ya que sus aplicaciones permiten la automatización de procesos que comúnmente se realizan de forma manual. Además, a nivel tecnológico, la IA puede hacer frente a problemas a gran escala, como el manejo de datos masivos, problemas de seguridad e interacción hombre-máquina.

A medida que avanza la tecnología es necesario darle importancia a los problemas presentes en el día a día de los ciudadanos ya que afectan su calidad de vida. Las Smart Cities vienen a solucionar problemas de interés social, medioambiental, de movilidad, entre muchos otros. Por eso es tan relevante. Como contribución al crecimiento de las Smart Cities, este proyecto desarrolla un sistema que resuelve un problema común en las ciudades urbanas, como es la búsqueda de rutas óptimas en casos de congestión de tráfico o emergencias. Utilizamos las tecnologías de Internet de las cosas (IoT) para crear un sistema distribuido que nos permite establecer una red global donde es posible el intercambio de datos entre dispositivos, sensores y humanos. Basándonos en un modelo cliente-servidor, buscamos conectar nodos ESP8266 que puedan comunicarse a través de un conocido broker llamado MQTT. A su vez, se desarrolla un agente inteligente capaz de aprender y actuar racionalmente dentro de un entorno, buscando maximizar una función objetivo. Los resultados nos permiten obtener agentes inteligentes entrenados capaces de explotar rutas óptimas dentro de un laberinto usando Q-learning. Además obtenemos un sistema distribuido capaz de comunicarse a través de MQTT desde cualquier parte del mundo y en tiempo real.

Palabras Clave:

Agentes inteligentes, Internet de las Cosas, Ciudades Inteligentes,Q-learning, MQTT.

Abstract

Today the interaction between computer systems and people is more common and frequent worldwide. As a result, artificial intelligence (AI) techniques have become increasingly relevant and necessary in everyday life since their applications allow the automation of processes commonly done manually. Moreover, at a technological level, AI can deal with large-scale problems such as handling massive data, security problems, and human-machine interaction.

As technology advances, it is necessary to give importance to the problems present in the daily life of citizens since they affect their quality of life. Smart Cities come to solve problems of social, environmental, and mobility interests, among many others. That is why it is so relevant. Therefore, as a contribution to the growth of Smart Cities, this project develops a system that solves a common problem in urban cities, such as searching for optimal routes in cases of traffic congestion or emergencies. We use the Internet of Things (IoT) technologies to create a distributed system that allows us to establish a global network where exchanging data between devices, sensors, and humans is possible. Based on a client-server model, we seek to connect ESP8266 nodes that can communicate through a known broker called MQTT. In turn, an intelligent agent is developed capable of learning and acting rationally within an environment, seeking to maximize an objective function. The results allow us to obtain trained intelligent agents capable of exploiting optimal paths within a maze using Q-learning. In addition, we obtain a distributed system capable of communicating through MQTT from anywhere in the world and in real-time.

Keywords:

Agents, Internet of Things, Smart Cities, Q-learning, MQTT.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Information and Communication Technologies have revolutionized the digital and technological era, leaving behind the barriers of time and space to reinvent a new world where humanity's quality of life has significantly progressed due to contributions in all areas of society.

Since the appearance of a new "network" called the Internet of Things (IoT), the digital interconnection between everyday objects is now possible to improve the quality of life. Furthermore, the creation of Smart Cities allows environmental and economic sustainability to be improved, which is why the IoT has become an essential component within the technology industry.

On the other hand, artificial intelligence (AI) is a useful tool within the technological world since it can be extended to various areas that register the handling of large volumes of information, high precision, repetitive tasks, and high complexity in problem-solving, among others [7]. The current trend shows that different AI methods such as neural networks, genetic algorithms, hyperheuristics, and in general, machine learning methods have taken a large scale in the field of the Internet of Things, and have allowed achieving results in the development of Smart Cities [8][9].

Reinforcement Learning (RL) is a Machine Learning method that has revolutionized AI applications since it involves an agent that can learn from its environment to make decisions that benefit its next actions to execute. Furthermore, the RL is managed under

a reward and penalty system, allowing it to maximize an objective function [10].

Within the IoT, Reinforcement Learning has become very functional because it can replace human intervention in systems or applications that require adjusting their behavior and configuration. In this work, we have implemented a Q-learning agent applied to a maze solution, which can simulate different urban mobility environments. The agent is developed through an architecture that involves states, actions, rewards, environment, and value function. It is also based on the Markov Decision Processes (MDP) and the solution of the Bellman Equation.

1.2 Problem statement

Learning to solve maze problems is a matter of much research within artificial intelligence because it offers the possibility of many practical applications in real life, such as optimal routing for vehicles considering traffic congestion or emergencies in urban cities. This study has become a big motivation for developing this work because we can even contribute to solving social conflicts and climate change, among others.

Reinforced Learning is a field of great interest for researchers in the application of IoT technologies due to its contribution to the development of Smart Cities. IoT systems require real-time interconnection between devices, sensors, and other devices capable of exchanging information through the Internet. Moreover, on the other hand, the development of Reinforcement Learning requires an environment that allows the agent to evaluate new strategies to find an optimal policy, using high-level language but low-level algorithms.

1.3 Objectives

1.3.1 General Objective

To apply reinforcement learning agents (RL agents) to develop an IoT distributed processing system where the role of agents allows the planning of optimal routes in emergency situations in a virtual city.

1.3.2 Specific Objectives

- To build a distributed IoT system with three ESP8266 nodes, where one works as a server and the others as clients.
- To assure continuous and secure MQTT communication between nodes through unpredictable network conditions.
- To apply Q-learning to train agents and improve their ability to learn optimal policies, which allows for finding optimal routes to the target.

Chapter 2

Theoretical Framework

2.1 Internet of things (IoT)

The internet of Things term was used for the first time in 1999 by the British Kevin Ashton to refer to a system that connects physical objects to the internet through sensors. Ashton used the term IoT to show that radio frequency identification (RFID) tags could be used on corporate supplies to connect them to the internet and thus perform a counting and tracking system without human intervention [11].

Today the Internet of Things is a term used to refer to the network interconnection of everyday objects with computing capacity that extends to various devices and sensors [11][12]. More than 99% of things in the physical world are not connected to the Internet. With the Internet of Things, we can connect everything we can imagine [13].

With technological advances, we enter a new era of ubiquity where forms of communication will expand from human-human to human-human, human-thing, and thing-thing (also called M2M). A new dimension in which information and communication technologies allow connectivity from anywhere and anytime to anything and anyone [1]. Figure 1 shows a representation of the new dimension.

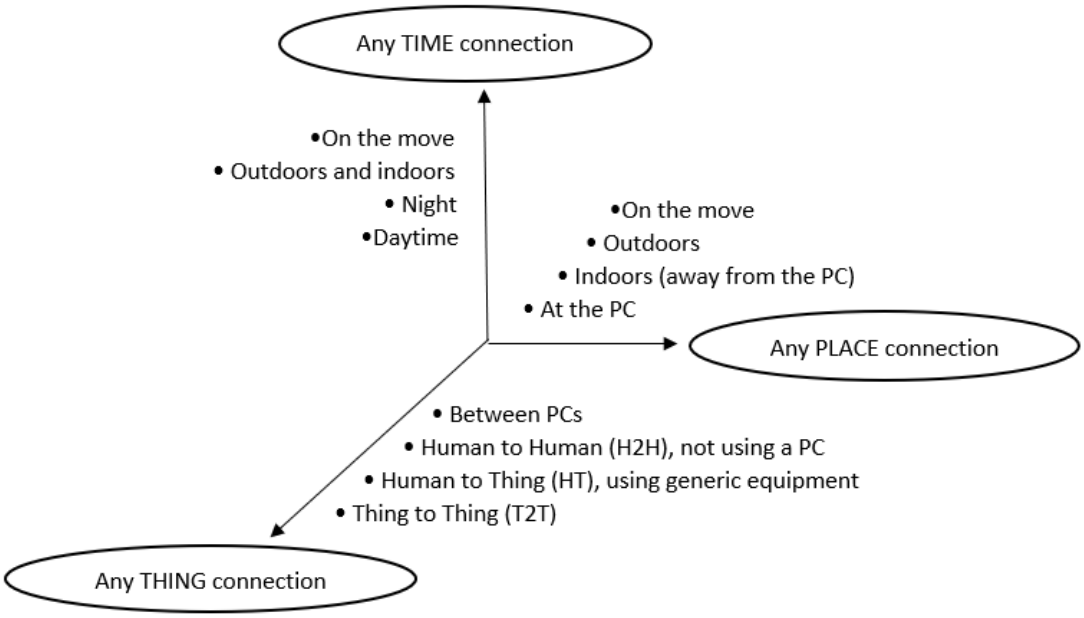


Figure 2.1: A new IoT dimension [1].

2.1.1 Technology and Platforms for the Internet of Things

IoT implementation often requires the combination of various hardware and software components to connect a product. Generally, this combination results in a stack of three main layers: The thing or device layer, the connectivity or communication layer, and the IoT cloud layer [14].

Device Layer

This layer is the specific IoT hardware, that is, any device such as sensors, actuators, or additional processors [1]. At an industrial level, IoT can encompass intelligent vehicles, robots, portable terminals, and meters, among other devices or equipment. These devices collect a large amount of data through different types of wiring such as Ethernet, fiber optics, or wireless networks such as WiFi, Bluetooth, RFID, 5G, etc. [15].

Connectivity or Communication Layer

This layer is responsible for processing and forwarding the data flow obtained from the layer of the thing or device [15]. Communication protocols such as MQTT that provide different

services such as perimeter security, privacy protection, perimeter data analysis, intelligent computing, process optimization, and real-time control are involved here [14][15].

IoT cloud layer

This layer refers to the software that manages the communication of all interconnected devices. It obtains and shares data through collaboration in the cloud, communicates, provisions, and manages it. At the same time, an application platform allows the development and execution of IoT applications. In addition, the data generated by connected things are stored, processed, and analyzed, coordinating the interaction for a specific purpose [14].

Furthermore, using platforms to create IoT applications is very frequent in IoT technologies. *Platforms* are usually defined as a group of technologies that are used as a base to develop other technologies, applications, or processes. Within the IoT, these platforms are software that offers independent functionalities to create IoT applications. Since the nature of these platforms can vary depending on different aspects of the IoT technology stack, there is no standard configuration. However, there exists a variety of IoT platforms that are tailored to specific needs and application areas [14].

Large corporations and technology companies like Intel, Cisco, Microsoft, and other hardware developers predicted the rapid growth of IoT, which is why today, there are a multitude of embedded platforms and operating systems that provide different features for IoT development. These features include processor performance, amount of RAM, and other additional features such as WiFi/Ethernet modules, input/output pins, etc. So, choosing a suitable platform has become a common problem [13].

2.1.2 Node MCU ESP8266

It is a simple but high-performance, low-cost WiFi chip that provides an Internet connection to any microcontroller via SPI/SDIO or I2C/UART communication. This WiFi module is an SoC (System on Chip) that integrates a stack of TCP/IP protocols and is programmed using AT-Command commands if it uses serial communication [13][16].

In this case, we use the nodeMCU ESP8266 since it offers a more straightforward platform, is low cost, and meets the needs that we require, which is WiFi connectivity between nodes. The device model is shown in Figure 2.2.

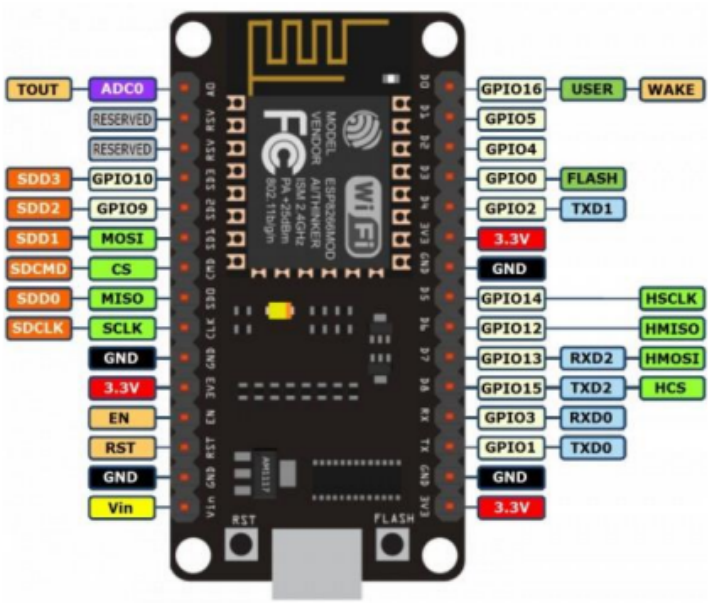


Figure 2.2: ESP8266 Node MCU Module [2].

2.1.3 Arduino IDE

Arduino is is an integrated development environment software (IDE), which allows programming applications in C++ language. This software connects to Arduino hardware to upload programs and communicate with them. It has a console that displays text output from the Arduino IDE software, including full errors and other information. There is a wide variety of boards to which it can be connected, and it is fully configurable because there are tools that facilitate adaptation for each experiment. In addition, libraries provide additional functionality to work with different hardware or manipulate data [17].

2.1.4 MQTT

The internet of things uses internet connectivity to collect and share data with other nodes (physical objects) without human intervention. In order to ensure that data exchange between all IoT nodes is secure and reliable, several messaging and communication protocols have been developed, such as the Advanced Message Queuing Protocol (AMQP), Restricted Application Protocol (CoAP), and Extensible Messaging Presence Protocol (XMPP), among many others. However, the Message Queuing Telemetry Transport (MQTT) protocol is widely used for smart homes, industrial applications, agricultural

IoT, and other applications [18]. MQTT is a lightweight machine-to-machine publish-subscribe network protocol. Among the advantages of this protocol is its design, which supports communication between devices with low bandwidth and memory resources [19].

The main components in an MQTT protocol are the broker (central device), the clients (IoT nodes), the subject, and the message. MQTT clients communicate through a central node, also called a broker, which can be local or in the cloud. The function of a remote broker is to allow IoT nodes to publish or subscribe to topics according to the functionality of the nodes [18][19]. Figure 4.2 graphically represents the operation of the MQTT protocol.

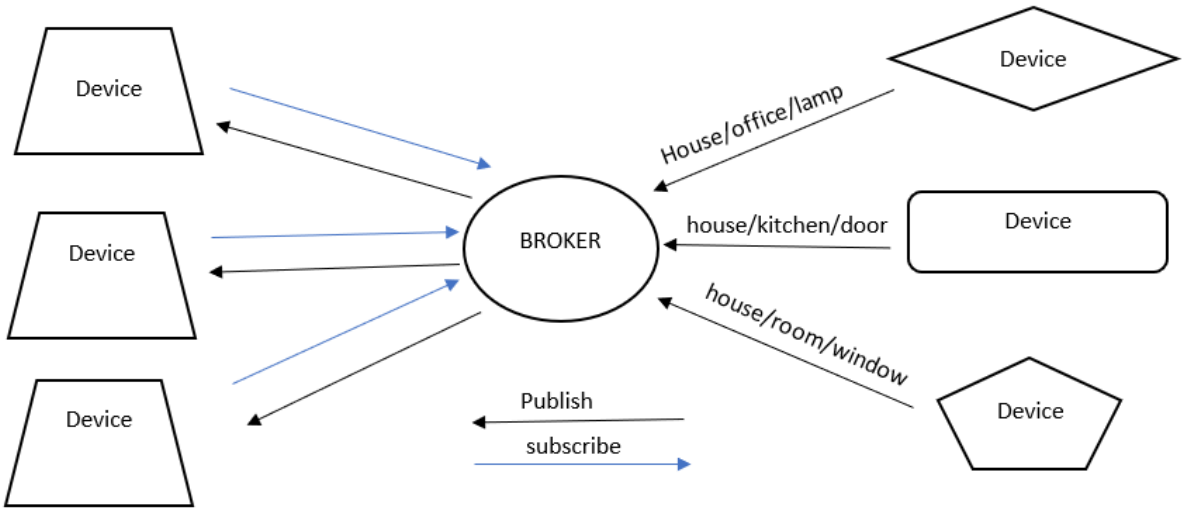


Figure 2.3: MQTT broker operation [3].

2.1.5 HiveMQ

HiveMQ is a client-based messaging platform and MQTT broker created to transfer data between various systems, IoT devices, and the cloud efficiently, quickly, and reliably. HiveMQ uses the MQTT protocol to perform real-time, two-way data transfer between your device and your company's systems.

HiveMQ is a platform that solves the technical challenges faced by various organizations in creating Internet of Things applications. One of the benefits of this platform is that it has a lower cost of operation by using the network, hardware, and cloud resources. It can create reliable and scalable applications for different requirements, has a fairly fast data delivery, and integrates IoT data in other existing systems.

HiveMQ broker instances scale with the underlying hardware [20]. It has been proven that a non-blocking and multi-threaded approach allows a simultaneous connection of up to 10 million devices [21], an essential aspect of developing IoT technologies. However, because HiveMQ and MQTT are based on a pub-sub architecture, the overall network traffic is reduced, and the MQTT message size is significantly smaller than the regular HTTP protocol, so the data is transferred through the network is reduced [21].

In this work, we will use some of the elements explained above since they are accessible technologies and have characteristics that facilitate the implementation of this project, which will be explained further in the following sections.

2.2 Smart Cities

As urbanization grows, new challenges and problems arise with them. Initiatives for smart cities, which offer opportunities, solve urban issues and provide citizens with a better living environment, are becoming increasingly frequent. The Telephone Foundation, in its book, shows that the urban population had surpassed the world's rural population in 2007 [22].In the article "A literature survey on smart cities," the authors explain that by 2014, the level of urbanization had reached 54%, and according to the United Nations, they deduce that in 2050 the urban population will get 66% [23].

Cities significantly impact a nation's economic and social development since people work and live there, companies carry out their activities and provide different services. However, cities are also large consumption centers, spending 75% of global energy and generating 80% of the gases responsible for the greenhouse effect [22].

Currently, developing countries are urbanizing more rapidly. A clear example is China, which in the last ten years, has managed to increase its population from 41% to 54%.This country desperately faces problems of overcrowding, environmental degradation, air and water pollution, contagious diseases, and crime. Mexico faces an increase in gases that affects air quality. The US faces high traffic congestion on its streets, causing high fuel consumption and, therefore, pollution [23].

These scenarios show that population growth magnifies new economic, environmental, and social demands on large cities. In contrast, citizens and governments demand increased

efficiency, sustainable development, resource management, quality of life, and solutions to reduce urban problems. [22] [23].

Smart Cities encompass several aspects of a city's proper functioning and management. The most common domains of a Smart City are governance, economy, management, infrastructure, technology, and people. Figure 2.3 summarizes the six main dimensions of a Smart city and its possible applications [4].

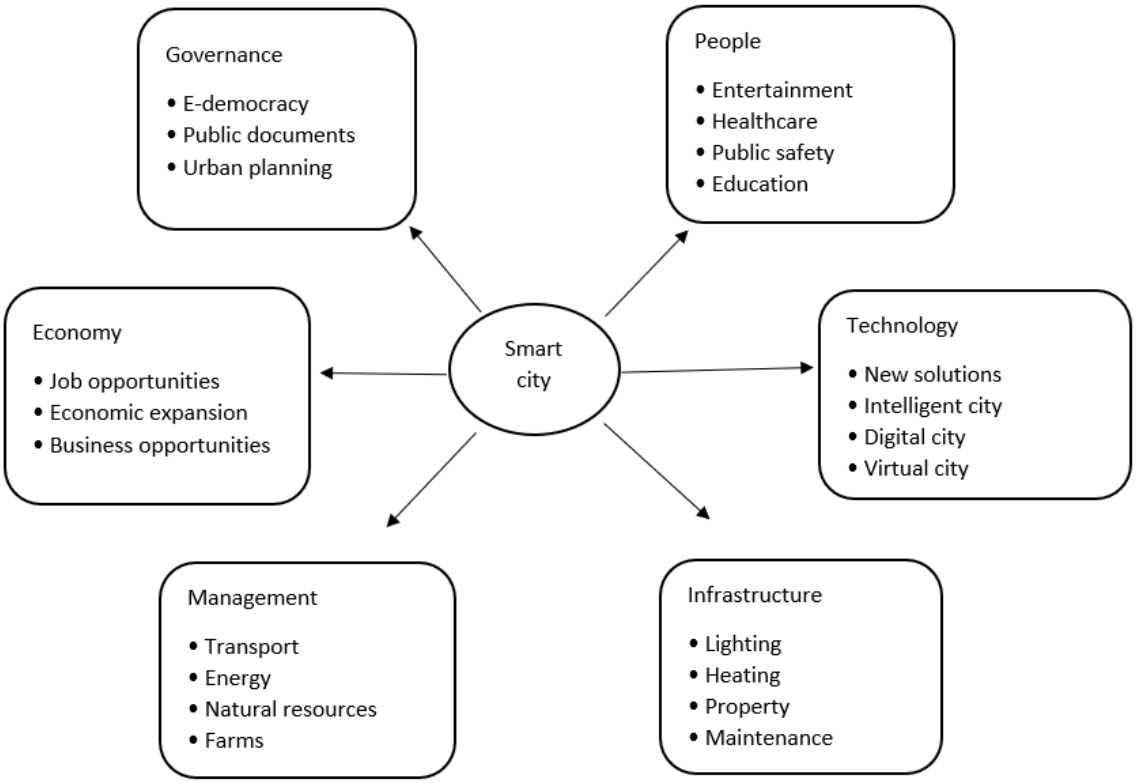


Figure 2.4: Smart city dimensions [4].

Identifying the domain area where wireless technologies are applied is essential since this may vary depending on the environmental conditions and the different parameters involved.

This project focuses mainly on the Smart City Management domain, one of the most developed factors among smart cities, and primarily covers transportation, energy, and natural resource management. Implementing IoT technologies in this domain solves the problems of data collection, resource management, and traffic allocation problems. Some initiatives currently benefiting from IoT solutions are public transport, agriculture, crop

management, water flow measurement, municipal and domestic purposes, public lighting, renewable energy, and city logistics [4].

2.3 Alternate emergency routes

2.3.1 Emergency vehicle routing

Currently, urban traffic congestion is a problem with a significant impact on the quality of life since it directly or indirectly affects public services. A study by the US Department of Transportation talks about the impact of urban traffic congestion and the importance of alternative routes. Traffic congestion is equivalent to economic losses for companies and decreased performance [24]. The causes of traffic congestion depend on traffic density and events that occur after disasters, such as traffic accidents, construction sites, fires, earthquakes, etc. [25]. In general, these events fall into two groups: planned and unplanned events.

Planned events refer to closures of road sections and construction works. In contrast, unplanned events occur when there are emergencies, adverse weather, traffic incidents, and natural disasters [24]. Emergency service vehicles, such as ambulances, firefighters, and police, are required in these situations. Respond to emergencies on time; for example, an ambulance must reach the hospital within 60 minutes to guarantee the victim's survival. According to the Theory of the Golden Hour, that is the time when there are more chances of survival [26]. Therefore, the expected result caused by planned and unplanned events is a high demand to reduce emergency vehicle travel delays to create efficient vehicle routing in a smart city [24].

Advances in IoT technologies coupled with data fusion techniques hold promise for handling the problem of road traffic congestion. Sensor networks are useful for real-time data acquisition in various decision-making applications; in this case, they help optimize emergency routing by implementing alternative routes.

2.4 Distributed processing system

Distributed computer systems today are becoming more frequent in the technological field because they allow to carry out processing activities more efficiently, improving performance and allowing the distribution of resources. This computer model allows configuring several central processing units (CPU) to solve computational problems massively, bringing out distributed processing. This system is based on distributing information through the Internet to different computers so that, once the problem is solved, they return the result to the server.

A distributed processing system introduces various models to ensure that the systems can solve real-world problems, that is, potential threats or challenging circumstances. We describe the Client-Server architectural model, which we applied in the project.

2.4.1 Client-Server Model

Client-server architecture employs a distributed application model that uses two main components: information providers called servers and requestors called clients. This system performs client and server functions to promote the exchange of information between them. Users have the same accessibility to data simultaneously and very often communicate over a computer network on different hardware. However, both components can reside in the same system [27].

Client-server communication begins when the client requests data from the server, and the server responds by sending requested data to the client in a request-response messaging pattern [5]. Some of the benefits of this model include that it allows for easier sharing of client resources to servers, it reduces data replication by storing data on each server rather than the client, and the server can receive requests from many clients in a short period. Furthermore, within Cloud computing, the primary importance lies in the scalability of millions of virtual machines [27]. Figure 2.4 shows the communication model of client and server network systems.

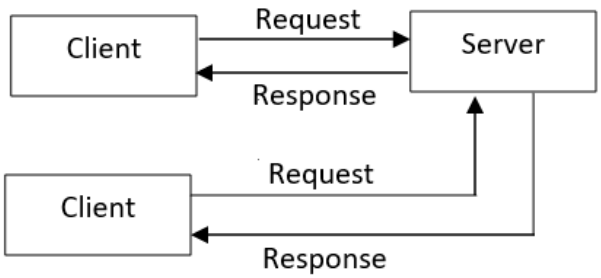


Figure 2.5: Client-server network block diagram [5]

Chapter 3

State of the Art

3.1 Reinforcement learning

Reinforcement learning (RL) is a sub-areas of Machine Learning that studies systems where an agent interacts in a dynamic environment to determine which actions can maximize its reward or prize pool [28]. The agent initially has a specific state belonging to the state space, a limited set of possible legal states, then chooses an action within the possible legal actions, also called an action set, to receive a reward that depends on the selected action and the current state [28][29]. The environment determines the state space and set of actions. RL systems can be formulated using Markov decision processes, which will be explained in the next section.

Reinforcement learning is strongly inspired by behavioral psychology, based on a scheme of ”rewards and punishments,” as the example of the classical conditioning case, where Pavlov's dog salivated every time the bell rang since it learned to associate the sound of the bell with the reception of food [30].

RL has two different approaches: model-based algorithms that use a model to build the environment and form a control policy based on the learned model, and model-free algorithms that directly update the function of value, that is, the expected reward from the current state or policy [31].

The difference from other types of learning is that RL is a systematic assessment that occurs at the same time as learning. It does not have the ”correct” pairs of inputs or outputs. Instead, the agent detects different aspects of the environment and collects valuable

experience in states, actions, transitions, and rewards to operate optimally [6].

In general, the reinforcement learning objective is to identify the actions chosen in the different states to maximize the reward. The agent must find and learn an optimal policy that tells each state what action to take [32].

We can see many applications of RL in several other areas, such as robotics, automatic cars, health informatics, complex games [28], and other disciplines such as statistics, genetics, and operations research, among others [33].

3.1.1 Exploration and exploitation

Another important aspect is the balance between exploration and exploitation. To get a good reward, the agent prefers to follow specific actions but to know which actions to choose, the agent has to do some exploration first. It generally depends on how long the agent is expected to interact with the environment.

3.2 Markov Decision Processes (MDP)

Markov decision processes are defined as stochastic sequential decision processes, and they are based on notions of the current state, which describe the agent's situation (action or decision), which affect the dynamics of the process and the reward, which is present for each agent transition between states [34].

Because action effects are stochastic, and the selected action can result in different possible states in the subsequent decision stage, the optimal control strategy cannot necessarily be represented as a single sequence of actions [35]. Consequently, the solutions within an MDP are generally given as universal policies (rules or action strategies) that indicate what action to take at each step of the decision process and for each possible state reached by the agent. Due to the uncertainty in the results of the actions, the application of a given policy may result in different sequences of states/actions [34] [35].

In a standard RL, an agent is connected to an environment through perception and action (Figure 3.1). In each interaction the agent receives as input an indication of its current state ($s \in S$) and selects an action ($a \in A$). The action changes the state and the agent receives a reinforcement or reward signal ($r \in R$) [6].

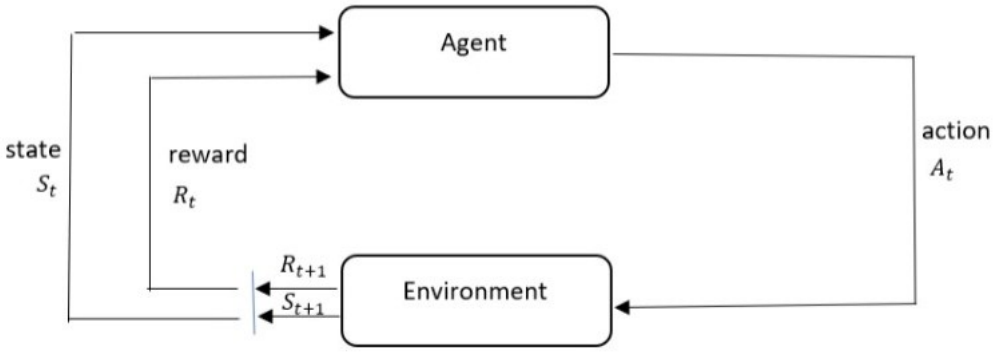


Figure 3.1: Reinforcement Learning [6]

Formally, a MDP is a tuple $M =< S, A, \phi, R >$. The elements of an MDP are:

- A finite set of states $S(1, ..., n)$.
- A finite set of actions A, which can depend on each state.
- Reward function (R): defines the goal. It maps each state–action to a number (reward), indicating the desirable status.
- Environment model (optional): mimics the behavior of the environment. It can be used for planning by considering possible future situations based on the model. $\phi : A \times S \rightarrow \Pi(S)$ is a state transition function given as a probability distribution. The probability of reaching the state $s' \in S$ by performing the action $a \in A$ in the state $s \in S$, which can be denote as $\phi(a, s, s')$.
- Policy (π): defines how the system behaves at a given time. It is a (sometimes stochastic) mapping of states to actions.
- Value function (V): indicates what is good in the long run. It is the total reward an agent can expect to accumulate starting in that state (reward predictions). Actions are sought that give the highest values, not the highest reward. The rewards are given by the environment, but the values must be to estimate (learn) based on observations. Reinforcement learning learns value functions while interacting with the environment.

There are several ways to apply this learning process. Here we will focus on what is known as Q-learning, which for any Markov Decision Process can determine an optimal action-selection norm given infinite exploration time and partial random criteria.

3.2.1 Reward Models

Given a state $s_t \in S$ and an action $a_t \in A(s_t)$, the agent receives a reward r_{t+1} and moves to a new state s_{t+1}. The mapping of states to probabilities of selecting a particular action defines its policy (π_t). Reinforcement learning can specify policy changes as a result of its experience.

As such, this model does not try to maximize the immediate reward but rather the long-term (cumulative) reward, which is calculated by the environment. If the reward received after time t is denoted as $r_{t_1}, r_{t_2}, r_{t_3}, ...$, what we want is to maximize what we expect to receive as a reward (R_t), which is in the simplest case:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{3.1}$$

If the tasks have an endpoint, they are called episodes. On the other hand, if the tasks are infinite, they are called continuous tasks. In this case, equation 3.1 is inappropriate since the calculation cannot be performed when T has no limit. Therefore, it uses an alternative equation (3.2) where the contributions of the more distant rewards are reduced [32].

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.2}$$

where γ is known as the discount ratio and is between $0 \leq \gamma < 1$. When $\gamma = 0$, the total reward is maximized only considering the immediate rewards.

Likewise, depending on the objectives, we consider the following models:

1. Finite horizon: the agent tries to optimize its expected reward in the following h steps without worrying about future consequences.

$$E(\sum_{t=0}^{h} r_t) \tag{3.3}$$

   where r_t means the reward received t steps in the future.

This model is applied in two ways: (i) receding-horizon control, where the next h steps always are taken, and (ii) non-stationary policy, where the following h steps are taken in the first step, in the following the $h-1$, and so on until finished. The main problem is that it is not always known how many steps to examine.

2. Infinite horizon: the rewards received by the agent are geometrically reduced according to a discount factor $\gamma(0 \leq \gamma \leq 1)$:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{3.4}$$

3. Average reward: long-term optimization of the average reward:

$$lim_{h\to\infty} E(\frac{1}{h}\sum_{t=0}^{h} r_t) \tag{3.5}$$

The problem with this model is there is not always a way to recognize policies that are highly rewarded from those that are not.

In general, the agent's actions not only determine the immediate reward but also define the next state of the environment (at least probabilistically). Reinforcement learning satisfies the Markovian property and the transition probabilities, which are given by:

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{3.6}$$

Then, the expected reward value is:

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{3.7}$$

The objective is to estimate the value functions, that is, to analyze how good it is to be in a state or perform an action in terms of future rewards and expected rewards.

3.3 Agent

Reinforcement learning seeks that the agent learns to make decisions optimally. The reward and discount function allows us to generate a measurable benchmark that helps us compare different decision-making processes and identify the most optimal option. Under these two

concepts, we can define the agent's goal in a reinforcement learning problem as finding a series of actions that maximize the return.

3.3.1 Policy

The policy π is a mapping of each state $s \in S$ and action $a \in A(s)$ to the probability $\pi(s, a)$ of taking action a while in state s. The value of a state s under policy π, denoted as $V\pi(s)$, is the expected reinforcement in state s and following policy π. The agent receives a state as input, and the policy allows it to determine what is the most appropriate action to take at a given instant time.

Deterministic politics occurs when there is no random component to the actions within the Markov decision process. This policy is defined as $a = \pi(s)$: given a state s, the policy will tell us what action to execute with absolute certainty (because it is deterministic). However, the implementation of deterministic policies is not typical because there is often a random component to the Markov decision-making process in real-life situations.

Furthermore, since we have a completely random environment, the resulting policy will not indicate the following action the agent should take with absolute certainty. Instead, it tells us the probability of a particular action. Therefore, the conditional probability notation is used to represent this policy mathematically:

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{3.8}$$

Where the policy $\pi(a|s)$ is the probability of executing at the instant of time t the action a being in the state s. This value (the probability) will be in the range from 0 to 1.

The policy does not specify the value of the estate nor the actions taken to assess its benefits. Therefore, different functions are defined according to the state and the action that make it possible to perform this task.

3.3.2 State-value function

In reinforcement learning, the policy needs functions that determine the goodness of a particular state or action. When the policy is stochastic, the probability distribution is given for each case of possible actions.

The state value function defines a method of determining the advantages or disadvantages of a particular policy. The reward is the process of measuring how good or bad an agent's decisions are. In order to quantify the goodness of a state, we use the sum of the rewards (total reward) received by the agent for executing a particular trajectory.

To build the state-value function, we need to consider the following aspects:

- Since the environment is stochastic, the paths under a policy are not unique since actions are taken according to a probability distribution. Therefore, when calculating the reward, we must consider all possible ways the environment might react to the policy. That is, the expected reward will give us the result of the average of all the potential rewards that will be obtained after analyzing all the possible trajectories.
- The agent will not take an arbitrary trajectory; the policy will define the course.

Then, given the definition of the expected reward, the **value of a state** s is defined as the expected reward if the agent were to start from state s and follow the path defined by the policy π. Therefore, When we compute the value of each state for all possible states, we get precisely the state value function.

The **state-value function** will give us the expected reward following the policy π, for each state that is part of our environment. This function allows us to quantify each state precisely and thus determine how good or bad a particular state may turn out in the decision-making process. The equation 3.9 expresses this function mathematically:

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \tag{3.9}$$

for all states s in S.

3.3.3 Action-value function

Similarly, as the state-value function, the action-value function, called the "Q function," uses reward as its basis. This function calculates the expected reward that the agent would receive by taking action a while in state s and following policy π. The following equation

expresses this function mathematically:

$$Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \tag{3.10}$$

The Bellman equations allow us to express the state-value and action-value functions in terms of policies and environment variables, which is necessary to solve using different algorithms. Here is the importance of these equations.

3.4 Bellman equation

The Bellman equation, named after its discoverer Richard E. Bellman, is a necessary condition for achieving optimization associated with a mathematical optimization method known as dynamic programming. Dynamic programming aims to simplify more complex problems or tasks by breaking them down into smaller problems and recursively solving them to attack the larger problem.

Generally, the Bellman equation refers to the dynamic programming equation that allows for solving discrete-time optimization problems. However, in continuous-time optimization problems, the analogous equation is the well-known Hamilton-Jacobi-Bellman equation, a partial differential equation.

The Bellman equation writes, at a given point in time, the value of the decision problem in terms of the payoff returned by some initial solutions and the value of the remaining decision problem as a result of the initial solutions. That decomposes the dynamic optimization problem into simpler sub-problems, as described in the Bellman optimization principle.

In discrete time, almost any multistage optimization problem that can be solved using optimal control theory also may be solved by analyzing the appropriate Bellman equation. That is achieved by introducing new state variables to the problem (increasing state) [36]. However, if the cost function of the multistage optimization problem satisfies a "backward separable" structure, then the appropriate Bellman equation can be found without increasing the state [37].

The Bellman equations are an alternative way to mathematically represent state-value and action-value functions in terms of policy and environment variables, which serve as a

starting point for building reinforcement learning-based algorithms.

3.4.1 Bellman's equation for state-value function (V)

In section 3.2.1 the discounted return function at time t had been defined as:

$$G_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} = r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+3} + ... + \gamma^{T-1} r_T \tag{3.11}$$

Equation 3.11 can also be written recursively:

$$G_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + ... + \gamma^{T-2} r_T), G_t = r_{t+1} + \gamma(G_{t+1}) \tag{3.12}$$

The objective of defining the discounted return helps us to also determine the Bellman equation for the function V, which states that the value of the state can be acquired as the sum of the immediate reward and the discounted value of the next state. Therefore, this function is defined as:

$$V(s_t) = r_{t+1} + \gamma \cdot V(s_{t+1}) \tag{3.13}$$

The above equation can be reformulated in terms of MDPs using the reward function. Equation 3.14 reflects which action a is performed by the agent to generate the reward r_{t+1} and under which policy.

$$V_\pi(s) = R(s, a, s') + \gamma \cdot V + \pi(s') \tag{3.14}$$

Under a stochastic environment, when we perform an action a on the state s, it cannot be guaranteed that the next state s' will always be the same; the states may vary. Therefore, we contemplate this stochasticity in the following equation:

$$V_\pi(s) = \sum_{s'} P(s'|s, a) \cdot [R(s, a, s') + \gamma \cdot V_\pi(s')] \tag{3.15}$$

Where $P(s', s, a)$ performs the probability of reaching state s' by performing action a in state s, a sum of the estimation is made from each potential next state multiplied by the probability of transition to this state.

On the other hand, when there is a stochastic policy, instead of always performing the same action on a state, we select an action based on the probability distribution over the

action space. So the Bellman equation for the function V is:

$$V_\pi(s) = \sum_a \pi(a|s) \cdot \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot V_\pi(s')] \tag{3.16}$$

or,

$$V_\pi(s) = \mathbb{E}_{a\sim\pi, s'\sim p} R(s,a,s') + \gamma \cdot V_\pi(s') \tag{3.17}$$

In this way, we find the solution to the problem by dividing it into simpler recursive subproblems.

3.4.2 Bellman's equation for action-value function (Q)

Similarly, like the Bellman function for the V function, the Bellman function for the Q function can be obtained as a sum of the immediate reward and the following state V function:

$$Q_\pi(s,a) = R(s,a,s') + \gamma \cdot V_\pi(s') \tag{3.18}$$

In a deterministic environment, the Bellman equation for the function Q is developed using mathematical expectation to show the transition probability corresponding to the next state:

$$Q_\pi(s,a) = \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot V_\pi(s')] \tag{3.19}$$

Nevertheless, we are interested in defining the value $Q_\pi(s,a)$ as a function of the value Q of the following state-action pair $Q_\pi(s',a')$ to obtain the Bellman equation equivalent to the value function of the state. Then, a possible term is added, taking into account the stochasticity of the policy, which determines the choice of the following action a':

$$\sum \pi(a'|s') \tag{3.20}$$

The resulting equation is the Bellman expectation equation of the action value function or Q function:

$$Q_\pi(s,a) = \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot \sum_{a'} \pi(a',s') \cdot Q_\pi(s',a')] \tag{3.21}$$

3.4.3 Optimal bellman equation

As mentioned before, the goal of the Markov process is to find a policy that defines the agent's behavior and maximizes the reward in the long run. An optimal policy π^* is deduced when our state-value and action-value functions obtain the highest possible values [38].

We want to find the maximum values for these functions so that for the function V (equation 3.16), we will obtain a $V_{\pi^*}(s) \leq V_\pi(s)$ for all states s, then we have:

$$V^*(s) = \max_\pi V_\pi(s) \tag{3.22}$$

The optimal bellman equation is calculated by choosing the action that gives the maximum value, i.e., we calculate the state value using all possible actions and then select the maximum value for the state value. Therefore the optimal Bellman equation for the function V is expressed as:

$$V_*(s) = \max_a \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot V_*(s')] \tag{3.23}$$

On the other hand, the maximum for the function Q is represented as:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a) \tag{3.24}$$

And, like the Bellman equation for the optimal function V, all possible actions in state s' are chosen, and the optimal function for the function Q is calculated (equation 3.21), which is defined as:

$$Q_*(s,a) = \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot \max_{a'} Q_*(s',a')] \tag{3.25}$$

Finally, it turns out that the optimal value of a state V*(s) is equal to the best action-value function that we can obtain from this state, that is:

$$V_*(s) = \max_a Q_*(s,a) \tag{3.26}$$

Equation 3.26 allows us to find the optimal policy if we know the optimal values of the function Q. The best association of the action to each state will be the action that maximizes

the value of this function:

$$\pi_*(s) = \underset{a}{\text{argmax}} Q_*(s, a) \tag{3.27}$$

Therefore, the optimal policy is given by the optimal functions $V_*(s)$ and $Q_*(s, a)$. So, we can solve the problem by knowing the values of the optimal functions.

3.5 Q-learning

In machine learning, Q-learning is a type of Reinforced Learning without a model. It can also be defined as the asynchronous dynamic programming (DP) method in the sense that it updates a single entry each step [39].For any Markovian domain, Q-learning allows agents to gain the ability to learn to act optimally by experiencing the consequences of actions without having to build domain maps [40][41].

The Q-learning model finds the best course of action from its current state by creating its own rules or policy. That is considered out of policy because the q-learning function learns from actions out of current policy, taking random actions, so no policy is needed. This value-based model optimizes the value function based on the environment or problem.

Q-learning is an incremental algorithm that builds a function called "Q-function" that estimates the discounted cumulative reinforcement (Q-value) for each possible state/action pair. The learning agent executes an action based on these Q-values at each time step. Then the Q-value of the pair of the current state and the selected action is updated according to the immediate reward and the evaluation of the next state caused by the action [42].

The one-step equation Q-update equation expresses this:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{3.28}$$

Where Q is the value resulting from performing action a in state s; s is the state vector; a is the action vector; R is the reward, and γ is the discount factor. The discount factor makes rewards earned earlier more valuable than rewards received later [32].

The advantage of the Q-learning algorithm learns the values of all actions rather than

simply finding the optimal policy. Q-learning is not sensitive to discovery, any action can be carried out at any time, and the information is obtained from this experience [32].

Furthermore, such a feature allows Q-learning to learn from other drivers, providing valuable data, even if they aim to complete a different task. This way, knowledge of several Q-learners is obtained and combined to avoid non-optimal actions and find a compromise action [32].

3.5.1 Maze problem

The maze problem is a widespread application for Reinforcement Learning, generally used to test the effectiveness of different RL algorithms. The maze problem can be summarized in a matrix game area of size $n * m$ (where $n, m > 0$) in which the agent's goal is to reach the destination position, starting from a starting point and dodging certain obstacles. The purpose is to find the shortest path (of the many options) from the starting point to the objective position [43]. Mazes can keep the locations of the start position, finish position, and obstacles variable. Figure 3.2 shows an example of a labyrinth with a particular configuration in the maze problem game area. A smaller cyan square indicates the start position of the agent, a yellow square indicates the destination position, blue squares indicate blocks, and red squares indicate the clear path.

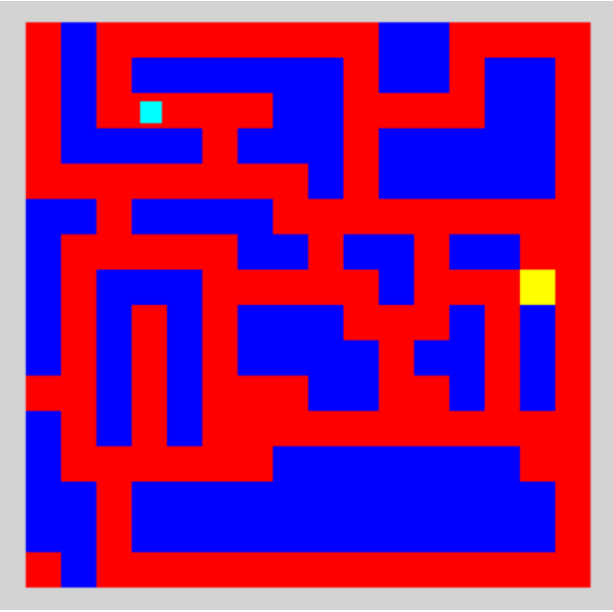


Figure 3.2: Labyrinth 15x15 size play area.

Chapter 4

Methodology

This chapter presents the processes carried out to achieve the objectives defined in this project. It also describes each implementation step for an agent to learn to find a target in a maze.

4.1 Phases of Problem Solving

- First, we need to implement continuous communication in a distributed system of nodes MCU ESP8266 through the MQTT broker, in which one chip acts as a server and the others as clients. Each client will be an agent.
- Next, it is necessary to implement the environment and the rules based on the solution of a classic maze.
- The last step is to use Q-learning to train the agents using the exploration and finally manage to exploit a solution to the maze problem.

4.1.1 Description of the Problem

Recent advances in Artificial Intelligence and Machine Learning techniques have enabled extracting features with enormous potential to efficiently manage the automated operation of IoT nodes deployed in smart cities [44]. Supervised learning, reinforcement learning, and unsupervised learning algorithms have been the most used within IoT applications. Currently, these methods have a variety of applications to solve a wide range of problems, such as intelligent traffic monitoring, where the system helps the driver to choose the most

efficient route. This system obtains traffic information, including road traffic conditions and congested locations, by tracking the location information of a large number of vehicles [45]. This is the type of problem treated in this work, where an intelligent agent has to learn to search for an efficient route under different conditions that may arise in the real world, such as traffic congestion or unforeseen disasters.

The first problem is to create an environment that can mimic a miniature real-life traffic scenario. In a maze, the free paths could simulate the streets of a city; the blocks could simulate the houses, buildings, and obstacles; and the goal could simulate the direction we want to go.

The second problem is creating a reinforcement learning agents that uses this information and learns an optimal policy to find efficient routes to reach the goal. In order to create an optimal policy, the agent must recognize the logic of a maze solution and use this information in its decision-making structure. The agent is trained and gains the ability to link present states with future states, which is the main objective of reinforcement learning. The last problem is to create a distributed system of nodes in which a server sends map information and coordinates to clients so that they can find an optimal route to the target.

4.1.2 Analysis of the Problem

To solve the first and second problems, we use the Arduino IDE platform, an integrated development environment that supports the C and C++ languages using special code structuring rules. This free software is used to create applications for Arduino boards or microcontrollers. Different modules and functions are developed in C++ to implement the Reinforcement learning agents and their environment. Then, the HiveMQ broker is used to solve the third problem, which communicates to our agents. These points make up the main contributions of this thesis and will be explained in detail in section 4.1.4.

4.1.3 Implementation

This section describes the planning and execution of the code of the model proposed in this work. The implementation is based on C++ programming language, a high-level programming language with outstanding performance and efficiency when creating appli-

cations that involve artificial intelligence. In addition, it provides extensive libraries that help create intelligent agents in similar projects.

Materials

The materials used, the configuration, and the implementation of the system are explained below.

Requirement for setup:

1. 3 Node MCU ESP8266 (one server and two clients).
2. Arduino IDE.
3. MQTT.
4. Laptops with internet connection.
5. Wires.

The first step to implement this work is to connect 3 Nodes MCU ESP8266 through the MQTT broker. One node will be the server, and the other will be the clients. All these nodes must be connected to a system (computer) with the Arduino IDE integrated development environment to be programmed.

We will program the nodes to fulfill two different types of tasks:

- **Server.** Provides a maze with paths, blocks, and objectives.
- **Clients/Agents.** They use Q-learning to learn to find optimal routes to the target. They explore the environment for many episodes until they are trained and ready to exploit solutions in a maze. Once the agents are ready, it notifies the server and sends the coordinates of the routes in real-time.

With this system, we carry out a simulation of a smart city, where emergencies occur, and the need arises to find short or faster paths to reach a place.

Figure 4.1 represents the implementation that we want to achieve with this work.

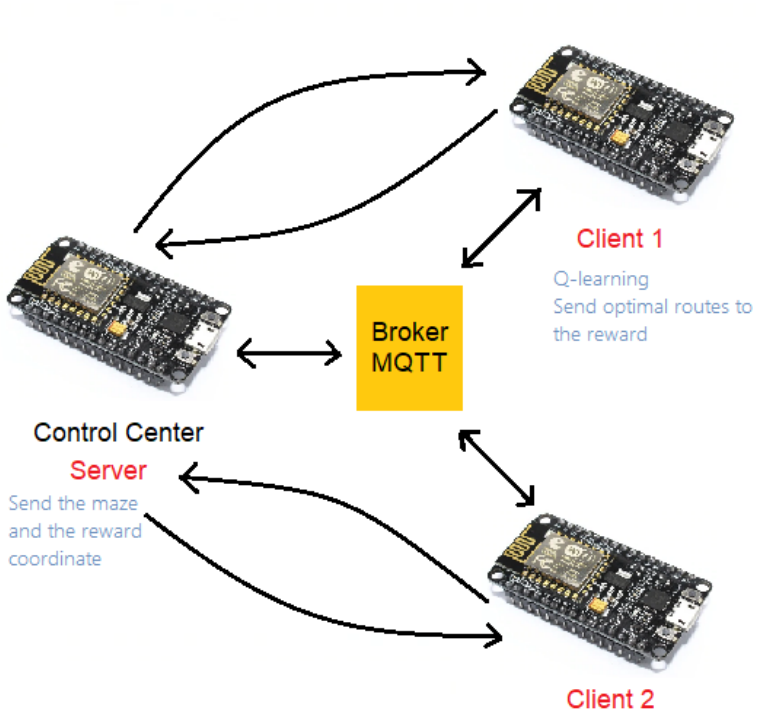


Figure 4.1: Implementation Scheme.

Below we detail the modules to use in the implementation.

MQTT communication modules.

- ***Send_data_MQTT.h***

  Module that contains functions on both the server and the client to establish communication through the MQTT broker.

  - **setup()** Module that sets main components to establish MQTT communication, such as client, MQTT server, port, and callback.
  - **setup_wifi()** Initialize the local WiFi connection by setting the WiFi SSID, WiFi password, and local WiFi IP.
  - **reconnect()** Module that contains functions to achieve the MQTT connection. Here the different users subscribe to an established *intopic*, the topic where the receivers subscribe to receive the message.

- **callback()** Module that contains functions responsible for receiving incoming messages.
- **loop()** Function that reconnects the client in case of communication interruption.

Agents modules.

- ***Agent_in_maze.h***

  Main module where the whole program is carried out. The maze and other modules are initialized.

  - **loop()** Module that loops the Q_explore module, and prints the Q and R matrix.
  - **setup()** In Arduino IDE, one task of this module is to set the data rate in bits per second for serial data transmission and initializes the loop module.

- ***Q_structures_lib.h***

  Module that contains the main functions to carry out Q-learning reinforcement learning, such as the calculation of the transition rule, the search for the maximum, and the output of the Q table.

  - **print_Q()** is the function that prints the matrix Q to the console for each episode, equivalent to one training session.
  - **print_R()** is the function that prints the matrix R, which represents a maze.
  - **move_left(), move_right(), move_up(), move_down()** are functions that generate the agent's movement according to the labyrinth.
  - **Q_explore()** is the function that will allow our agent to learn through experience. The agent will explore from state to state until reaching the goal. Each time the agent arrives at the goal state, the program goes to the next episode. This function finds the value of the transition rule, where a value assigned to a specific element of matrix Q is equal to the sum of the corresponding value matrix R and the learning parameter gamma, multiplied by the maximum value of Q for all possible actions in the next state.

  - **search_for_Max()** is the function that looks for the maximum value within the gradient produced in the matrix Q.
  - **Q_player()** Function that allows the agent to exploit the knowledge obtained after the exploration stage. The agent places itself in a random position and uses the *get_gradient* function to move in the final Q-matrix until it reaches the maximum reward.

- ***Manage_data().h***
  Module that handles the data.
  - **receive_data()** Function that receive incoming data from the server.
  - **print_route()** Function that prints the solution of a maze in the form of "*" (accessible paths), "-" (obstacles), and "@" (reward).
  - **print_route_mqtt()** Function that posts the maze solutions in the MQTT topic.

- ***Read_Write_Q_matrix.h***
  Module that consists of saving and loading the Q-table in the EEPROM memory of the ESP8266.
  - **save_Q()** is a function that uses the EEPROM.h library that allows us to use the microcontroller memory of the Arduino and Genuino AVR-based board to save values maintained when the board is turned off. In addition, this library allows us to read and write those bytes like a small hard drive. In this case, we use the $EEPROM.put$ function.
  - **load_Q()** is a function that allows loading the saved values in the board memory using the same EEPROM library used for saving, but in this case, the $EEPROM.get$ function is used.

4.1.4 Testing

For the agent's tests, it will first be trained by going through the accessible paths randomly, starting from different positions within the maze until the objective is found. After

several episodes of his training, the agent passes to the exploitation stage of the obtained knowledge, where the agent is expected to find the most optimal routes to the objective. Finally, the agent returns the shortest path from a given coordinate to a target point at the end of the process.

4.2 Model Proposal

In the present work, we propose to use the Q-learning algorithm as the primary reinforcement learning method to develop our agent. This model consists mainly of looking for the computer numerical solution of the Bellman equation. This Bellman equation guarantees a maximal value obtained in controlling a finite sequence of events that occur in a complex environment with an underlying logic context, with rewards scattered in the space time. The problem's numerical solution requires an explorer agent with its own memory elements that are used to create internal representations (Q-matrix or Q-table) and to remember the exploration paths that lead toward optimal rewards (policy). This section explains in detail how the model we propose is established.

4.2.1 Environment

The agent in our model has to learn to find an optimal route to reward. Therefore, the first step in designing our model is to create an environment that simulates a maze with its paths, walls, and a goal. In this work, the environment is represented in a 16x16 matrix called the "R-matrix," where its structures are randomly represented by 0 (accessible paths), -1 (walls or obstacles), and 100 (reward). Figure 4.3 shows the matrix used to represent the maze in this work.

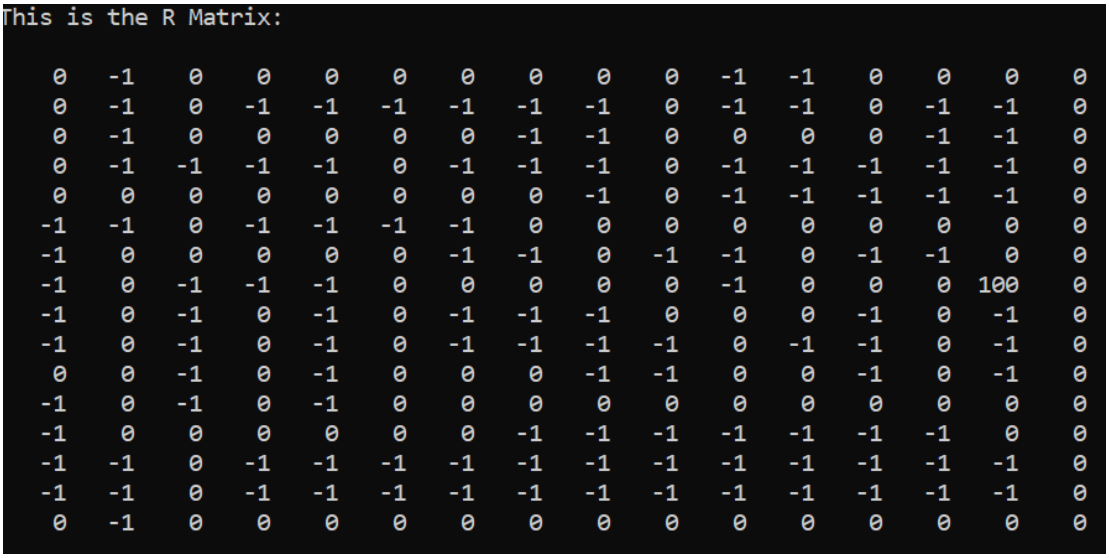


Figure 4.2: R-matrix

4.2.2 Q-learning

The Q-Learning algorithm can be summarized in the following pseudocode:

Algorithm 1: Q-learning to solve maze problem

Input: Q-table

Output: Maze solution

1 Initialize the Q-table for every state s, action a pair;

2 **while** *a completion episode has not been reached* **do**

3 Observe the state s.

4 Choose an action a, based on the values of the Q-Table, and execute it in the environment.

5 Observe the next state s' and its reward r.

6 Apply the Q-Learning formula updating the value $Q(s, a)$ of the Q-Table, taking into account the next observed state s' and the reward obtained r.

7 **end**

Initialization

To start the Q-learning process, we need to initialize the data table, the Q matrix, also defined as "Q-table." The Q-table is initialized depending on which action selection policy we will use. At startup, the agent has no knowledge. Therefore, we initialize all the values

in the Q-table to 0s, as shown in Figure 4.4.

```
This is the Q Matrix:

   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

Figure 4.3: Initial Q-matrix

The agent observes its current state and assesses all possible actions from that state. The agent then selects a specific action and executes it depending on the values stored in the Q-table. This action modifies the current state, and a reward is obtained based on the new state.

With the new data obtained from the previous state, action executed, and reward, the value of the Q-learning formula is calculated. The cell corresponding to the previous state and the action executed in Q-matrix is updated. The formula of the Q-learning transition rule increases or decreases the current value in the Q-table according to the reward obtained, whether positive or negative, creating a gradient.

Once the Q-Table is updated, the agent's current state is observed again, and a new action is then selected until a final state is reached.

To safely approximate the value of Q, numerous iterations are performed using only a sequence of immediate rewards propagating over time. In this way, we achieve that the agent learns an optimal policy from several executions where there is a trial-error approach.

Exploration or Exploitation

The agent, in this case, needs a policy that allows it to select the actions to execute in a given state. Since this work is based on reinforcement learning, we must find a balance

between exploration and exploitation. Therefore, the agent must try to execute different actions and progressively favor those that seem to be better. The mechanism used for stock selection is the greedy policy, which helps us to maximize our Q value. When the agent has no experience, it is necessary to explore. Once the agent adapts to the environment, the exploitation is performed, that is, when our "Q-table" becomes stable.

Bellman Equation

When performing a particular search for MAX, we are looking to retrieve the MAX within a spatial environment. Therefore, the approximation of the Bellman equation used in this work is expressed as follows:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a') \tag{4.1}$$

where:

- $Q(s,a)$: Q-value
- r : reward
- γ : discount factor, which in this case we use 0.8.
- $\max_{a'} Q(s',a')$: estimate of optimal future value

Equation 4.1 searches for a MAX in the matrix Q to construct a gradient connection from each exploration episode and openings in memory Q. Furthermore, an important part is the discount rate, which is the constant that determines the importance of future rewards. This constant helps balance the effect of upcoming rewards on the new values. A lower discount factor and increasing it towards its final value accelerates learning, so the discount factor chosen here is 0.8.

4.3 Experimental Setup

We define a randomly generated maze by placing the target consistently in the same place but varying the agent's starting location and the obstacles' location in the labyrinth. In each maze, the percentage of available spaces is less than the space occupied by blocks.

Therefore, the agent is forced to take a different route each time for each added obstacle. A random maze is considered solvable if the agent finds all target states within 3000 steps when performing a random walk through the maze [46]. Thus, we use such solvable maze problems in the experiments of this work.

Chapter 5

Results and Discussion

This chapter presents the experimental and final results obtained by the system developed in this project. The process in which the ESP266 chips communicate through the MQTT broker will be shown so that independent agents (clients) the begins the Q-learning process from its training phase to the exploitation phase.

5.1 MQTT communication

In real time running situations the control center is capable of distributing route maps to the gents (clients) by using MQTT channels. The agents use this received route maps and Q-learning, with C code embedded in each server (ESP8266), to find optimal routes by themselves in a simulated city disaster. When required. the control center can ask the clients to help in a concurrent, distributed manner, in finding independent, optimal routes and transmit this clue information to the control center via MQTT.

Figure 5.1 shows a part of the HiveMQ broker window, where we can see that communication has been established between the three ESP8266 nodes, one of which is the server, and the other two are clients.

The server starts by sending the map as a string to the two clients. Clients receive the map and notify the server. Then, the map is stored to transform them into a matrix of integers. This resulting matrix will be the R-matrix, which will represent a maze with its respective reward and will be used in the Q-learning process.

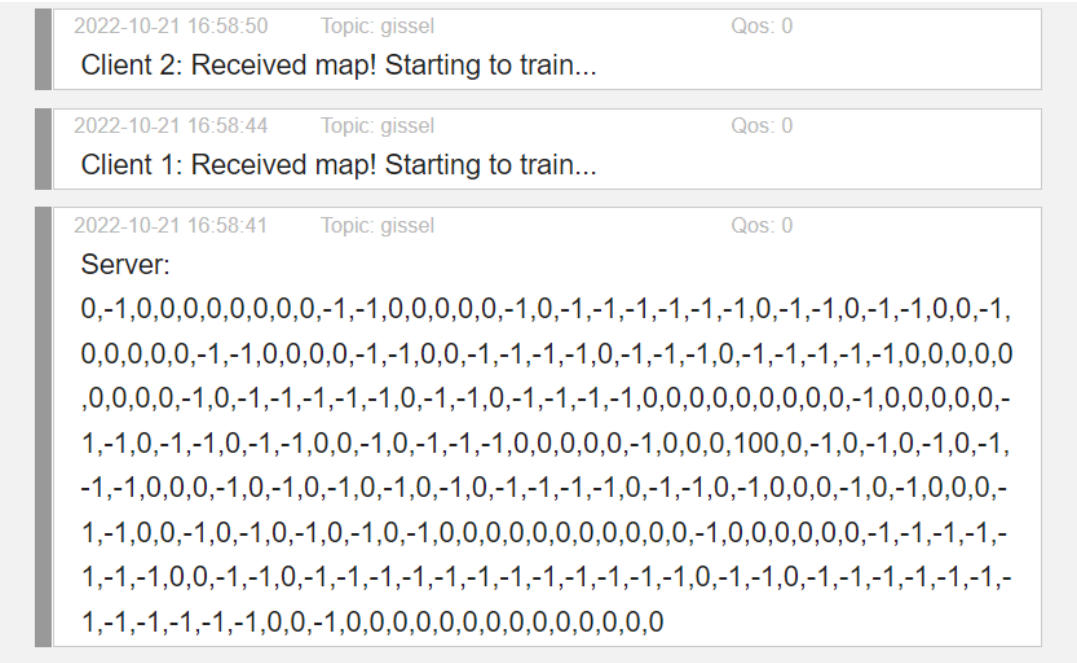


Figure 5.1: Server-Clients communication.

5.2 Agents exploration

On the other hand, the clients start with the Q-learning development. This process is reflected in the Arduino IDE consoles for each client.

In Figures (5.2-5.3), we can see the initialization of the Q-matrix and R-matrix matrices and the reward coordinate, which is located in row 7 and column 14. From this point, the clients start working and training by selecting an action randomly during 100 episodes.

```
Begins Q_learning
This is the Q Matrix:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.2: Initial Q-matrix.

```
This is the Goal coordinate:
7,14
This is the R Matrix:

0 -1 0 0 0 0 0 0 0 0 -1 -1 0 0 0 0
0 -1 0 -1 -1 -1 -1 -1 -1 0 -1 -1 0 -1 -1 0
0 -1 0 0 0 0 0 -1 -1 0 0 0 0 -1 -1 0
0 -1 -1 -1 -1 0 -1 -1 -1 0 -1 -1 -1 -1 -1 0
0 0 0 0 0 0 0 0 -1 0 -1 -1 -1 -1 -1 0
-1 -1 0 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0
-1 0 0 0 0 0 -1 -1 0 -1 -1 0 -1 -1 0 0
-1 0 -1 -1 -1 0 0 0 0 0 -1 0 0 0 100 0
-1 0 -1 0 -1 0 -1 -1 -1 0 0 0 -1 0 -1 0
-1 0 -1 0 -1 0 -1 -1 -1 -1 0 -1 -1 0 -1 0
0 0 -1 0 -1 0 0 0 -1 -1 0 0 -1 0 -1 0
-1 0 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0
-1 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 0 0
-1 -1 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0
-1 -1 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0
0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.3: Initial R-matrix.

Figures (5.4-5.6) show gradual episodes of the Q-matrix that reflect the process where the agent calculates the value of the reward and where the algorithm uses the Bellman

equation to update the values in the different states.

```
This is the Q Matrix:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 52 66 83 104 104
0 0 0 0 0 0 0 0 0 0 0 0 0 0 197 157
0 0 0 0 0 0 0 0 0 0 0 0 147 184 247 163
0 0 0 0 0 0 0 0 0 0 0 0 0 130 0 104
0 0 0 0 0 0 0 0 0 0 0 0 0 104 0 83
0 0 0 0 0 0 0 0 0 0 25 32 0 83 0 32
0 0 0 0 0 0 0 0 0 25 32 41 52 66 52 41
0 0 0 0 0 0 0 0 0 0 0 0 0 0 41 32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.4: Q-matrix after few subsequent episodes.

```
This is the Q Matrix:
0 0 0 0 0 0 0 0 0 0 0 0 20 25 32 41
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 52
0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 66
0 0 0 0 0 12 0 0 0 0 0 0 0 0 0 83
0 0 7 9 12 16 20 25 0 0 0 0 0 0 0 104
0 0 5 0 0 0 0 32 41 52 66 83 104 131 164 131
0 3 4 0 0 0 0 0 0 0 0 0 0 0 205 164
0 2 0 0 0 0 0 0 0 0 0 0 147 197 257 163
0 1 0 0 0 0 0 0 0 0 0 0 157 0 104
0 0 0 0 0 0 0 0 0 0 0 0 125 0 83
0 0 0 0 0 0 0 0 0 0 25 32 0 100 0 32
0 0 0 0 0 0 0 0 0 25 32 41 52 80 64 41
0 0 0 0 0 0 0 0 0 0 0 0 0 0 51 32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.5: Q-matrix after some subsequent episodes.

```
This is the Q Matrix:
0 0 0 0 0 0 0 1 2 3 0 0 22 28 35 44
0 0 0 0 0 0 0 0 0 4 0 0 17 0 0 56
0 0 0 0 0 9 0 0 0 6 8 10 13 0 0 71
0 0 0 0 0 12 0 0 0 4 0 0 0 0 0 89
0 0 7 9 12 16 20 25 0 0 0 0 0 0 0 112
0 0 5 0 0 0 0 32 44 56 71 89 112 140 176 140
0 3 4 0 0 0 0 0 35 0 0 112 0 0 220 176
0 2 0 0 0 0 0 0 28 22 0 140 176 220 276 220
0 1 0 0 0 0 0 0 0 71 89 112 0 176 0 175
0 0 0 0 0 0 0 0 0 0 71 0 0 140 0 140
0 0 0 0 0 0 0 0 0 0 0 56 44 0 112 0 112
0 0 0 0 0 0 0 0 0 28 35 44 51 64 89 71 89
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 56 71
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 32
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 25
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5.6: Q-matrix after many following episodes.

Finally, Figure 5.7 shows the Q-matrix stabilized after exploring new states and actions to improve its value function estimates. Once all the training episodes have been completed, the Q values converge to the optimal value we are looking for. Therefore, the next step to take is the exploitation stage.

```
This is the Q Matrix:
0 0 2 3 4 6 8 10 13 17 0 0 22 28 35 44
1 0 3 0 0 0 0 0 0 22 0 0 17 0 0 56
2 0 4 6 8 10 8 0 0 28 22 17 13 0 0 71
3 0 0 0 0 13 0 0 0 35 0 0 0 0 0 89
4 6 8 10 13 17 22 28 0 44 0 0 0 0 0 112
0 0 6 0 0 0 0 35 44 56 71 89 112 140 176 140
0 6 8 10 13 17 0 0 35 0 0 112 0 0 220 176
0 4 0 0 0 22 28 35 44 56 0 140 176 220 276 220
0 3 0 1 0 17 0 0 0 71 89 112 0 176 0 176
0 2 0 2 0 13 0 0 0 0 71 0 0 140 0 140
0 1 0 3 0 10 13 17 0 0 56 44 0 112 0 112
0 2 0 4 0 13 17 22 28 35 44 56 71 89 71 89
0 3 4 6 8 10 13 0 0 0 0 0 0 0 56 71
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 56
0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 44
0 0 1 0 1 2 3 4 6 8 10 13 17 22 28 35


--Ready to exploit--
```

Figure 5.7: Q-matrix stabilized.

5.3 Agent exploitation

In this stage, the agents begin to exploit the information it has about the value functions. First, clients load from their own memory the Q-matrix, as shown in Figure 5.8, which stores a gradient that the agents use to search for optimal paths to the reward.

```
lectura de Q matrix lista

This is the Q Matrix:
0 0 2 3 4 6 8 10 13 17 0 0 22 28 35 44
1 0 3 0 0 0 0 0 0 22 0 0 17 0 0 56
2 0 4 6 8 10 8 0 0 28 22 17 13 0 0 71
3 0 0 0 0 0 13 0 0 0 35 0 0 0 0 0 89
4 6 8 10 13 17 22 28 0 44 0 0 0 0 0 112
0 0 6 0 0 0 0 0 35 44 56 71 89 112 140 176 140
0 6 8 10 13 17 0 0 35 0 0 112 0 0 220 176
0 4 0 0 0 22 28 35 44 56 0 140 176 220 276 220
0 3 0 1 0 17 0 0 0 71 89 112 0 176 0 176
0 2 0 2 0 13 0 0 0 0 71 0 0 140 0 140
0 1 0 3 0 10 13 17 0 0 56 44 0 112 0 112
0 2 0 4 0 13 17 22 28 35 44 56 71 89 71 89
0 3 4 6 8 10 13 0 0 0 0 0 0 0 56 71
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 56
0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 44
0 0 1 0 1 2 3 4 6 8 10 13 17 22 28 35
```

Figure 5.8: Q-matrix reading.

The agents then indicate possible routes to the target point from different locations. Figures 5.9-5.11 show the coordinates of the paths traveled by an agent and then print the maze representing the path taken by the agent. The labyrinth shows the accessible paths with the symbol “*”, the obstacles with “-“, and the path of the agent with “@”.

```
-----Ready to play-----
peso: 44, File agent: 14, Column agent: 15
peso: 56, File agent: 13, Column agent: 15
peso: 71, File agent: 12, Column agent: 15
peso: 89, File agent: 11, Column agent: 15
peso: 112, File agent: 10, Column agent: 15
peso: 140, File agent: 9, Column agent: 15
peso: 176, File agent: 8, Column agent: 15
peso: 220, File agent: 7, Column agent: 15
peso: 276, File agent: 7, Column agent: 14
Done!
```

```
This is the Route Matrix:

* _ * * * * * * * * _ _ * * * *
* _ * _ _ _ _ _ _ * _ _ * _ _ *
* _ * * * * * * _ _ * * * * * *
* _ _ _ _ * _ _ _ * _ _ _ _ _ *
* * * * * * * * _ * _ _ _ _ _ *
_ _ * _ _ _ _ * * * * * * * * *
_ * * * * * _ _ * _ _ * _ _ * *
_ * _ _ _ * * * * * _ * * * @ @
_ * _ * _ * _ _ _ * * * _ * _ @
_ * _ * _ * _ _ _ _ * _ _ * _ @
* * _ * _ * * * _ _ * * _ * _ @
_ * _ * _ * * * * * * * * * * @
_ * * * * * * _ _ _ _ _ _ _ * @
_ _ * _ _ _ _ _ _ _ _ _ _ _ _ @
_ _ * _ _ _ _ _ _ _ _ _ _ _ _ *
* _ * * * * * * * * * * * * * *
```

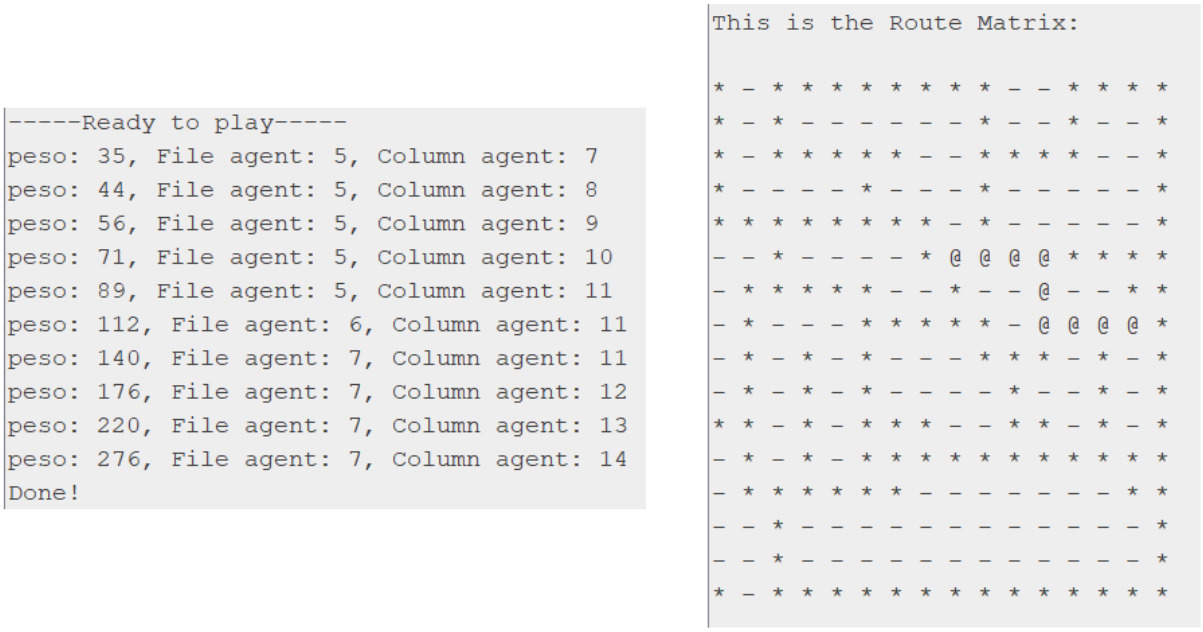
Figure 5.9: A possible route to the reward from a random point 1.

```
-----Ready to play-----
peso: 13, File agent: 2, Column agent: 12
peso: 17, File agent: 1, Column agent: 12
peso: 22, File agent: 0, Column agent: 12
peso: 28, File agent: 0, Column agent: 13
peso: 35, File agent: 0, Column agent: 14
peso: 44, File agent: 0, Column agent: 15
peso: 56, File agent: 1, Column agent: 15
peso: 71, File agent: 2, Column agent: 15
peso: 89, File agent: 3, Column agent: 15
peso: 112, File agent: 4, Column agent: 15
peso: 140, File agent: 5, Column agent: 15
peso: 176, File agent: 6, Column agent: 15
peso: 220, File agent: 7, Column agent: 15
peso: 276, File agent: 7, Column agent: 14
Done!
```

```
This is the Route Matrix:

* _ * * * * * * * * _ _ @ @ @ @
* _ * _ _ _ _ _ _ * _ _ @ _ _ @
* _ * * * * * * _ _ * * * * _ _ @
* _ _ _ _ * _ _ _ * _ _ _ _ _ @
* * * * * * * * _ * _ _ _ _ _ @
_ _ * _ _ _ _ * * * * * * * * @
_ * * * * * _ _ * _ _ * _ _ * @
_ * _ _ _ * * * * * _ * * * @ @
_ * _ * _ * _ _ _ * * * _ * _ *
_ * _ * _ * _ _ _ _ _ * _ _ * _ *
* * _ * _ * * * _ _ * * _ * _ *
_ * _ * _ * * * * * * * * * * *
_ * * * * * * _ _ _ _ _ _ _ * *
_ _ * _ _ _ _ _ _ _ _ _ _ _ _ *
_ _ * _ _ _ _ _ _ _ _ _ _ _ _ *
* _ * * * * * * * * * * * * * *
```

Figure 5.10: A possible route to the reward from a random point 2.



```
-----Ready to play-----
peso: 35, File agent: 5, Column agent: 7
peso: 44, File agent: 5, Column agent: 8
peso: 56, File agent: 5, Column agent: 9
peso: 71, File agent: 5, Column agent: 10
peso: 89, File agent: 5, Column agent: 11
peso: 112, File agent: 6, Column agent: 11
peso: 140, File agent: 7, Column agent: 11
peso: 176, File agent: 7, Column agent: 12
peso: 220, File agent: 7, Column agent: 13
peso: 276, File agent: 7, Column agent: 14
Done!
```

```
This is the Route Matrix:

* - * * * * * * * * - - * * * *
* - * - - - - - - * - - * - - *
* - * * * * * * - - * * * * - - *
* - - - - - * - - - * - - - - - *
* * * * * * * * * - * - - - - - *
- - * - - - - * @ @ @ @ * * * *
- * * * * * * - - * - - @ - - * *
- * - - - * * * * * - @ @ @ @ *
- * - * - * - - - * * * - * - *
- * - * - * - - - - - * - - * - *
* * - * - * * * - - * * - * - *
- * - * - * * * * * * * * * * *
- * * * * * * * - - - - - - - * *
- - * - - - - - - - - - - - - - *
- - * - - - - - - - - - - - - - *
* - * * * * * * * * * * * * * * *
```

Figure 5.11: A possible route to the reward from a random point 3.

5.3.1 Maze solutions

The clients notify the server and send the solutions of the mazes problems in a matrix form through the MQTT broker, as seen in Figures 5.12-13.

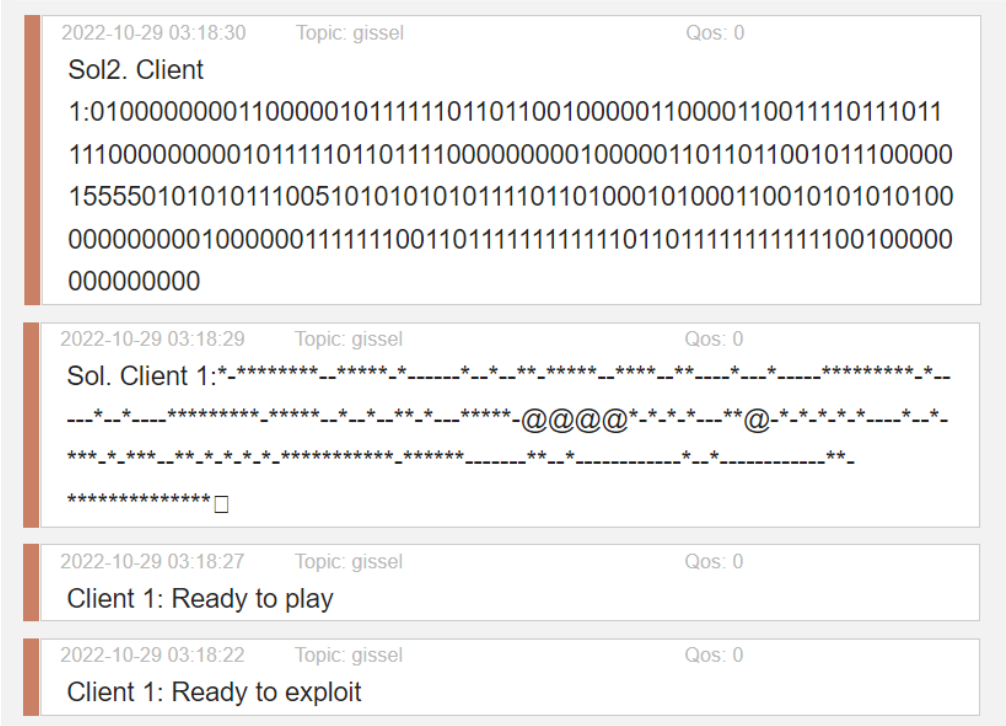


Figure 5.12: Client 1 solution to a maze problem.

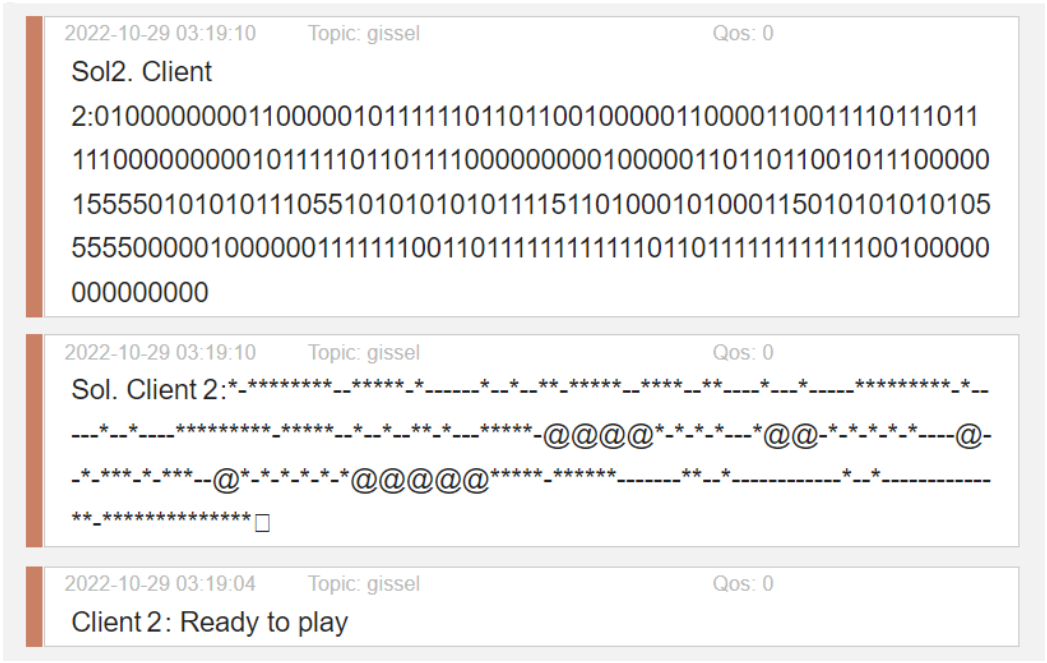


Figure 5.13: Client 2 solution to a maze problem

On the other hand, in the server serial monitor, we can see that it receives the solutions sent by the clients (Figure 5.14).

```
[gissel] Client 1: Received map! Starting to train...
[gissel] n...
[gissel] Server: 0,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1
[gissel] Server: 0,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1
[gissel] Client 1: Ready to exploit
[gissel] Client 1: Ready to play
[gissel] Server: 0,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1
[gissel] Sol. Client 1:*-********--*****-*------*--*--**-*****--******--**-
[gissel] Sol2. Client 1:01000000001100000101111110110110010000011000011100
[gissel] Server: 0,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1
[gissel] Client 2: Ready to play
[gissel] Sol. Client 2:*-********--*****-*------*--*--**-*****--******--**-
[gissel] Sol2. Client 2:01000000001100000101111110110110010000011000011100
[gissel] Server: 0,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1
[gissel] Client 1: Ready to play
[gissel] Sol. Client 1:*-********--*****-*------*--*--**-*****--******--**-
```

Figure 5.14: Server serial monitor.

5.3.2 Agents Results

Figure 5.15 graphically shows typical results obtained from the agents. Of course, each agent's behavior may vary to find the best route. Moreover, these results may depend on the training time taken by the agents; therefore, we will obtain better routes while the training is greater. Furthermore, since we are talking about a simulation of a real-life problem, it is necessary to have immediate solutions. In this case since the clients have already found a Q-matrix that represents an optimal policy, the desired optimal path are found rapidly and transmitted to the control center (sever) in a short time.

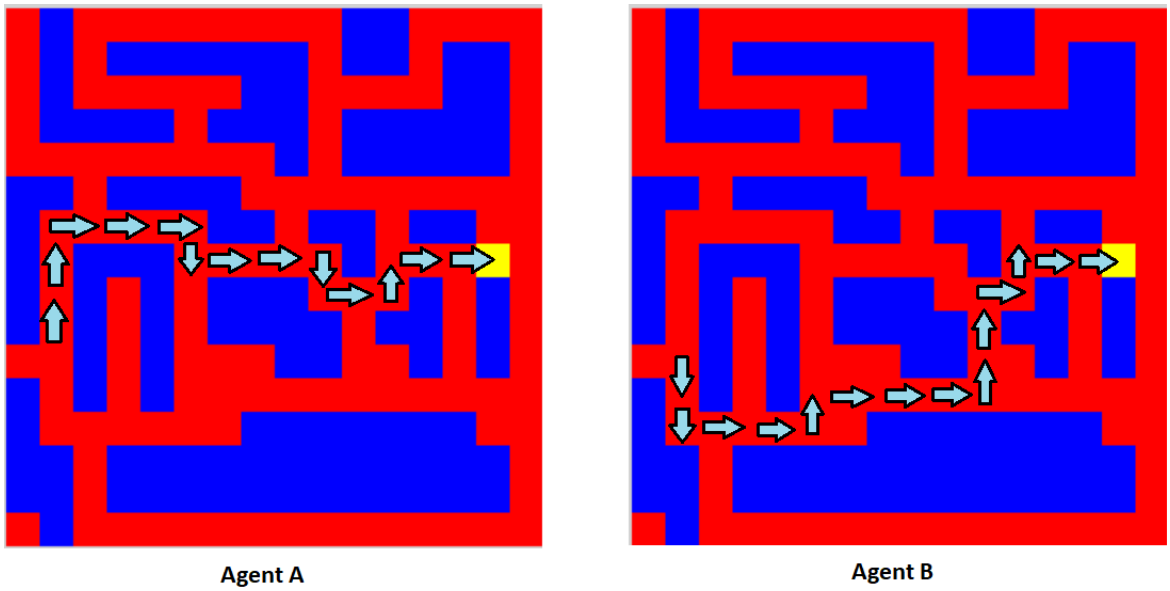


Figure 5.15: Agents response.

As can be seen, the IoT is the maximum representative of Concurrent Programming and the future of technology that supports Artificial Intelligence to regulate and understand things in a much stronger way. Offering the right mix of IoT hardware and software, Arduino is an easy-to-use IoT platform. It operates through a series of hardware specifications that can be given to interactive electronics, such as the search for alternative routes in this case.

Chapter 6

Conclusions

A IoT system was built capable of interconnect a control center (server) and two route finding agents (clients) through MQTT channels. The agents are capable of using Q-learning to find optimal routes in a simulated city disaster. The agents can help in a distributed manner to find optimal routes for the control center. The develop system is a miniature of a real life situation where distributed agents can independently learn efficient routes for a given maze, mimicking a big random interruption of roads in a city, and then contribute with this information to a call center. In principle, since the utilized elements are members of the internet of things IoT, many useful applications can be derived from the methods established in this thesis.

One of the most relevant characteristics of creating modern distributed systems, such as IoT, is that they are a way of designing high-tech equipment in all possible human, network, industrial production, and scientific activities.

Integrated IoT platforms and tools are considered the most crucial component of the IoT ecosystem. Any IoT device allows us to connect to other IoT devices and applications to transmit information using standard Internet protocols.

Arduino, in particular, has a large set of easy-to-use, pre-built C routines that communicate with specialized Arduino modules, which in turn measure real-world variables like humidity in agriculture or acceleration in robotics.

On the other hand, the MQTT broker is energy efficient and easy to implement. As a result, data management can be carried out between millions of devices and from anywhere in the whole world. Furthermore, MQTT in IoT uses QoS levels to ensure guaranteed

delivery of messages to receivers, even when connections between devices are unreliable. Therefore, we agree that the contribution to the development of Smart Cities with the methods used in this thesis is possible.

Bibliography

[1] L. Tan and N. Wang, "Future internet: The internet of things," in *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, vol. 5. IEEE, 2010, pp. V5–376.

[2] D. A. Aziz, "Webserver based smart monitoring system using esp8266 node mcu module," *International Journal of Scientific & Engineering Research*, vol. 9, no. 6, pp. 801–808, 2018.

[3] E. Gamess, T. N. Ford, and M. Trifas, "Performance evaluation of a widely used implementation of the mqtt protocol with large payloads in normal operation and under a dos attack," in *Proceedings of the 2021 ACM Southeast Conference*, 2021, pp. 154–162.

[4] L. García-García, J. M. Jiménez, M. T. A. Abdullah, and J. Lloret, "Wireless technologies for iot in smart cities," *Network Protocols and Algorithms*, vol. 10, no. 1, pp. 23–64, 2018.

[5] S. R. Sulistiyanti, F. Setyawan, S. Purwiyanti, H. Fitriawan, and A. R. Adnan, "Monitoring and control system with a client-server model based on internet of things (iot)," *IIUM Engineering Journal*, vol. 22, no. 1, pp. 93–102, 2021.

[6] E. Morales and J. González, "Reinforcement learning," *Presentacion En Linea en: https://ccc. inaoep. mx/˜ emorales/Cursos/Aprendizaje2/Acetatos/refuerzo. pdf*, 2012.

[7] M. A. Azar, J. L. García, S. Bernal, L. Aleman, and M. Tolaba, "Artificial intelligence applied to iot," in *XXII Workshop de Investigadores en Ciencias de la Computación (WICC 2020, El Calafate, Santa Cruz).*, 2020.

[8] M. A. Azar, M. Tapia, J. L. García, and A. J. M. Pérez, "Artificial intelligence of things," in *XXI Workshop de Investigadores en Ciencias de la Computación (WICC 2019, Universidad Nacional de San Juan).*, 2019.

[9] S. Barroso, A. Sánchez, P. Núñez, B. Muriel, M. L. Bonilla, and P. B. G. de Castro, "Experiences of a smart campus using artificial intelligence and iot to optimize water consumption," in *Greencities, 11º Foro de Inteligencia y Sostenibilidad Urbana: Actas del XI International Greencities Congress.* Palacio de Ferias y Congresos de Málaga (FCMA), 2021, pp. 213–231.

[10] E. Escribá Pina, "Reinforcement learning through deep learning for smart cities," Ph.D. dissertation, ETSI_Informatica, 2021.

[11] A. S. Abdul-Qawy, P. Pramod, E. Magesh, and T. Srinivasulu, "The internet of things (iot): An overview," *International Journal of Engineering Research and Applications*, vol. 5, no. 12, pp. 71–82, 2015.

[12] K. Rose, S. Eldridge, and L. Chapin, "The internet of things: An overview," *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.

[13] A. Polianytsia, O. Starkova, and K. Herasymenko, "Survey of hardware iot platforms," in *2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T).* IEEE, 2016, pp. 152–153.

[14] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, 2015.

[15] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[16] M. Kusriyanto and A. A. Putra, "Weather station design using iot platform based on arduino mega," in *2018 International Symposium on Electronics and Smart Devices (ISESD).* IEEE, 2018, pp. 1–4.

[17] T. A. Team, "Overview of the arduino ide 1: Arduino documentation." [Online]. Available: https://docs.arduino.cc/software/ide-v1/tutorials/Environment

[18] M. A. Khan, M. A. Khan, S. U. Jan, J. Ahmad, S. S. Jamal, A. A. Shah, N. Pitropakis, and W. J. Buchanan, "A deep learning-based intrusion detection system for mqtt enabled iot," *Sensors*, vol. 21, no. 21, p. 7016, 2021.

[19] D. Dinculeană and X. Cheng, "Vulnerabilities and limitations of mqtt protocol used between iot devices," *Applied Sciences*, vol. 9, no. 5, p. 848, 2019.

[20] F. Antonielli, "Development and comparison of mqtt distributed algorithms for hivemq," 2021.

[21] H. Koziolek, S. Grüner, and J. Rückert, "A comparison of mqtt brokers for distributed iot edge computing," in *European Conference on Software Architecture*. Springer, 2020, pp. 352–368.

[22] F. Telefónica, *Smart Cities: a first step towards the internet of things*. Fundación Telefónica, 2011.

[23] C. Yin, Z. Xiong, H. Chen, J. Wang, D. Cooper, and B. David, "A literature survey on smart cities," *Science China Information Sciences*, vol. 58, no. 10, pp. 1–18, 2015.

[24] F. A. R. Handbook, "Alternate route handbook," *Federal Highway Administration, US Department of Transportation*, 2006.

[25] R. R. Rout, S. Vemireddy, S. K. Raul, and D. Somayajulu, "Fuzzy logic-based emergency vehicle routing: An iot system development for smart city applications," *Computers Electrical Engineering*, vol. 88, p. 106839, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790620306935

[26] J. Barrachina, P. Garrido, M. Fogue, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni, "Reducing emergency services arrival time by using vehicular communications and evolution strategies," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1206–1217, 2014.

[27] H. S. Oluwatosin, "Client-server model," *IOSR Journal of Computer Engineering*, vol. 16, no. 1, pp. 67–71, 2014.

[28] A. Alharin, T.-N. Doan, and M. Sartipi, "Reinforcement learning interpretation methods: A survey," *IEEE Access*, vol. 8, pp. 171 058–171 077, 2020.

[29] M. A. Wiering and M. Van Otterlo, "Reinforcement learning," *Adaptation, learning, and optimization*, vol. 12, no. 3, p. 729, 2012.

[30] M. N. Cansado, A. S. Morillas, and D. M. Sastre, "Pavlov's principles of classical conditioning in the creative advertising strategy," *Opción*, vol. 31, no. 2, pp. 813–831, 2015.

[31] C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan, "Is q-learning provably efficient?" *Advances in neural information processing systems*, vol. 31, 2018.

[32] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," in *Australasian joint conference on artificial intelligence*. Springer, 1999, pp. 417–428.

[33] M. Lee, "Reinforcement learning," Apr 2005. [Online]. Available: https://web.archive.org/web/20090806064734/http:/www.cs.ualberta.ca/~sutton/book/ebook/node7.html

[34] F. Garcia and E. Rachelson, "Markov decision processes," *Markov Decision Processes in Artificial Intelligence*, pp. 1–38, 2013.

[35] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[36] M. Jones and M. M. Peet, "Extensions of the dynamic programming framework: Battery scheduling, demand charges, and renewable integration," *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1602–1617, 2020.

[37] ——, "A generalization of bellman's equation with application to path planning, obstacle avoidance and invariant set estimation," *Automatica*, vol. 127, p. 109510, 2021.

[38] L. L. Lozano, "Aprendizaje por refuerzo elementos básicos y algoritmos."

[39] E. Even-Dar, Y. Mansour, and P. Bartlett, "Learning rates for q-learning." *Journal of machine learning Research*, vol. 5, no. 1, 2003.

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

[41] D. Pandey and P. Pandey, "Approximate q-learning: An introduction," in *2010 second international conference on machine learning and computing*. IEEE, 2010, pp. 317–320.

[42] T. Horiuchi, A. Fujino, O. Katai, and T. Sawaragi, "Fuzzy interpolation-based q-learning with continuous states and actions," in *Proceedings of IEEE 5th International Fuzzy Systems*, vol. 1. IEEE, 1996, pp. 594–600.

[43] T. Tompa and S. Kovács, "Q-learning vs. friq-learning in the maze problem," in *2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. IEEE, 2015, pp. 545–550.

[44] H. Sharma, A. Haque, and F. Blaabjerg, "Machine learning in wireless sensor networks for smart cities: a survey," *Electronics*, vol. 10, no. 9, p. 1012, 2021.

[45] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of iot: Applications, challenges, and opportunities with china perspective," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 349–359, 2014.

[46] A. D. 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 (SSCI)*. IEEE, 2016, pp. 1–8.

Appendices

.1 Appendix 1.

.1.1 Server

Send_data_mqtt_server.ino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

//WiFi settings
#define wifi_ssid       "NETLIFE-OFFLINE"      // local router name
#define wifi_password   "#1003432562."                 // router
    password

//MQTT setting
#define mqtt_server   "broker.mqttdashboard.com"
#define mqtt_user     "oscar"
#define mqtt_password "racso"
#define mqtt_port 1883
#define  files 16
#define  columns 16

//MQTT Topic to be suscribe
#define intopic "gissel"    //
using std::string;
WiFiClient espClient;
PubSubClient client;

//Global variables
String command;
//Maze matrix
String R="0,-1,0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,
0,-1,0,-1,-1,-1,-1,-1,-1,0,-1,-1,0,-1,-1,0,
0,-1,0,0,0,0,0,-1,-1,0,0,0,0,-1,-1,0,
0,-1,-1,-1,-1,0,-1,-1,-1,0,-1,-1,-1,-1,-1,0,
0,0,0,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1,0,
-1,-1,0,-1,-1,-1,-1,0,0,0,0,0,0,0,0,0,
-1,0,0,0,0,0,-1,-1,0,-1,-1,0,-1,-1,0,0,
-1,0,-1,-1,-1,0,0,0,0,0,-1,0,0,0,100,0,
-1,0,-1,0,-1,0,-1,-1,-1,0,0,0,-1,0,-1,0,
-1,0,-1,0,-1,0,-1,-1,-1,-1,0,-1,-1,0,-1,0,
0,0,-1,0,-1,0,0,0,-1,-1,0,0,-1,0,-1,0,
-1,0,-1,0,-1,0,0,0,0,0,0,0,0,0,0,0,
-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,0,
-1,-1,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,
```

```
-1,-1,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,
0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0";

//---------------------------------------
void setup()
 {
    Serial.begin(115200);
    setup_wifi();
    client.setClient(espClient);
    client.setServer(mqtt_server, mqtt_port);
    client.setCallback(callback);
    client.setBufferSize(1024);
 }
//---------------------------------------
void setup_wifi()
{
    delay(10);
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(wifi_ssid);
    WiFi.begin(wifi_ssid, wifi_password);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");                // here a time out could be
           necesary
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    Serial.println(intopic);
}
//---------------------------------------
void reconnect() {        // Loop until reconnection with Broker

 while (!client.connected()) {
    Serial.print("Intentando conexi n Mqtt...");
    String clientId = "IOTICOS_H_W_";   // Create a ID client
    clientId += String(random(0xffff), HEX);
    // Trying to connect
    if (client.connect(clientId.c_str(),mqtt_user,mqtt_password)) {
      Serial.println("Conectado!");
```

```
82        // Suscribe to broker
83        if(client.subscribe(intopic)){
84          Serial.println("Suscripcion ok");
85        }else{
86          Serial.println("fallo Suscripcii n");
87        }
88      } else {
89        Serial.print("fall  :( con error -> ");
90        Serial.print(client.state());
91        Serial.println(" Intentamos de nuevo en 5 segundos");
92        delay(1000);
93      }
94    }
95 }
96
97 //-------------------------------------
98 void callback(char* topic, byte* payload, unsigned int length) {
99     Serial.print("Mensaje recibido [");
100    Serial.print(topic);
101    Serial.print("] ");
102    String(receivedChar) = "";
103    for (int i = 0; i < length; i++) {
104        receivedChar += (char)payload[i];
105    }
106    Serial.println(receivedChar);
107 }
108 //-------------------------------------
109 void loop()
110 {
111    if (!client.connected()) {reconnect();}
112          int x=random(0,5);
113          int y=random(0,5);
114          //Publish the maze map in broker
115          String result="Server: "+ R;
116          client.publish(intopic,(char*)result.c_str());
117
118    client.loop();
119   delay(10000);
120 }
```

.1.2 Clients

Agent_in_maze.ino

```
1  //Client
2
3  #include <ESP8266WiFi.h>
4  #include <PubSubClient.h>
5  #include<EEPROM.h>
6
7  #define relay 2          // pin al que se conecta el rel  o Led, para
       este ejemplo (GPIO0)
8  #define wifi_ssid        "NETLIFE-OFFLINE"      // local router name
9  #define wifi_password    "#1003432562."                 // router
       password
10 #define mqtt_server    "broker.mqttdashboard.com"
11 #define mqtt_user      "gissel"
12 #define mqtt_password "12345"
13 #define mqtt_port 1883
14 #define files 16
15 #define columns 16
16 #define intopic "gissel"    //
17
18 WiFiClient espClient;
19 PubSubClient client;
20 String stack[256];
21
22 //Q-learning variables
23 int Q[files][columns];
24 int R[files][columns];
25 int a[256];
26 int i=0;
27 int currentState;
28 int episode;
29 int file_agent;    // file
30 int column_agent;  // column
31 int MAX;
32 int sensor[4];
33 int sample[4];
34 int grad_pointer;
35 int maxQ;
36 char route[files][columns];
37 int peso;
38 int x_agent_pos, y_agent_pos;
39 const float gamma1 = 0.8;
40 char temp[256];
41 char camino[256];
42 char str2[256];
```

```
43 String cad;
44 String cad2;
45 String(receivedChar) = "";
46 String(command) = "";
47 //----------------------------------------
48 void setup()
49  {
50     pinMode(relay, OUTPUT);         //  Blue LED in the board
51     digitalWrite(relay, HIGH);
52     Serial.begin(115200);
53     setup_wifi();
54     client.setClient(espClient);
55     client.setServer(mqtt_server, mqtt_port);
56     client.setCallback(callback);
57     client.setBufferSize(1024);
58  }
59 //----------------------------------------
60 void setup_wifi()
61 {
62     delay(10);
63     Serial.println();
64     Serial.print("Connecting to ");
65     Serial.println(wifi_ssid);
66
67     WiFi.begin(wifi_ssid, wifi_password);
68
69     while (WiFi.status() != WL_CONNECTED)
70     {
71         delay(500);
72         Serial.print(".");                 // here a time out could be
               necesary
73     }
74     Serial.println("");
75     Serial.println("WiFi connected");
76     Serial.println("IP address: ");
77     Serial.println(WiFi.localIP());
78     Serial.println(intopic);
79 }
80 //----------------------------------------
81 void reconnect() {                    // to connect with Broker
82  // Bucle hasta conseguir la reconexi n
83  while (!client.connected()) {
84     Serial.print("Intentando conexi n Mqtt...");
85     // Creamos un cliente ID
```

```
86      String clientId = "IOTICOS_H_W_";
87      clientId += String(random(0xffff), HEX);
88      // Intentamos conectar
89      if (client.connect(clientId.c_str(),mqtt_user,mqtt_password)) {
90        Serial.println("Conectado!");
91        // Nos suscribimos
92        if(client.subscribe(intopic)){
93          Serial.println("Suscripcion ok");
94        }else{
95          Serial.println("fallo Suscripcii n");
96        }
97      } else {
98        Serial.print("fall  :( con error -> ");
99        Serial.print(client.state());
100       Serial.println(" Intentamos de nuevo en 5 segundos");
101       delay(1000);
102     }
103   }
104 }
105
106 //-------------------------------------
107 void callback(char* topic, byte* payload, unsigned int length) {
108     Serial.print("Mensaje recibido [");
109     Serial.print(topic);
110     Serial.print("] ");
111 //    String(receivedChar) = "";
112 //    String(command) = "";
113     for (int i = 0; i < length; i++) {
114         receivedChar += (char)payload[i];
115     }
116 //    char *ret;
117 //    ret = strcasestr(receivedChar, "server");
118 //    if(ret){
119
120
121     for (int i= receivedChar.indexOf(":")+2; i < length; i++) {
122         command += (char)payload[i];
123     }
124
125     char str[1024];
126     strcpy(str,(char*)command.c_str());
127     char* token1 = strtok(str, ",");
128     while (token1 != NULL) {
129       if(i>=0){
```

```
130          a[i]= atoi(token1);
131          }
132          token1 = strtok(NULL, ",");
133          i++;
134     }
135          Serial.print("\nSerial: Client 1: Received map! Starting to
                train...\n");
136          client.publish(intopic, "Client 1: Received map! Starting to
                train...");
137          delay(1000);
138
139 /*******************START***********************************/
140        receive_data();
141        delay(1000);
142        file_agent=0;
143        column_agent=0;
144        print_Q(); //Print initial Q and R matrix
145        print_R();
146        client.publish(intopic, "Initialize...");
147        //client.loop();
148        delay(500);
149        explore();
150        exploit();
151
152 }
153
154 //============================================================
155 void explore(){ //loop
156
157       Q_explore();
158       print_R();
159       print_Q();
160       Serial.print( "\n--Ready to exploit--");
161       client.publish(intopic, "Client 1: Ready to exploit");
162       delay(4000);
163   }
164 void exploit(){
165   load_Q();
166   print_Q();
167   max_Q();
168   while (1){
169   Serial.print( "\n--Ready to play--");
170   client.publish(intopic, "Client 1: Ready to play");
171
```

```
172   Q_player();
173   delay(5000);
174     }
175   }
176
177 void loop(){
178
179     if (!client.connected()) {reconnect();}
180        Serial.print(" \nBegins Q_learning...");
181        //client.publish(intopic, "Hello");
182        client.loop();
183        delay(200);
184 }
```

Q_structures_lib.ino

```
1  void print_Q()
2  {
3   int i,j;
4
5      Serial.print("\nThis is the Q Matrix:\n");
6      for(i = 0; i < files ; i++)
7          {
8          for(j = 0; j < columns; j++)
9              {
10             //cout << setw(5) << Q[i][j];
11             Serial.print( Q[i][j]);
12             if(j < files - 1) Serial.print(" ");  // separador de de
                   columnas
13     } // j
14         Serial.print("\n");
15         //cout << "\n";
16   } // i
17     //cout << "\n";
18     Serial.print("\n");
19 }
20 //-------------------------------------------------------
21 void print_R()
22 {
23  int i,j;
24
25     Serial.print("This is the R Matrix:\n \n");
26     for(i = 0; i < files; i++)
27         {
```

```
28          for(j = 0; j < columns; j++)
29              {
30                Serial.print( R[i][j]);
31              //cout << setw(5) << R[i][j];
32                if(j < files - 1) Serial.print(" ");  // separador de de
                      columnas
33     } // j
34       Serial.print("\n");
35         //cout << "\n";
36   } // i
37     //cout << "\n";
38     Serial.print("\n");
39 }
40 //-------------------------------------------------------
41 void move_left(void)
42 {
43  int color,temp;
44
45   if(column_agent>0) temp= R[file_agent][column_agent-1];     // lee  a
         izquierda
46   else return; //goto scape;
47   if(temp>-1)
48     {
49      column_agent--;
50     }
51 //scape:
52 }
53 //-------------------------------------------------------
54 void move_right(void)
55 {
56  int temp;
57   if(column_agent<15) temp= R[file_agent][column_agent+1];
58   else return; //goto scape;
59   if(temp>-1)
60    {
61     column_agent++;
62    }
63 //scape:
64 }
65 //-------------------------------------------------------
66 void move_up(void)
67 {
68  int temp;
69   if(file_agent>0) temp= R[file_agent-1][column_agent];
```

```
70   else return;//goto scape;
71   if(temp>-1)
72    {
73     file_agent--;
74    }
75 //scape:
76 }
77 //-------------------------------------------------------
78 void move_down(void)
79 {
80  int temp;
81   if(file_agent<15) temp= R[file_agent+1][column_agent];
82   else return;
83   if(temp>-1)
84    {
85     file_agent++;
86    }
87 //scape:
88 }
89 //-----------------------------------------------
90 void Q_explore(){
91
92  int temp, explore_state;
93  int i,j,k,timer;
94  float max1;
95  int initial_state;
96
97
98  //randomize();
99    episode=1;
100   do
101    {
102          do
103          {
104           i=random(files);
105           j=random(columns);
106           //i=2+random(4);
107           //j=7+random(5);
108           temp=R[i][j];
109          } while(temp==-1);    // el agente se ubica en estado inicial
                aleatorio con entrada diferente de -1
110
111          file_agent=i;          // ubicamos al agente en la matriz
112          column_agent=j;
```

```
113
114          //file_agent=0;           // ubicamos al agente en la matriz
115          //column_agent=0;
116
117          do
118           {
119            i=random(4);      // el agente se mueve a un nuevo estado
                  aleatorio
120
121            if(i==0) move_left();
122            if(i==1) move_right();
123            if(i==2) move_up();
124            if(i==3) move_down();
125
126            search_for_MAX();
127            max1=MAX;
128
129            Q[file_agent][column_agent]= R[file_agent][column_agent]+
                  gamma1*max1;
130
131            //plot_maze();
132            //plot_agent();                      //  sensor captures
                  color
133            delay(1);
134            timer++;
135            //Serial.print(timer);
136            //getch();
137            if(timer>310) break;
138           }while(i!=7 && j!=14);      //  hasta que capture recompensa
                maxima
139
140         delay(200);
141
142         episode++;
143         //system("cls");
144
145         if(timer<320) {
146           //system("cls");
147           //delay(500);
148           print_R();
149           print_Q();}
150           //Serial.print("\nEpisode: ");
151           //Serial.print(episode);
152           Serial.print("\n");
```

```
153          timer=0;
154      }while(episode<30);  // 40
155
156       save_Q();
157  }
158
159 //-------------------------------------------------------
160  void search_for_MAX(void)
161 {
162  int i,j;
163      sensor[0]= Q[file_agent][column_agent-1];     // lee  a izquierda
164      if(column_agent==0) sensor[0]=-1;             // desborde a la
             izquierd
165
166      sensor[1]= Q[file_agent][column_agent+1];     // lee a  derecha
167      if(column_agent==columns-1) sensor[1]=-1;
168
169      sensor[2]= Q[file_agent-1][column_agent];     // lee arriba
170      if(file_agent==0) sensor[2]=-1;
171
172      sensor[3] = Q[file_agent+1][column_agent];     // lee abjo
173      if(file_agent==files-1) sensor[3]=-1;
174
175      MAX=sensor[0];
176      for(i=0;i<4;i++) if(sensor[i]>=MAX) {MAX=sensor[i];grad_pointer=i
             ;}  // grad_pointer apunta al maximo valor
177     //cout <<" MAX: "<< MAX <<endl;
178 }
179 //------------------exploit--------------------
180 void get_gradient(void)
181 {
182  int i, center;
183
184      //center= Q[file_agent][column_agent];
185      //cout <<"center "<< center <<endl;
186      //cout<<endl;
187
188      sample[0]= Q[file_agent][column_agent-1];     // lee  a izquierda
189      if(column_agent==0) sample[0]=-1;
190
191      sample[1]= Q[file_agent][column_agent+1];     // lee a  derecha
192      if(column_agent==columns-1) sample[1]=-1;
193
194
```

```
195
196      sample[2]= Q[file_agent-1][column_agent];     // lee arriba
197      if(file_agent==0) sample[2]=-1;
198
199      sample[3]= Q[file_agent+1][column_agent];     // lee abjo
200
201      if(file_agent==files-1) sample[3]=-1;
202
203      MAX=sample[0];
204      for(i=0;i<4;i++)
205       {
206         if(sample[i]>=MAX) {MAX=sample[i];grad_pointer=i;}
207       }
208
209  }
210  //-------------------------------------------------------
211  void max_Q(){
212    int i,j,mayor, temp;
213
214    mayor = Q[0][0];
215      for(i=0;i <files ;i++)
216      {
217
218          for(j=0; j< columns;j++)
219          {
220              temp=Q[i][j];
221              if(temp > mayor)
222                  mayor = temp;
223          }
224
225      }
226      maxQ = mayor;
227  }
228
229  //---------------------------------------------------------------
230  void clean_Route(void)
231  {
232     int i,j,k;
233     for (j=0; j<files;j++)
234      for (i=0; i<columns; i++)
235      {
236        route[j][i]='*';
237      }
238
```

```
239 }
240
241 //------------------------------------------------------------
242 void plot_Route(void){
243     int i,j,sx,sy,x,y,color,scale;
244 int temp, temp2;
245
246 char wall='-', floor='*', reward='$' ,way='@';
247
248
249     for(int i = 0; i < files; i++)
250         {
251        for(j = 0; j < columns; j++)
252             {
253                 if (i==x_agent_pos && j==y_agent_pos)
254                     R[i][j]= Q[x_agent_pos][y_agent_pos];
255                     if(R[i][j]!=0) route[x_agent_pos][y_agent_pos]=way
                        ;
256
257                     if(R[i][j]<0) route[i][j]=wall;           // WALL
258                     if(R[i][j]==0) route[i][j]=floor;           //
                        FLOOR
259                     if(R[i][j]==1000) route[i][j]=reward;      //
                        REWARD
260
261     } // j
262
263   } // i
264
265     //memset(route, 0, sizeof(route));
266 }
267 //------------------------------------------------------------
268 void print_Route(void){
269
270    int i,j, k;
271
272 k=0;
273     Serial.print("This is the Route Matrix:\n \n");
274     for(i = 0; i < files; i++)
275         {
276         for(j = 0; j < columns; j++)
277             {
278               temp[k]=route[i][j];
279               k++;
```

```
280                 Serial.print( route[i][j]);
281               //cout << setw(5) << R[i][j];
282                 if(j < files - 1) Serial.print(" ");  // separador de de
                        columnas
283     } // j
284       Serial.print("\n");
285         //cout << "\n";
286   } // i
287     //cout << "\n";
288     Serial.print("\n");
289
290
291
292     }
293 //--------------------------------------------------------
294
295 void Q_player(void)
296 {
297  int i,j,k;
298  int max1,temp;
299  int initial_state, next_state;
300
301
302  //randomize();
303     do
304       {
305        do
306          {
307           i=random(files);
308           j=random(columns);
309           temp=Q[i][j];
310
311          } while(temp==0);    // buscamos sitios alejados de la
               recompensa
312
313
314
315          file_agent=i;
316          column_agent=j;
317         // String result= "\nClient 1: (" +String(file_agent)+","+
               String(column_agent)+")";
318          //Serial.print(result);
319          //client.publish(intopic,(char*)result.c_str());
320
```

```
321         Serial.print("\npeso: ");
322         Serial.print(Q[file_agent][column_agent]);
323         Serial.print(", File agent: ");
324         Serial.print(file_agent);
325         Serial.print(", Column agent: ");
326         Serial.print(column_agent);
327 
328         delay(300);
329 
330         do{
331 
332            get_gradient();
333            if(grad_pointer==0) column_agent=column_agent-1;
334            if(grad_pointer==1) column_agent=column_agent+1;
335            if(grad_pointer==2) file_agent=file_agent-1;
336            if(grad_pointer==3) file_agent=file_agent+1;
337 
338            x_agent_pos=file_agent;
339            y_agent_pos=column_agent;
340 
341            Serial.print("\npeso: ");
342            Serial.print(Q[file_agent][column_agent]);
343            Serial.print(", File agent: ");
344            Serial.print(file_agent);
345            Serial.print(", Column agent: ");
346            Serial.print(column_agent);
347            //result= "\nClient 1: (" +String(file_agent)+","+String(
                  column_agent)+")";
348            //Serial.print(result);
349            //client.publish(intopic,(char*)result.c_str());
350 
351            delay(400);
352            plot_Route();
353 
354          }while(Q[file_agent][column_agent]!= maxQ);
355 
356          Serial.print("\nOne problem solved!\n");
357 
358 
359         //system("cls");
360         print_Route();
361         print_route_mqtt();
362         clean_Route();
363         //print_Q();
```

```
364           break;
365
366         } while(1);
367 }
```

Manage_data.ino

```
1  void receive_data(){
2
3   int k=0;
4  for (int i=0; i<files; i++){
5    for (int j=0; j<columns; j++){
6      R[i][j]= a[k];
7      k++;
8      //delay(100);
9    }
10 }
11 }
12
13 void print_route_mqtt(){
14
15        for (i =0;i<256;i++){
16
17            if(temp[i]=='-'){
18              camino[i]='-1';}
19            else if(temp[i]=='*'){
20              camino[i]='0';}
21            else if(temp[i]=='@'){
22              camino[i]='5';}
23      }
24      char temp3[256];
25      for (i =0;i<256;i++){
26        temp3[i]=camino[i];
27      }
28      cad=String(temp3);
29      cad2=String(temp);
30      String resp="Sol. Client 1:" + cad2;
31      String resp2="Sol2. Client 1:" + cad;
32      //Serial.print("\n");
33      //Serial.print(cad);
34      //Serial.print("\n");
35      //Serial.print(cad2);
36      Serial.print("\nDone\n");
37      client.publish(intopic, (char*)resp.c_str());
```

```
38      client.publish(intopic, (char*)resp2.c_str());
39      delay(3000);
40  }
```

read_write_Q_matrix.ino

```
1  //-------------------------------------
2  //   write in prom routine
3  //   save weights in prom
4  //-------------------------------------
5   void save_Q()
6   {
7   int i,k;
8   int eeAddress = 0;
9   float femp=0.0;
10  EEPROM.begin(4096);           //tama o maximo 4096 bytes
11     for (k=0; k<files; k++){
12       for (i=0; i<columns; i++)
13       {
14         EEPROM.put(eeAddress, Q[k][i]);
15         //EEPROM.put(eeAddress, femp);
16         //femp=femp+0.1;
17         eeAddress += sizeof(float);
18         Serial.print(Q[k][i]);
19         if(i < files - 1) Serial.print(" ");
20         delay(10);
21       }
22       Serial.print("\n");
23     }
24     Serial.print("\n");
25
26    EEPROM.commit();
27    Serial.println("escritura de Q matrix lista");
28  }
29  //-----------------------------------------
30  void load_Q()
31  {
32   int i,k;
33   int eeAddress = 0;
34   EEPROM.begin(4096);
35        for (k=0; k<files; k++)
36        for (i=0; i<columns; i++)
37        {
38          EEPROM.get(eeAddress, Q[k][i]);
```

```
39          eeAddress += sizeof(float);
40          //Serial.println(Q[k][i]);
41          delay(3);
42        }
43   Serial.println("lectura de Q matrix lista");
44 }
```