Creation of an agent in reinforcement learning using explainability methods in a complex environment

Project Thesis

Author: Bartomeu Perelló Comas
Director: Sergio Álvarez Napagao
Co-Director: Dmitry Gnatyshak
GEP Tutor: Andujar Larios Agustin

Computation
Facultat d'informàtica de Barcelona (FIB)
Universitat politècnica de Catalunya (UPC)
Barcelona
Delivery: 18/10/2021
Defence : 25/10/2021

Abstract

La intel·ligència artificial es una de les àrees de la computació que está en auge actualment, aquest treball s'emmarca dins de l'aprenentatge per reforç. L'aprenentatge per reforç es centra en la resolució de problemes o també anomenats escenaris mitjançant la creació de models d'intel·ligència artificial anomenats agents. Els objectius d'aquests agents es maximitzar el valor que retorna una funció una vegada aquest ha realitzat una acció. Generalment tots aquests agent estan formats per xarxes neuronals. Quan tenim una xarxa neuronal entrenada aquesta es vista desde fora com una capsa negra que no entenem com perquè està funcionant d'una forma determinada degut a la seva complexitat. La explicabilidad són un conjunt de tècniques que s'encarreguen de donar llum a aquesta capsa i donar explicacions de com funcina. En aquest treball s'apiquen tècniques de explicabilidad en un videojoc anomenat ViZDoom, en el qual podem trobar escenaris complexos a resoldre.

La inteligencia artificial es una de las áreas de la computación que se encuentra actualmente en auge, este trabajo se encuentra bajo el marco del aprendizaje por refuerzo. El aprendizaje por refuerzo se centra en la resolución de problemas conocidos como escenarios mediante la creación de modelos de inteligencia artificial llamados agentes. Los objetivos de esto es maximizar el valor que nos retorna una función que una vez el agente ha realizado una acción en el escenario. Generalmente todos estos agentes han sido creados mediante el uso de redes neuronales. Al tener entrenada una red neuronal, desde fuera puede parecer como una caja negra ya que no sabemos porqué o cómo está solucionando el problema debido a su complejidad. La explicabilidad son un conjunto técnicas que se encargan de arrojar luz a esta caja intentando así dar explicaciones del porqué de su funcionamiento. En este trabajo se aplican este tipo de técnicas en el videojuego ViZDoom, en el cual se encuentran escenarios complejos de resolver.

Since the beginning of artificial intelligence as a field it has shown humanity how useful it has been to solve complex problems which there were no feasible algorithms for. One of the main problems is that once we have a system trained is like a black box. Explainability tries to cast light on those black

boxes and explain how they are working or the case of Reinforcement Learning explains how they behave. In this project we will use some explainability techniques in the ViZDoom game which contains complex environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and context

1.1 Introduction

Since the beginning of artificial intelligence as a field it has shown humanity how useful it has been to solve complex problems which there were no feasible algorithms for. Artificial intelligence requires the existence of hardware capable of handling the computation required in order to solve the desired problem. As time went by, more powerful hardware was being developed such as graphics cards which in recent years have allowed machine learning algorithms be present on most of the services provided by enterprises such as recommendation systems, self-driving cars, video editing software among others; it is also used in medical applications to help doctors reduce their time when searching through patient data.

Reinforcement learning is an area of machine learning which relies on the agent taking decisions based on a reward system which will lead to a behaviour that is able to solve the problem; in order to do so for complex problems neural networks can be used. And when using deep neural networks it is almost impossible for humans to understand how the agent is working.

Explainability methods in reinforcement learning make it possible for us to extract information that can be understood by humans from a complex model of Artificial Intelligence, which allows us to check whether the trained agent is performing the actions based on a strategy.

The goal of this project is to see if an agent trained with these techniques is able to perform as well as another agent trained with conventional ones as a follow-up to a previous project in which positive results were achieved

in a simple environment (Cart Pole) using a policy graph extracted from the trained agent (similar to a Markov Decision Process).

1.2 Extent

Up next we will define the main objective of the project as well as some risks and problems we may encounter while doing the project.

1.2.1 Main goal

This problem has been investigated before in simpler environments such as Cart Pole; we would like to recreate positive results in a video game with a complex environment. We have chosen the original Doom game, a first person shooter in which life or ammunition management are key when trying to play the game. In order to have a clear view on how this project will be handled we split the main objective into six.

1.2.2 Split goals

In the following list the differtent sub-goals will be explained:

- Looking for existing projects: The aim of this goal is to obtain the required knowledge to carry out this project.
- In order to train our agent several neural networks will have to be used so we will have to research for the ones that fit our project the most.
- Generation of predicates: In this part we want to add code into the agent or the environment so we can extract predicates which will allow us to create a policy graph.
- In this goal we will have to code an algorithm which will allow us to ask questions about the behaviour of the agent to the policy graph.
- Along with the code created is the previous goal and the explainability method selected we will be able to train a new agent based on the behavior of the original agent
- Finally we will compare the performance of both agents and get conclusions.

1.2.3 Requirements

In this project we have two types of requirements, functional and non-functional. We will start with the functional ones.

Functional requirements

- As we want an agent which is capable of playing original Doom we will probably use a Deep Q Network with some sort of memory system so it can learn different strategies depending on the state.
- Also we will need to instrument the code so in the end we can generate a policy graph which will be very helpful in the explainability part of the project
- Finally once the policy graph is created along with the explainability method we selected we will have to create a simpler agent capable of playing the game as close as possible to the neural network one.

Non-functional requirements

- Our algorithm that will give us an action given the current state of the game should be efficient enough to return an answer so each run execution does not last longer than expected.
- The code must be modular so if changes to the environment are made the adaptation of functions should be minimal, reducing the development time.

1.2.4 Risks and problems

There is no project where there never have been any problem to face which delays the progress, in our case we think that we may encounter two different problems defined below.

Time

The first and main problem may be that the time is limited, due to the complexity of the problem and the non-existing experience with this topic. It may

take a large amount of time just to understand and get the required knowledge. Another problem related with time is training, because this project focuses on a complex environment multiple trainings will probably be done because the performance obtained will not be as good as it is desired. Moreover the process of creating the agent can end up as a trial and error which can be very time consuming.

Bugs

The other problem we may face are software bugs which may lead to our agent to malfunction, the delay it can produce may be very big due to the use of a new programming language. Bugs can come from different sections of the overall code from the image input, which will be the only feedback the agent will receive, to the interpretation of the rewards of the environment. The amount of different modules that this project will be made of can create a huge amount of errors that may be hard to find and correct.

1.3 Stakeholders

The stakeholders of this project will probably be engineers that work with reinforcement learning systems, so once their agent is trained they will be able to understand if the agent has learned a strategy they already taught or has discovered a new one which was never intended. Also as this project is an extension of Dmitry Gnatyshak's PhD project.

1.4 Research Objectives

In this brief section we would like to state a few objectives to complete by the end of the project.

- First of all, the main objective is to check if applying the same explainability techniques used in [3] which try to replicate some behaviour based on the creation of predicates which end up forming a Policy graph. In case the same methods do not work properly we will have to modify them so we are able to achieve positive results.
- Secondly as time passes probably some problems will arise, at that case we should note down what the problem has been and also if this

problem can be postponed as future research in case it could lead to delays.

Chapter 2

State of the art

2.1 Terms and concepts

In this section we will show a few concepts related to the area of the project.

2.1.1 Machine Learning

Machine learning [12] is one of several branches of Artificial Intelligence, this one is focused on letting the system learn or train itself by feeding it with data instead of programming its behaviour. There are several approaches on how models are trained, supervised learning, unsupervised learning and the one this project is based on, reinforcement learning.

2.1.2 Reinforcement Learning

Reinforcement Learning [19] is an area of Machine Learning which focuses in the creation of models called Agents which perform actions in an environment, the aim of those Agents is to maximize a reward function provided by the environment.

A problem in a reinforcement learning context can be split into several states, an agent which is the model or program we want to train and the one in charge of taking actions, another state is the environment, which describes the problem to solve. When the agent takes an action a reward is assigned to it and a new representation of the state is given to it, this process is repeated until the agent is capable of solving the problem.

2.1.3 Neural Networks

A neural network [11, 18, 30], as the name suggests, is a structure which tries to replicate the human brain. Each network is an interconnection of layers formed by elements called neurons. A neuron is composed of a vector of weights and a mathematical function called activation function. When a neuron receives an input vector, it uses its weights to calculate the weighted sum of the input vector and then pass this result to the activation function to get this neuron's output. In the most common neural network models the outputs of neurons of each layer is then used as an input vector for the next layer. The classical network models consist of an input layer, an output layer and a number of hidden layers in between. The minimal neural network is a single layer, which works as both an input and output layer.

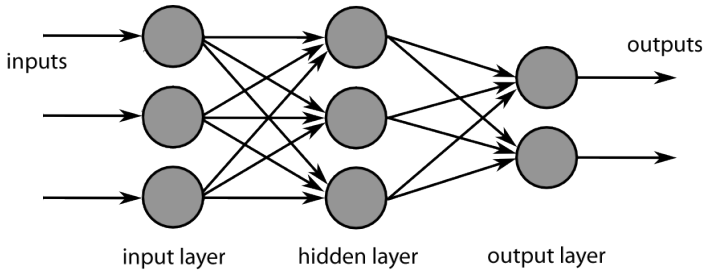


Figure 2.1: Example of a neural network. Image taken from [7]

2.1.4 Convolutional Networks

Convolutional neural networks (CNN) [10] are a variation of neural networks. They are specialized in pattern recognition by having layers formed by neurons that apply a transformation to the input by using a matrix called filter, this operation is called convolution. The result of each layer is often called feature or features depending on the number of filters we apply. These layers sometimes are connected to common networks to process the information they have gathered.

2.1.5 Agent

An agent in reinforcement learning is the entity which performs the actions, the environment where the agent is located represents the state or the prob-

lem we are trying to solve, due to the complexity of the environment a deep neural network will be used as an agent.

Every reinforcement learning training process is almost the same, first of all the agent performs an action, afterwards the environment is updated and a new state is sent to the agent as well as a reward with its value depending on the results of the actions taken, so they can guide our agent into solving the problem.

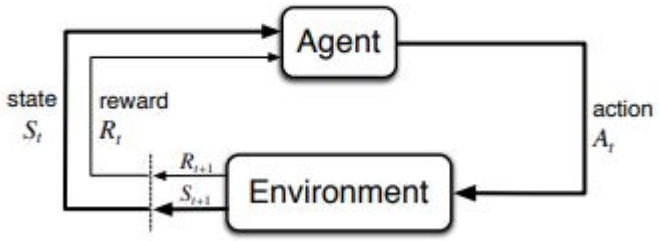


Figure 2.2: Simplified agent training process. Image taken from [25] Fig 3.1

2.1.6 Markov decision process

A Markov Decision Process (MDP) [2] is a form of representing the behaviour of an agent throughout the process of solving a problem. Each MDP can be represented as a graph where each node represents a state and each edge represents an action with a probability of going into another state, along with the probability we can assign to each edge the reward we want to give to the agent for doing this action. In our case we will use a similar approach which will be called policy graph.

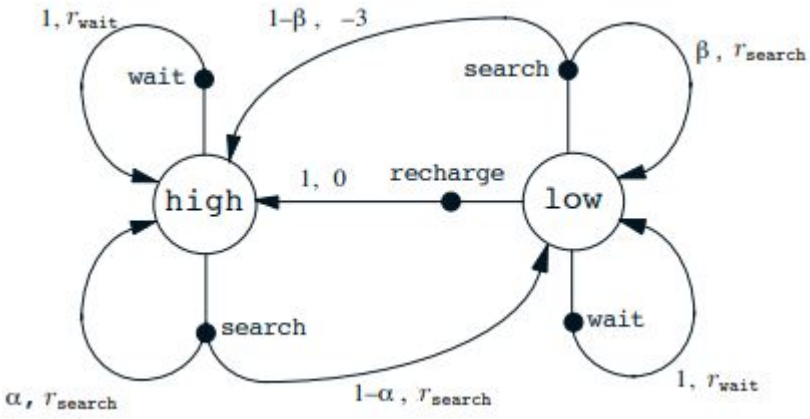


Figure 2.3: Example of a Markov Decision Process. Image taken from [25], Fig 3.2

2.1.7 Policy

A policy is a function which returns the probability of going into a state given an action. We can define a policy π for a given action a and state s as $\pi = (a|s)$.

2.2 Environments

Reinforcement learning has a lot of complex environments on which we could train the agent, let us have a look at different options.

- Neural MMO [21]: Neural MMO is a multi-agent game environment provided by OpenAI, one the the leading industries in the machine learning area, the objective is to create a player who is able to manage food and drink while competing against other players in three different combat styles.

- Pommerman [15]: This environment is based on the original Bomberman, it has all of the original features such as power ups which allow the player to get extra bombs, increase its range and the capability of kicking bombs, each one of these upgrades may change the strategy of the agent.

- SMAC [20]: SMAC stands for StarCraft Multi-Agent Challenge which focuses on micromanagement in the StarCraft 2 game, micro managements are strategies which focus on maximizing the output of an action while minimizing the losses it may produce due to interaction with other players.

- ViZDoom [31]: This is an application whose objective is to use the original doom game as a training environment using only visual information, it contains different scenarios so we can create an all-round agent.

2.3 Explainability methods

The explainability methods are key for this project and the selection of the method to implement must be well thought out. We came across three possible methods.

Rationalization

We can understand rationalization as a process in which a trained agent is able to explain the reasons behind its behaviour [5]. The way they approached this problem is by feeding a network with examples of human gameplay along with words or commentary about the decisions taken. A major problem about this implementation is the time it takes to create the required input data, also as this will be implemented in a complex environment we would need a huge amount of gameplay to cover most of the situations.

LIME

LIME [16] stands for local interpretable model-agnostic explanations, as the name suggests the goal of this method is to be able to explain any model as long as it a classifier. In order to train the model first the input must be human readable, in our case it can be possible depending on the environment, afterwards some noise is added to the input data so the model can learn which variables are important to check in order to give explanations.

Autonomous Policy Explanation

In another study a method is introduced that is able to make any agent explain itself without the intervention of the user in the training phase, and once this phase is finished humans can ask questions to the system on why it is performing an action or when it will perform a given action [9]. This is a very interesting approach because it allows us to get direct knowledge about its behavior.

2.4 Justification

In the end we have selected to use ViZDoom as the training environment due to the following reasons:

- It provides almost all game variables which will be useful at the time of creating the policy graph
- It contains a huge amount of scenarios so we can train our agent in different skills
- It is a single agent environment, which is very helpful because of a lack of experience in agent training with neural networks.

There is a lot of information regarding this environment which will be helpful in case we have to face implementation problems or runtime bugs, which will be key due to the delivery deadline.

As we have selected VizDoom as the target environment we can already discard LIME as a viable approach due to its requirement of the input being human readable, in our case, ViZDoom uses the game screen as input so LIME cannot use it as information to train, also, if it were to add noise into the image it could remove vital information leading to errors in the explanations.

Finally rationalization cannot be selected due to the need of feeding it with gameplay, it would require us to spend a lot of time playing, so we will implement the Autonomous Policy Explanation which will be very helpful in the last phase of the project.

2.5 Methodology

For this project we will use the scrum methodology, the basis of scrum is to split the project into sections called sprints.

In our case we think that weekly sprints will be enough for the development of the project.

2.6 Tools

In order to organize this project and have it finished by the deadline we will use several tools.

- Trello [26]: Trello is a very handful tool for the scrum methodology it is easy to use and very effective, in pasts projects it has shown its utility.
- GitLab [8]: This is a Git version control tool which will be used to store the progress of the project so in case of any computer failing or data loss we will always have a copy on hand.
- RocketChat [17]: This is an application which will be used as a fast communication channel.
- Discord [4]: In case of meetings we will use this platform which allows its users to have a voice chat while being able to share the screen if it is necessary.

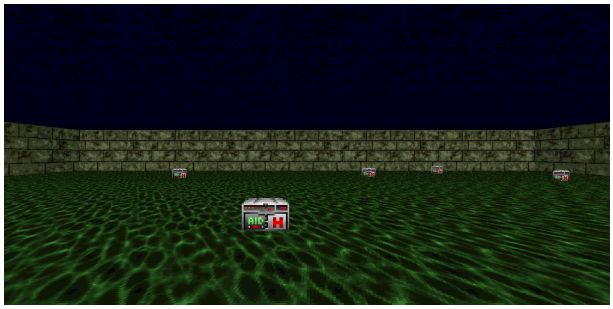
Chapter 3

Implementation

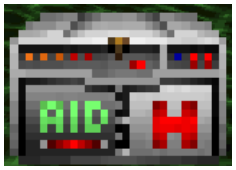
As mentioned before, in this project we want to use explainability methods to check if we can extract valid behaviour from a trained agent which will be able to solve the same problem. First of all we will describe the environment used for this research.

3.1 VizDoom environment

VizDoom is a platform for training AI models which is based on the original Doom game, this platform contains several complex environments, in our case we have selected the Health-gathering scenario. This environment is settled in a closed squared room, the floor is made of acid which deals damage to the player throughout the duration of the simulation. The goal of the scenario is to survive for the longest, in order to do so several med-kits, which restore health to the player if they are picked up, spawn randomly inside the room, so if the player is able to collect those med-kits, it will be able to survive longer. The simulation ends if the player dies which means that its health reaches a value of 0 or a total of two minutes have passed. The reward given to the player is 1 point for each tick of the game.



(a) Environment used in this project.



(b) Med-kit that restores health.

Figure 3.1: Images from the VizDoom environment. Self made

3.2 State representation

VizDoom has a high customization when it comes to state representation, it allows its users to select which variables can be accessed with a simple command [28], it also provides two different buffers, one which contains each element on the environment along with its own variables and the second one has the elements which are visible on the current scene [27].

In order to use explainability we need to discretize the different variables that the platform provides so it can be transformed into a policy graph. During the project several discretizations have been tested, in this document we will explain two different states and also a few changes that have been made in the environment.

3.3 Agent creation

In order to create our Deep Q Network Agent we have used an example code provided by ViZDoom which makes use of the Tensorflow 2.0 library as a starting point. This code is able to solve a simpler environment called "simpler_basic" the goal of the environment is to kill a static enemy which spawns randomly along a line in front of the player, so in order to make a network capable of solving our selected environment changes must be made.

A common structure for Deep-Q Agents that receive images from the game as input have three elements, our agent follows the same structure.

- A memory buffer: A memory buffer is used so we can keep track of the recent actions taken along with the reward obtain and the state of the problem. This is very useful in order to have some temporal information. This kind of information is critical in scenarios where there are moving objects because it allows the agent to keep track of the movement rather than just show the immediate image.

- Convolution layers: Those layers are responsible of applying filters to the incoming images and generating an output called feature, the combination of features are called feature maps. The values from the filters are also modified through training as well as the weight in the neurons of the neural networks.

- Neural network: The part of the agent that is in charge of the processing of the information that the convolution layers have gathered and based on the information give an action to perform.

In our case we have used a different structure in the neural network section, we use an architecture proposed by DeepMind [29] which splits the processing of the neural network into two streams, one with only one neuron to calculate the value of the current state and another with the amount of actions the agent can perform. This configuration leads to the agent being able to know how valuable is an state beforehand, which is useful when the actions taken do not modify the environment heavily, so it does not has to learn the consequences of an action after each step. The code for the model can be seen in Figure 3.2.

Some important parameters we have been testing when creating the agent are:

- the re-scaled resolution of the game images,

- the number of frames on which the agent has to maintain an action, and

- if we should use colored images or not.

The resolution of the image is important because if it is too low, we may be leaving details that are important to solving the problem, in this case could be the distance to the walls not being recognisable or not being able to see where med-kits are located, on the contrary if the resolution is too

```
class DQN(tf.keras.Model):
    def __init__(self, num_actions):
        super(DQN, self).__init__()
        self.conv1 = tf.keras.Sequential([
            tf.keras.layers.Conv2D(
                32, kernel_size=7, strides=4, input_shape
=(85,85,1)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.ReLU()
        ]) # (85,85,1)|(160,120,1)
        self.conv2 = tf.keras.Sequential([
            tf.keras.layers.Conv2D(
                32, kernel_size=5, strides=2, input_shape
=(20,20,32)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.ReLU()
        ])# (20,20,32) | (39,29,32)
        self.conv3 = tf.keras.Sequential([
            tf.keras.layers.Conv2D(
                32, kernel_size=3, strides=2, input_shape=(8,
 8, 32)),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.ReLU()
        ]) # (8, 8, 32) | (18,13,32)
        self.flatten = tf.keras.layers.Flatten()
        self.stateValue = tf.keras.layers.Dense(1) # Get the
value of the state
        self.advantage = tf.keras.layers.Dense(num_actions)

    def call(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        x1 = x[:, :144]
        x2 = x[:, 144:]
        x1 = self.stateValue(x1)
        x2 = self.advantage(x2)
        x = x1 + (x2 - tf.reshape(tf.math.reduce_mean(x2,
axis=1),shape=(-1,1)))
        return x
```

Figure 3.2: Neural Network model code for the trained agent

large it will require more space in the memory buffer. Linked to the problem of the space on the memory buffer we find the use of colored images, using those means that for each frame of the game we need to keep three versions of the same image, each one containing information of each color, red, green and blue, making the usable space for new states three times smaller. In the end we decided to use a resolution of 85X85 pixels and no colored images.

The other variable we mentioned before is the number of frames on which the agents has to commit to one action, having a high number of frames per action can lead to the agent not perform well because it cannot react to changes in the environment or its location and having a low amount of frames it can make the agent have an erratic behaviour too difficult to extract, also the aim of training an agent is to make it as close as human behaviour as possible.

3.4 Training process

We had to face some problems when training our agent, so in order to solve them we made a few changes in the reward system.

3.4.1 First approach

At first it was quite difficult to get the network to perform well into the environment in a reasonable training time so in order to make it converge faster we decided to add a little reward to the agent each time it would pick up a med-kit. Doing so improved the training time a lot. Now the agent would perform decently after 5 to 8 epochs of training which is equivalent to a range between 10 000 and 16 000 iterations. While the training time got faster even with these changes we could not appreciate a solid behaviour, a random walking agent could possibly work on average so we decided to make another change.

3.4.2 Second approach

The next modification was to change the reward for each tick, instead of rewarding a point, it would remove a low amount, in this case it was 0.001, by doing this change the agent finally adopted a clear behaviour which is to walk close to a wall an run in diagonals until it reaches the next one.

3.5 Policy graph

In order to use explainability we need to make a representation of the state so we are able to create a Graph that somehow can represent the behaviour of our trained agent. Throughout the duration of the project we have used different state representations in the next sections we will explain each state created and tested.

3.5.1 Health state

This state representation was used and tested before the second change to the reward system, at that moment the behaviour of the agent consisted on running to med-kits. In order to try and replicate the behaviour we created a representation that made use of two classes which are called med, and health_state. The med class (see Figure 3.3) is used to store information such as its position, distance to the player, the number of neighbours and two strings representing its location from the player.

```
class med:
    # Maximum distance to be considered a neighbour
    d2n = 10

    def __init__(self, pos, horizontal):
        self.neigh = 0
        self.pos   = pos
        self.d2p   = -1
        self.vert  = "NoVertical"  # ( "Far"|"Mid"|"Close")
        self.hor   = "NoHorizontal"# ("Left"|"Front"|"Right")
```

Figure 3.3: *med* class: player state representation

The health_state (see Figure 3.4) is the responsible of creating the med object based on the information retrieved from the game. It creates an array with all the med-kits on-screen and afterward takes the (N) closest to the player and creates an array representing the amount of med-kits in each type of location.

At the end we thought that with the behaviour the agent had it would not be as clear to spot big differences between the Graph Policy and a Random policy. At this point we decided to modify again the reward system (as mentioned in Section 3.4.2).

```
class healthState:
    def __init__(self, st):
        # Array to check the medkits available, and player position
        self.medkits = []
        self.playerPos = []
        self.maxElements = 7
        for l in st.objects:
            if l.name == "DoomPlayer":
                self.playerPos = np.array([l.position_x, l.position_y, l.position_z])
                break

        # Taking important info
        for l in st.labels:
            if l.object_name == "Medikit":
                currMed = med(
                    np.array([l.object_position_x, l.object_position_y, l.object_position_z]),
                    med.calculateHorizontal((l.x+l.width/2)))
                self.medkits.append(currMed)

        # Calculating distance to player and sorting vector
        for m in self.medkits:
            m.calculateD2p(self.playerPos)
        if len(self.medkits) > 0 and self.medkits[0].d2p != None:
            self.medkits = sorted(self.medkits, key=lambda x: x.d2p)
```

Figure 3.4: *health_state* class

3.5.2 Simple Wall 1.0

After the second change in the reward system our Agent started to behave in a clear way, it was moving in smoothed diagonals from one wall to another while picking up med-kits, when we saw this behaviour we knew this could be extracted with some representation. This representation is called simple_wall, we calculate an approximate distance from the player to each wall and then we made a transformation to a string, we decided to have four values for range: *Close*, *Mid*, *Far* and *VeryFar*, we also created a variable that recorded how much time had elapsed since the Agent had changed from one action

to another. Finally the state was composed of two distances those being the wall in front of the player and the wall on the left, this decision was made after watching the Policy Graph using only the right turn and the time variable having a range of: *Recent*, *WhileAgo* and *Ages*.

3.5.3 Simple Wall 2.0

After some testing, the previous representation did not work as good as we once expected, we decided to remove the temporal variable leaving the state representation into two variables (see Figure 3.5). Surprisingly, this small change made the Policy Graph more consistent throughout the runs.

At first it may seem that this state representation is simple, as it does not allow high precision at extracting behaviour. However, because we are using a graph to store the observable behaviour of a trained agent, we need to make a decision when it comes to the complexity of the representation: having a simpler representation could mean we would not be able to get a decent approximation, but, on the contrary, it would mean more variables therefore a more sparse graph with higher amount of states which would impact the performance of the explainability method by making the search of a similar state or even the exactly the same one more time consuming, thus, making the new agent unable to keep up with the pace of the game.

3.5.4 Creation of the Policy Graph

The creation of the Policy Graph and the results of the project follow the flow-chart depicted in Figure 3.6.

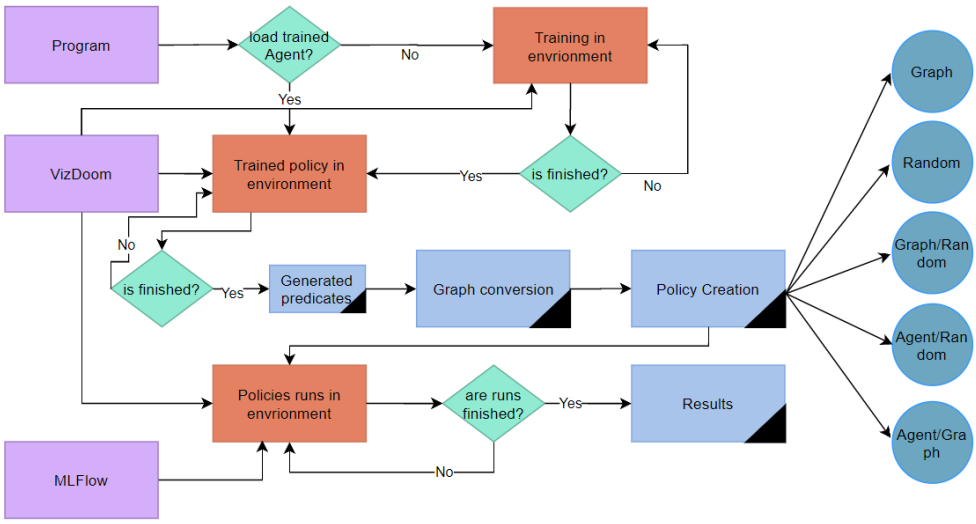


Figure 3.6: Flowchart of the program

In Figure 3.6 we can see the different steps that the program goes through. At the start we can decide if we want to train a new agent or use an already trained agent so we avoid the whole training process. Because ViZDoom is the game we are using to do the research, it is a part of all of the environment-related processes. Once we have finished training or have loaded the selected Agent, we run it in an environment to generate predicates which, once the runs are finished, are converted into the Graph policy. Finally what we create several new policies so we can run experiments on them with the help of MLFlow platform which allows us to keep track of the information that constitutes the results.

3.5.5 Usage of the Policy Graph

In order to make use of the Policy Graph at each step where the player can perform an action we get the current representation of the state and give it to a function which is the responsible of returning an action depending on the Policy Graph. If the current state is not located in the Policy Graph, this function tries to look for a state that is close to it, if there are no more states it returns a random action. The code for these choices is shown in Figures 3.7 and 3.8. As we mention this function is called from the function specified in Figure 3.9, which is called from the main game loop.

```
class simpleWall:
    pos = {"left":0,"top":1,"right":2,"bottom":3}
    leftPos = {"left":3,"top":0,"right":1,"bottom":2}
    def __init__(self, st, time):
        playerX = int(st.game_variables[0])
        playerY = int(st.game_variables[1])
        self.playerAngle = int(st.game_variables[3])
        for s in st.sectors:
            first = True
            minX = maxX = minY = maxY = 0
            for l in s.lines:
                if first:
                    first = False
                    minX = min(l.x1,l.x2)
                    maxX = max(l.x1,l.x2)
                    minY = min(l.y1,l.y2)
                    maxY = max(l.y1,l.y2)
                else:
                    currminX = min(l.x1,l.x2)
                    currmaxX = max(l.x1,l.x2)
                    currminY = min(l.y1,l.y2)
                    currmaxY = max(l.y1,l.y2)
                    if currminX < minX:
                        minX = currminX
                    if currmaxX > maxX:
                        maxX = currmaxX
                    if currminY < minY:
                        minY = currminY
                    if currmaxY > maxY:
                        maxY = currmaxY
        leftWall    = int(abs(playerX - minX))
        topWall     = int(abs(playerY - maxY))
        rightWall   = int(abs(playerX - maxX))
        bottomWall  = int(abs(playerY - minY))
        self.timeChange = "Error"
        if time < 0.35:
            self.timeChange = "Recent"
        elif time < 0.75:
            self.timeChange = "WhileAgo"
        else:
            self.timeChange = "Ages"
        self.state = [leftWall,topWall,rightWall,bottomWall]
        self.ststate = self.convert2String(self.state)
```

Figure 3.5: *simpleWall* class

```
def whatToDoString(state, myProbs):
    if state in myProbs.keys():
        random.seed()
        currState = myProbs[state].items()
        newAct = {key:round(value,1)*10 for key,value in
    currState}
        totalElm = int(sum(newAct.values()))
        aux = [ _ for _ in range(totalElm)]
        i = 0

        for key,value in newAct.items():
            for _ in range(int(value)):
                aux[i] = key
                i+=1
        pos = random.randint(0,totalElm-1)
        return aux[pos]
    # LOOKING FOR THE CLOSEST STATE
    else:
        posDict = {"Close":0,"Mid":1,"Far":2,"VeryFar":3}
        posArray = ["Close","Mid","Far","VeryFar"]
        currPos = posDict[state[0]]
        currLeftPos = posDict[state[1]]
        currState = (posArray[currPos],posArray[currLeftPos])

        if currPos < 2:
            while not (currState in myProbs.keys()) and
    currPos < 4:
                currLeftPos += 1
                if currLeftPos > 3:
                    currLeftPos = 0
                    currPos += 1
                currState = (posArray[currPos],posArray[
    currLeftPos])
        else:
            while not (currState in myProbs.keys()) and
    currPos > 0:
                currLeftPos -= 1
                if currLeftPos < 0:
                    currLeftPos = 3
                    currPos -= 1
                currState = (posArray[currPos],posArray[
    currLeftPos])
```

Figure 3.7: Code for making decisions based on the Policy Graphs (part I)

```
        try:
            newState = myProbs[currState].items()
            newAct = {key:round(value,1)*10 for key,value in
    newState}

            totalElm = int(sum(newAct.values()))

            aux = [ _ for _ in range(totalElm)]
            i = 0

            for key,value in newAct.items():
                for _ in range(int(value)):
                    aux[i] = key
                    i+=1
            random.seed()
            pos = random.randint(0,totalElm-1)

            return aux[pos]

        except:
            return "Random"
```

Figure 3.8: Code for making decisions based on the Policy Graphs (part II)

```
def mdpChooseAction(state, myProbs):
    # Maps the string action to the position in actions array
    myActionsMap={"None":0,"Left":4,"Right":2,"Forward":1,"
    LeftForward":5,"RightForward":3,"LeftRight":6,"
    LeftRightForward":7}

    action = whatToDoString(state,myProbs)

    if action == "Random":
        return np.random.choice(range(len(myActionsMap)), 1)
    [0]

    return myActionsMap[action]
```

Figure 3.9: Choose action based on an actions array

Chapter 4

Results

In this chapter we will explain everything related to the data obtained after some experiments.

4.1 Methodology of evaluation

In order to evaluate the results we will be using the same metrics as in previous research work on the topic [3] by creating different policies and tracking several variables that will allow us to compare the agent's behaviour to the one extracted by the explainability methods.

The policies we have created are the neural network-based agent, the extracted behaviour which we will be calling a graph, a random agent and a mix of agent with graph, agent with random and graph with random. Each part of the combined policies has an even chance of selecting the action that will be used in the testing process.

For extracting the data needed for evaluation we have set a seed for the environment. This way we can have the same state throughout the entire run so that we can achieve a fair comparison between policies. Also we have executed several runs with random seeds each time to check how each policy changes.

4.2 Analysis

In this section we well discuss the results obtained.

4.2.1 Overall sight

Now we will be looking at some charts that contain information of all the runs done, afterwards we will check case by case. Down below we have some charts that represent the average health throughout all runs.

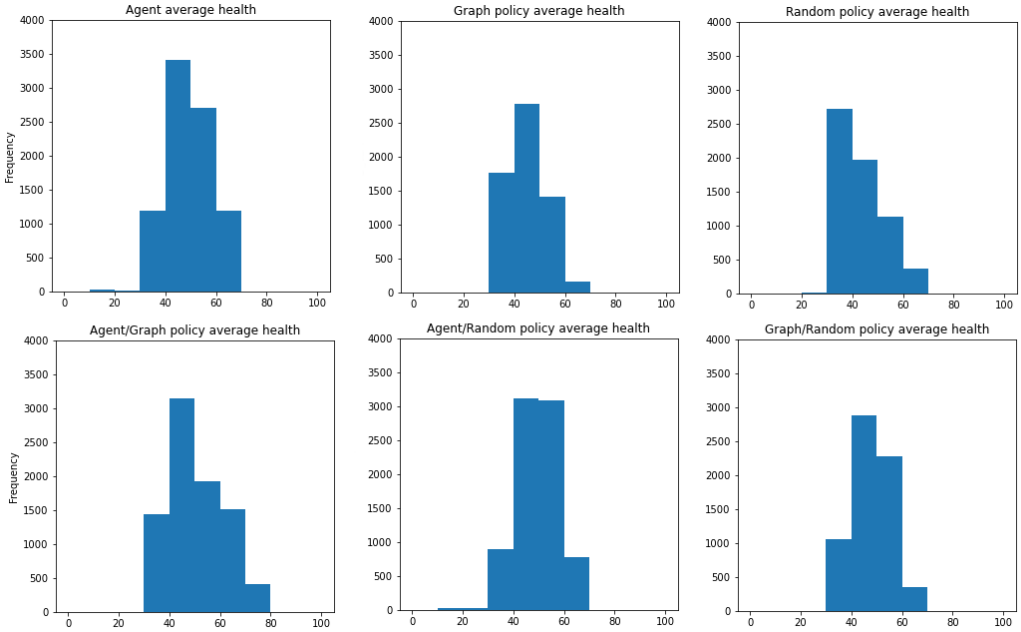


Figure 4.1: Histogram of average health. Self made.

As we mentioned in Section 1.4, our goal is to replicate as much as possible the behaviour of our Agent policy. The first thing we can appreciate is the similarity that the Graph and the Graph/Random policy to the Agent one, it seems that the Random policy has a tendency to have lower levels of health, which is completely understandable because it does not follow any strategy and there will be times where it we be moving in a close circle not picking up med-kits, therefore not gaining health. This is not a complete indicator that our extraction is working perfectly because med-kits spawn randomly and any behaviour is able to gain health, but it is a good start point. Now we will analyze some box plots representing the average distance to each wall through all the seed runs.

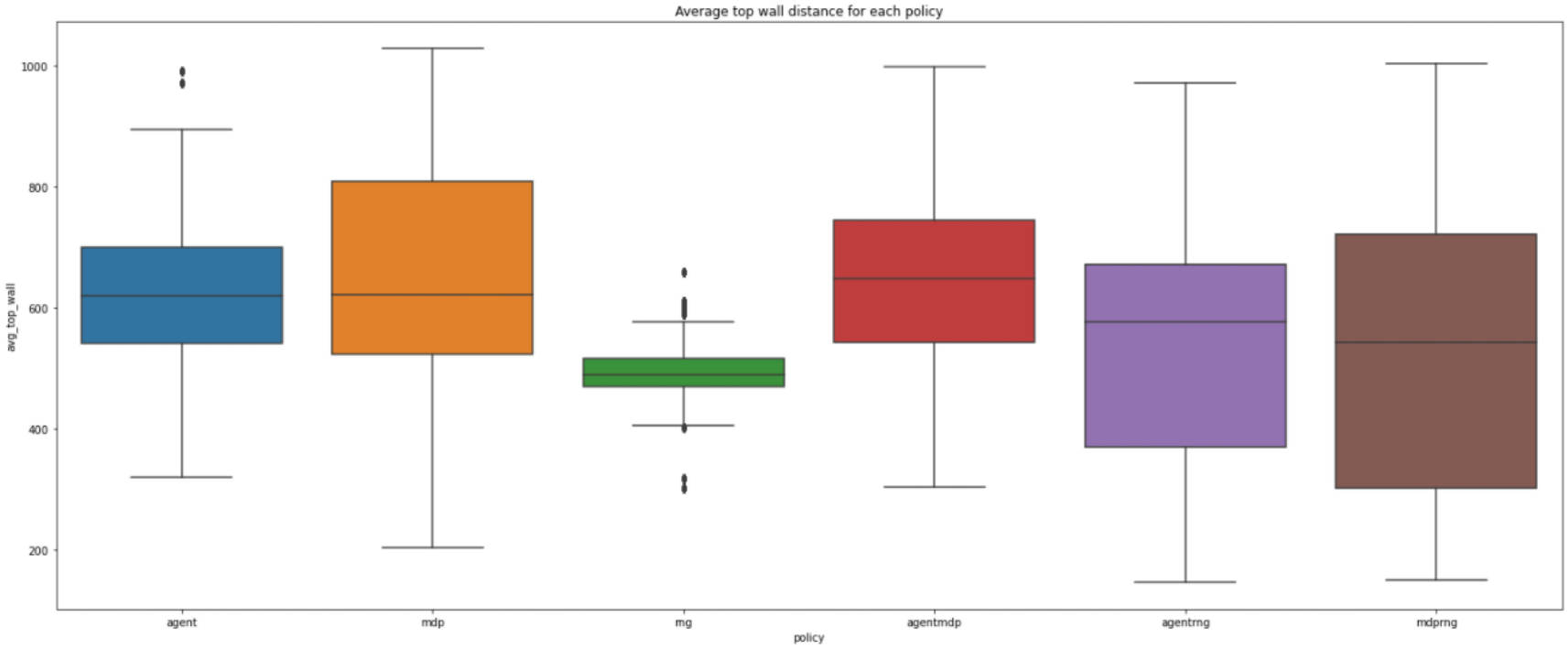


Figure 4.2: Each boxplot correspond to: Agent, Graph, Random, Agent/Graph, Agent/Random and Graph/Random/ respectively. Self made.

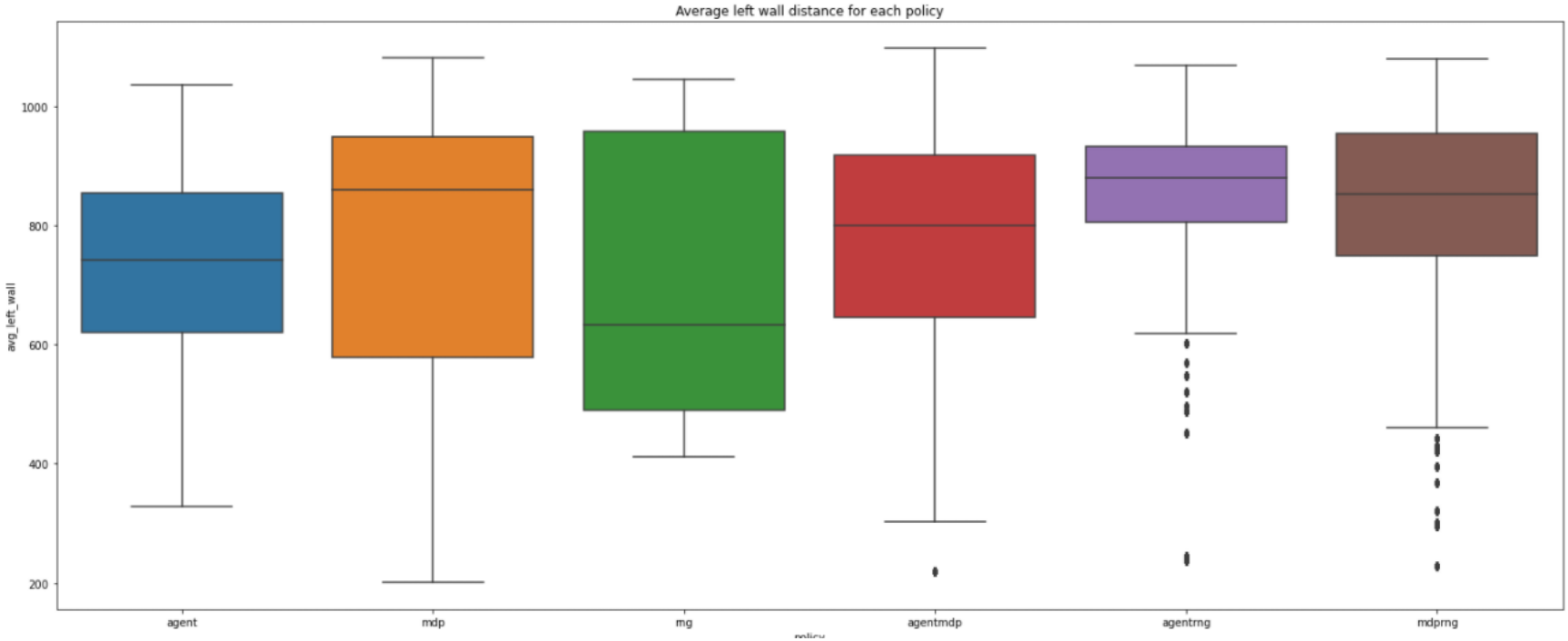


Figure 4.3: Each boxplot correspond to: Agent, Graph, Random, Agent/Graph, Agent/Random and Graph/Random/ respectively. Self made.

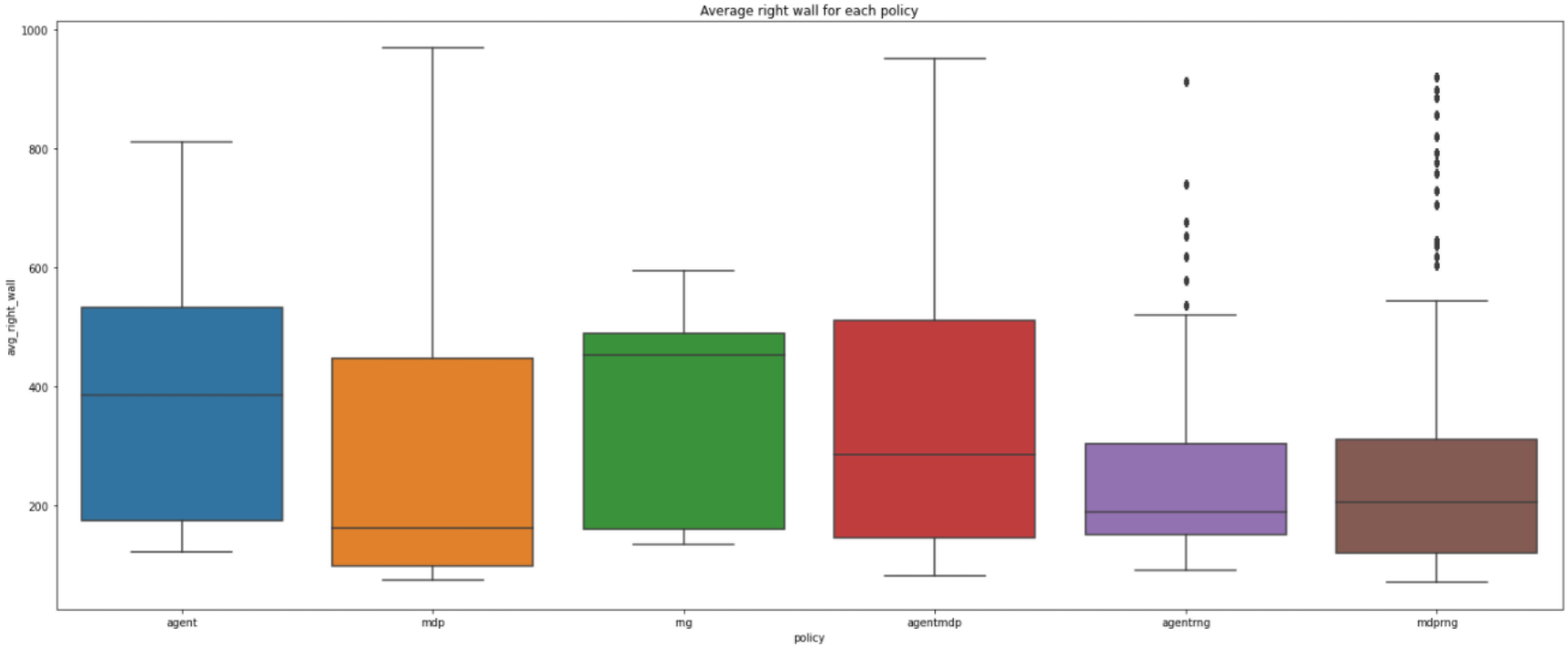


Figure 4.4: Each boxplot correspond to: Agent, Graph, Random, Agent/Graph, Agent/Random and Graph/Random/ respectively. Self made.

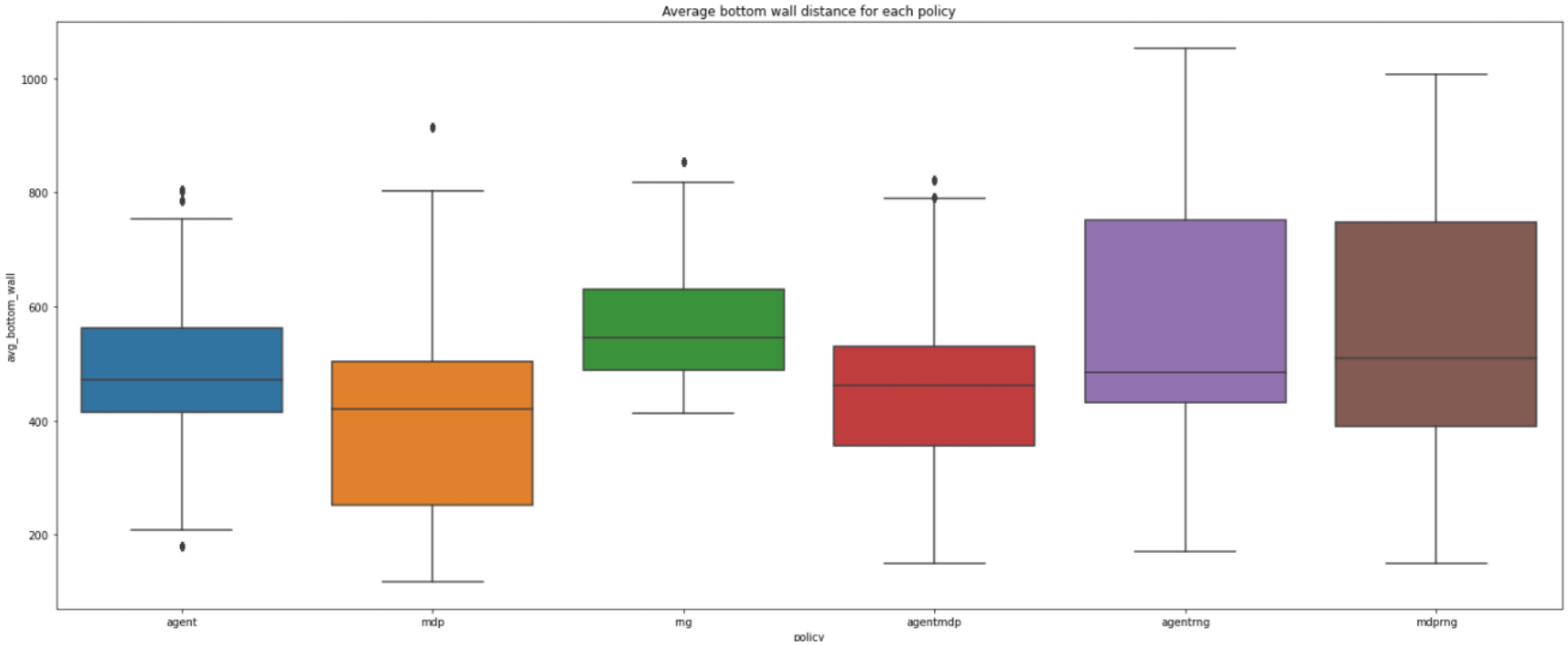


Figure 4.5: Each boxplot correspond to: Agent, Graph, Random, Agent/Graph, Agent/Random and Graph/Random/ respectively. Self made.

What we can appreciate from those box plots is that the Graph policy has the tendency to follow what the Agent policy is doing, which means our explainlability implementation may be extracting the behaviour in some way, of course is not perfect nor highly close but for what we have seen so far it is promising. Also, it looks like the Graph policy has the most impact when using the mixed policies, in most cases they have a similar boxplot.

4.2.2 Seed analysis

In this subsection we will analyse each set of runs made with the same seed, which means that every run has had the same initial condition and same spawn location for each med-kit. In order to do the comparisons scatter plots representing the average health with the average wall distance will be used.

First seed

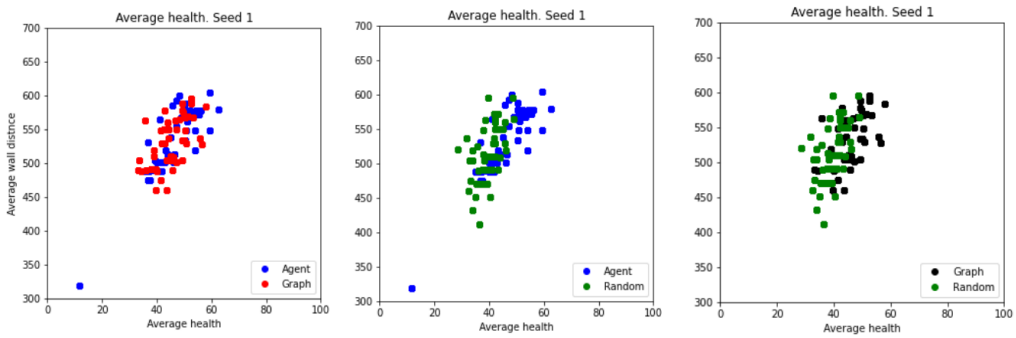


Figure 4.6: Scatter plots for Agent, Graph and Random policies. Self made.

In the first scatter we can check that the Graph policy is pretty similar to the Agent one, there are several points which diverge to lesser health with almost the same average distances, on the other side we can see how the Random policy is shifted to the left, meaning that it does not pick up that much med-kits, we can appreciate the same when compared to the Graph policy. One really interesting thing can be seen in the following scatter plot showing the results with the mixed policy Graph/Random.

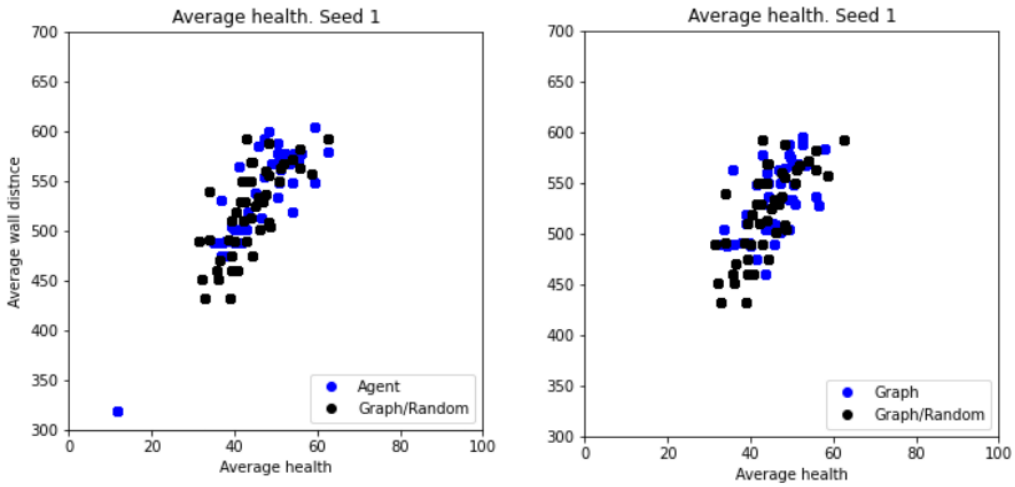


Figure 4.7: Scatter plots for Graph/Random policy. Self made.

The runs with this seed the Graph/Random policy has behaved closely to the pure Graph one, in the next seed evaluation we will see if this relation is maintained, in the box plots we could see that the average distances from walls was similar.

Second seed

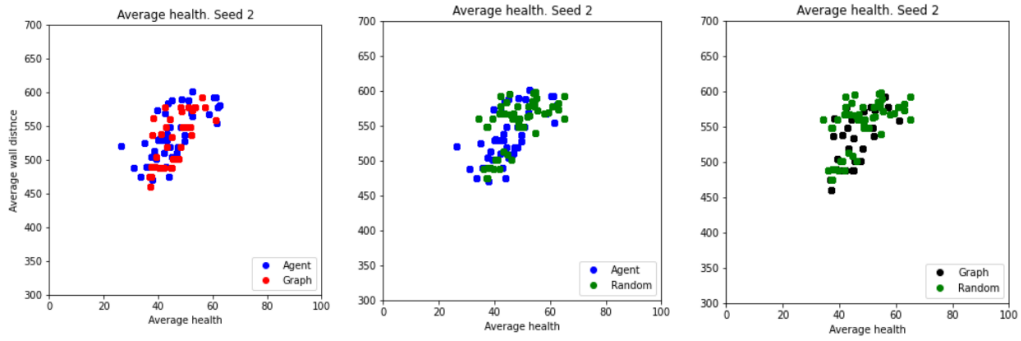


Figure 4.8: Scatter plots for Agent, Graph and Random policies. Self made.

In those scatter plots there is something interesting, in the case of the Agent and Graph policy we have a single group but on the contrary the Random

policy has two independent groups, which could mean that is has been running in close cycles in two different regions, rather than moving across the scenario most of the time.

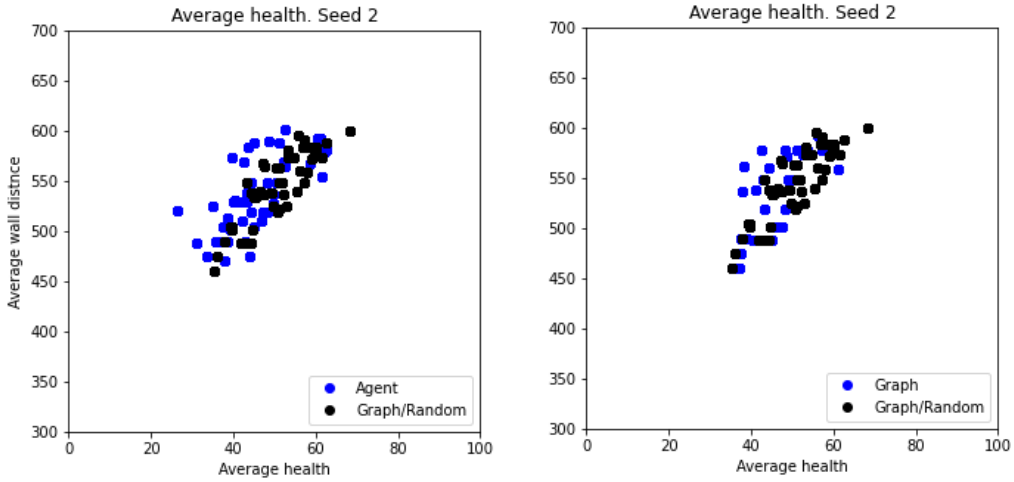


Figure 4.9: Scatter plots for Graph/Random policy. Self made.

Again, the Graph/Random policy seems to behave similar to the Graph and the Agent. In this case we can see that some of the points in the Graph policy have been shifted to the right, which could mean that the Random actions may have helped the Graph policy to get out from a stuck situation, for example walking straight into a wall or corner.

Third seed

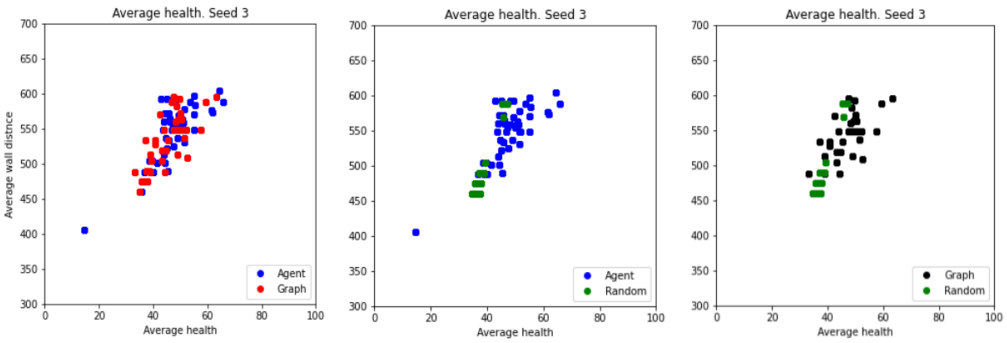


Figure 4.10: Scatter plots for Agent, Graph and Random policies. Self made.

In this seed execution we have a clearer difference between the Agent and Graph policies, similar to the previous seed, the Random policy has two differentiated groups meaning movement in a reduced area. Graph and Agent seem to be pretty similar again.

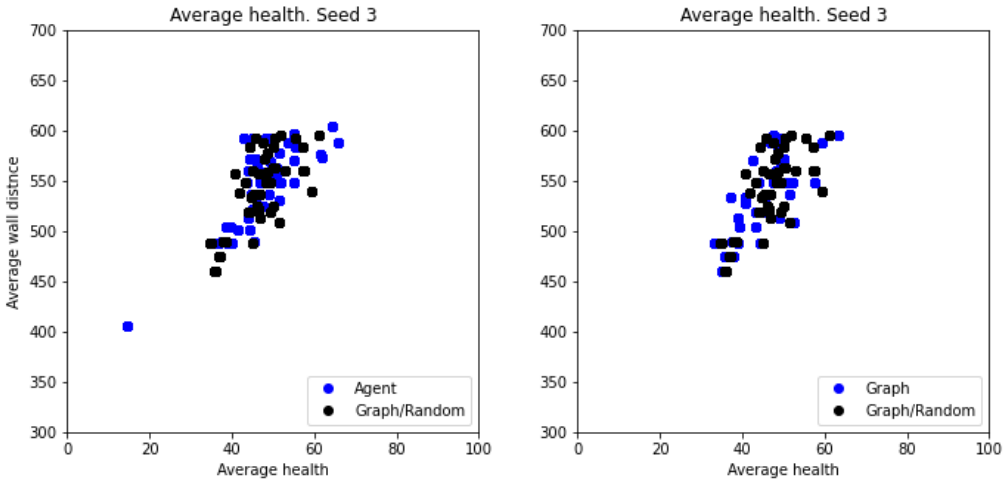


Figure 4.11: Scatter plots for Graph/Random policy. Self made.

Finally there are no huge differences between the Graph and Graph/Random policies, probably both could be decent approximations of the Agent behaviour.

4.3 Final thoughts

What we can understand from the obtained results is that our approach is decent enough to consider that we are able to extract the Agent policy behaviour. What has surprised us is the Graph/Random policy being able to keep up without much problems. We understand that the current state representation is not giving a clear extraction, but this happens because we are making the choice of the action proportional to the amount of times it has been used, so in some cases it will choose the action which is not optimal to mimic the Agent policy.

Chapter 5

Conclusions

AI is an area which is growing nonstop every day, and as time comes by there will be the need of understanding how those applications select or perform their actions which in most cases are left like black boxes that produce some output that works or is able to solve a problem efficiently enough. The work of explainability is to cast light on those black boxes and return feedback related to their behaviour.

In this project we are following the techniques used in [3] which consists on creating a Policy Graph based on some representation of the scenario/problem we want to solve, in the context of the project we have used the game ViZDoom [31] and the *health gathering* scenario, in the end there needs to be some expert in the topic of research to validate the results of the behaviour extraction.

5.1 Research questions

Relating to questions in Section 1.4, we think that we have achieved promising results: in Chapter 4 we have shown that our work is able to extract, to some extent and in a symbolic manner, the behaviour of a trained Agent. Of course, it is not perfect, probably due to the simplicity of state representation used in Section 3.5, but we do not have to forget that the environments coming form this game are really complex. With the results we have obtained we are pretty sure that keeping on track with this research is a great idea to explore the limits of explainability in ViZDoom environments, we have in mid to return to a previous state representation using vector representations and

also explore new environments such as *defend the line* which involves dodging enemy projectiles while at the same time the agent needs to shot and kill those enemies in order to win the scenario. Another scenario we would like to research on is the *deadly corridor* which consists on reaching a check point at the end of a corridor without dying, on the way to the checkpoint there are med-kits that restore health, shields that avoid direct damage to the health of the player and enemies which will try to kill our player. At the start of the project we defined some competences that would be covered by this work, now we will be talking about if we have fulfilled them. At first we should have analyzed the requirements of the project and select some platform to develop on, as programming language we selected python which is one of the lead languages when it comes to Neural Network programming and we have also used TensorFlow 2.0 library, it allowed us to use the full capabilities of our NVIDIA graphic card when training, which speed up the process a lot compared when the CPU is used to do the calculus required for training.

There are two other competences which involve the implementation of machine learning techniques, as it can seen in Chapter 3, we have created a Reinforcement Learning Agent by the use of a Convolutional Networks and Neural Networks, also there is a competence in which is required to implement some graphic application such as video games, in our case have not created the game on which we are researching but we are using direct images from the game as well as modified some of the code for the scenario in order to meet our requirements, and of course once the agent is trained we are making it play the game so we can extract data.

Finally there is one competence which is supposed to evaluate our capacity to extract or formalize the human knowledge in order to solve some problem. As the definition means our project is based on the extraction of the behaviour of a trained Agent, which could really be the behaviour of a human playing the game. We are trying to replicate it by using a representation that is human readable so we can truly understand if our assumption on how the trained Agent behaves is correct.

5.2 Experience used in the project

There are several subjects from the degree that have allowed us to finish this project, the first one is LP (Llengüatges de programació), in this subject we learned a lot of python and other several languages allowing a better adapta-

tion when learning how to use new libraries or learn new languages. Another subject has has helped with the analysis is APA (Apranentatge Automàtic), the final project of this subject consisted on creating a classification and regression system for a data-set of our choice, by the end of this project we learned how to analyse the data used as training and also understand the results, also we did en introduction to neural networks which was useful to have a big picture of what the project would be. There is another subject called CAIM (Cerca i anàlisi de dades massives) which has been useful to develop the vector based representation and similarity algorithm which will probably be used in future work due to its generalization and its ease when comparing similar states. Finally there is VC (Visió per computadors) which taught us what convolutions were and how they work thus making the process of using Convolution layers easily.

5.3 Experience gained from the project

Finally there are several aspects of the project that we would like to remark as they have been an expansion of the knowledge we previously had on some areas, such as neural networks, the work done in this project has allowed us to have a more deep knowledge on how Convolutional Networks and Neural networks work in a more real life scenario. Also it as been useful to learn how to analyse results in machine learning projects. Another huge lesson learned from the project is how subtle changes in the representation of the data makes the process of training more time and space consuming (see Section 3.4), making that some configurations could not converge fast enough or finish training due to lack of memory in the graphics card.

Bibliography

[1] *A European approach to artificial intelligence — Shaping Europe's digital future*. URL: https://digital-strategy.ec.europa.eu/en/policies/european-approach-artificial-intelligence (visited on 10/16/2021).

[2] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* 6.5 (1957), pp. 679–684.

[3] Antoni Climent Muñoz. "An application of explainability methods in reinforcement learning". In: (July 2020). URL: https://upcommons.upc.edu/handle/2117/335594.

[4] *Discord*. URL: https://discord.com/.

[5] Upol Ehsan et al. "Rationalization: A Neural Machine Translation Approach to Generating Natural Language Explanations". In: *AIES 2018 - Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society* (Feb. 2017), pp. 81–87. arXiv: 1702.07826. URL: http://arxiv.org/abs/1702.07826.

[6] *EUR-Lex - 52021PC0206 - EN - EUR-Lex*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1623335154975&uri=CELEX%3A52021PC0206 (visited on 10/16/2021).

[7] *File:MultiLayerNeuralNetwork english.png - Wikimedia Commons*. URL: https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetwork_english.png (visited on 10/16/2021).

[8] *Git Lab*. URL: https://gitlab.com/gitlab-org/gitlab.

[9] Bradley Hayes and Julie A. Shah. "Improving Robot Controller Transparency Through Autonomous Policy Explanation". In: *ACM/IEEE International Conference on Human-Robot Interaction*. Vol. Part F127194. New York, NY, USA: IEEE Computer Society, Mar. 2017, pp. 303–312. ISBN: 9781450343367. DOI: 10.1145/2909824.3020233. URL: https://dl.acm.org/doi/10.1145/2909824.3020233.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[11] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[12] Tom M Mitchell. "Machine learning and data mining". In: *Communications of the ACM* 42.11 (1999), pp. 30–36.

[13] *Precios de GPU — Documentación de Compute Engine — Google Cloud*. URL: https://cloud.google.com/compute/gpus-pricing (visited on 03/13/2021).

[14] *Regulatory framework on AI — Shaping Europe's digital future*. URL: https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai (visited on 10/16/2021).

[15] Cinjon Resnick et al. "Pommerman: A Multi-Agent Playground". In: *CoRR* abs/1809.07124 (2018). arXiv: 1809.07124. URL: http://arxiv.org/abs/1809.07124.

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Vol. 13-17-August-2016. Association for Computing Machinery, Aug. 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. arXiv: 1602.04938. URL: http://dx.doi.org/10.1145/2939672.2939778.

[17] *Rocket.Chat*. URL: https://rocket.chat/.

[18] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[19] Stuart J. Russell, Peter Norvig, and Ernest Davis. *Artificial intelligence: a modern approach*. 4th. Pearson, 2020.

[20] Mikayel Samvelyan et al. “The StarCraft Multi-Agent Challenge”. In: *CoRR* abs/1902.04043 (2019).

[21] Joseph Suarez et al. *Neural MMO: A Massively Multiagent Game Environment*. URL: https://openai.com/blog/neural-mmo/.

[22] *Sueldo: Junior Developer en Barcelona*. URL: https://www.glassdoor.es/Salaries/barcelona-junior-developer-salary-SRCH_IL.0,9_IM1015_KO10,26.htm (visited on 03/12/2021).

[23] *Sueldo: Project manager en Barcelona*. URL: https://www.glassdoor.es/Sueldos/barcelona-project-manager-sueldo-SRCH_IL.0,9_IM1015_KO10,25.htm (visited on 03/12/2021).

[24] *Sueldo: Qa-engineer-salary en Barcelona*. URL: https://www.glassdoor.es/Salaries/barcelona-qa-engineer-salary-SRCH_IL.0,9_IM1015_KO10,21.htm (visited on 03/12/2021).

[25] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. London, England, 2017. URL: http://incompleteideas.net/book/bookdraft2017nov5.pdf.

[26] *Trello*. URL: https://trello.com/.

[27] *ViZDoom/Types.md at master · mwydmuch/ViZDoom · GitHub*. URL: https://github.com/mwydmuch/ViZDoom/blob/master/doc/Types.md#gamestate (visited on 10/16/2021).

[28] *ViZDoom/Types.md at master · mwydmuch/ViZDoom · GitHub*. URL: https://github.com/mwydmuch/ViZDoom/blob/master/doc/Types.md#gamevariable (visited on 10/16/2021).

[29] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning*. 2016. arXiv: 1511.06581 [cs.LG].

[30] Paul J Werbos. “Applications of advances in nonlinear sensitivity analysis”. In: *System modeling and optimization*. Springer, 1982, pp. 762–770.

[31] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. “ViZDoom Competitions: Playing Doom from Pixels”. In: *IEEE Transactions on Games* (2018).

Appendix A

Planning

The length of this project is about four months, it started at the beginning of February and it is supposed to be completed at the end of June. The total amount of credits assigned to the project are 18 on which each credit is equivalent to 30 hours of work, so in the end it is supposed to last 540 hours which leads to 4.5 hours a day.

In the end problems have been faced so the defence of the project has been postponed to October

In this section we will have a look at the requirements, different tasks that will make up the project as well as the risk management and a Gantt chart.

A.1 Resources

We can divide the requirements into three groups, Human resources, Hardware and Software. Those are the following:

- Human: As for human resources we have the Director and Co-director which will take care of the organization such as deciding the steps to follow or control whenever a task is finished, also a programmer which will be in charge of all of the coding required and the document writing.
- Software: The software that will be used in this project are *Visual Studio Code* that will be the programming environment, *GitLab* as the control version manager, *Trello* as the organizing tool and finally for the communication *Rocket.Chat* and *Discord*.

- Hardware: The hardware that has been used for this project is a decent laptop which contains a GTX 1070 mobile. At first we thought this GPU wouldn't be able to handle the training process and we would require the use of TPU's provided by Google, but in the end the laptop has been enough, it has also been used for the writing of the document.

A.2 Task description

All of the tasks that made up the goals specified in the last chapter will be explained and a certain amount of hours will be assigned to each one along with an explanation about the time assigned and if it has any dependency with another task.

A.2.1 Project management [T1]

This task is one of the most important because it lasts the whole project, the first part of the task is done into a compulsory subject which is GEP, the goal of this subject is to plan the whole project beforehand so afterwards one can focus into the research or implementation of the project.

This task will be split into several sub-tasks, those are:

Context and Scope [T1.1]

It is required to give to the project a contextualization and an explanation of the terms that will be used in the project so the reader can understand what this project is about, the goal of the project, how the project will be organized and a justification. This was done in 20 hours.

Planning [T1.2]

We are asked to define the different tasks that make up the project along with an assignment of estimated hours and dependencies between them. It is very important because if it is well organized there should not be any major difficulty to complete the project in time. The time spent in this task has been 12 hours.

Budget and sustainability [T1.3]

In this case an analysis of the cost of the project will be made, in this case it is important because as powerful hardware will be required it may have a big impact on the budget. It is estimated that 8 hours will be enough to complete it.

Meetings [T1.4]

This subtask has been active throughout the project in order to check finished tasks or discuss different approaches to several problems, the meetings involved the director and the co-director, around 10 hours of meetings have been made.

Document Writing [T1.5]

Document writing is one of the most important tasks because the report has to be well written and structured. This task has been active the whole project, and we estimate that we have spent 90 hours.

Defense of the project [T1.6]

Once all the work is finished and the document has been written, there will be a presentation on which we have to defend and explain what we have achieved throughout the project, in order to prepare this presentation we estimate that 20 hours will be used.

A.2.2 State of the art and research [T2]

This task focuses on acquiring the desired knowledge in order to create an agent and apply explainability methods so we can investigate if similar results can be achieved.

First of all, many books and reports about different topics must be read, such as machine learning, reinforcement learning and Markov Decision Processes. It will also require us to make a research on different explainability methods in order to train a second agent. Finally we will have to look for an environment that suits our interests.

This task has been active until the implementation of the explainability techniques, at each step of the project we have had to look for information

in order to make sure everything is well coded. A total of 60 hours have been spent during the task.

A.2.3 Creation of an agent [T3]

Once the research on reinforcement learning and different methods to create agents finished it was the time to create the core of the project, the creation of the agent, this task can be split into two:

Coding the neural network [T3.1]

In order to code our neural network we decided to start with a base code which was provided by the developers of ViZDoom library, which was only able to solve a simple scenario, we had to invest a lot of time trying to understand how the code worked and also modify the neural network so it could be able to solve the scenario we had selected. Otherwise we would probably have to invest more than 50 hours which is the time we spent on this task.

Training the agent [T3.2]

At first, due to our lack of experience in this field we expected that 5 hours of training would be enough but in the end we have spent around 40 hours because of the different approaches

A.2.4 Generation of predicates [T4]

This task is very important because it will allow us to create one of the bases of the explainability process of the project which is the creation of a policy graph through instrumentation of the code, as there are many ways of doing this process it will require a lot of time to implement and test that it works correctly, this task lasted 200 hours.

A.2.5 Create an algorithm for the Policy Graph [T5]

In this task we will have to create an algorithm that allows obtaining an action from the policy graph by giving a state representation, at each iteration of the generation of predicates this algorithm needs to be modified, it has taken us around 180 hours.

A.2.6 Comparing the performance [T6]

This task focuses on obtaining data from the newly created agent from the explainability methods by creating a code which is able to extract variables that allow us to compare it to the original agent and achieve conclusions. We have spent around 40 hours in creating the code and the analysis.

ID	Task	Time	Dependency	Resources
T1	Project Management	162 h	-	-
T1.1	Context and scope	20 h	-	PC
T1.2	Planning	12 h	T1.1	PC
T1.3	Budget and sustainability	10 h	T1.2	PC
T1.4	Meeting	10 h	-	PC
T1.5	Document writing	90 h	-	PC
T1.6	Defense of the project	20 h	T6	PC
T2	State of the art and research	60 h	-	PC
T3	Creation of an agent	90 h	T2	PC, GitLab, Visual Studio Code
T3.1	Coding the neural network	50 h	T2	-
T3.2	Training the agent	40 h	T3.1	-
T4	Generation of predicates	200 h	T3	PC, GitLab, Visual Studio Code
T5	Create an algorithm for the MDP	180 h	T4	PC, GitLab, Visual Studio Code
T6	Comparing the performance	40 h	T5	PC

Table A.1: Distribution of tasks

A.3 Risk management

In this section we will explain the different problems that have occurred throughout the project and its implications.

At the start of the project we made a gantt chart in order to have an estimation of the duration of each task. In our previous planning we had almost two free weeks at the end in order to have some time to solve the problems we could encounter.

We expected that three tasks could deviate from our estimation, those are: the generation of predicates (T4), the creation of an algorithm for the graph (T5) and the training of a second agent.

In the end we realized that we could unite the training of a second agent to the comparison of the performance.

The remaining tasks have lasted at least 185 hours more which is longer than the almost 50 hours we had as spare time.

Down below we will explain what has happened with every task and also show the original gantt chart and an updated one.

- Generation of predicates (T4): This is the task that has hold us for longer, the reason behind this overtime is that whenever we try to extract a behaviour several iterations of representations are needed, also if we decide to give up on one behaviour because is not clear enough we need a completely different representation.

- Algorithm for behaviour graph (T5): This task is highly related to the previous one (T4). There are several ways of representing the state of the game in order to try to mimic a given behaviour, in our case we have tried a integer vector representations and string vectors representations. The advantage with integer vectors is that we can rely on one function to compare and look for similar states by using cosine similarity but when it comes to string vectors each time we change the strings or the structure we need to code a new comparison function. Using string representations makes it easier to understand what the agent is doing because we can create more specific states.

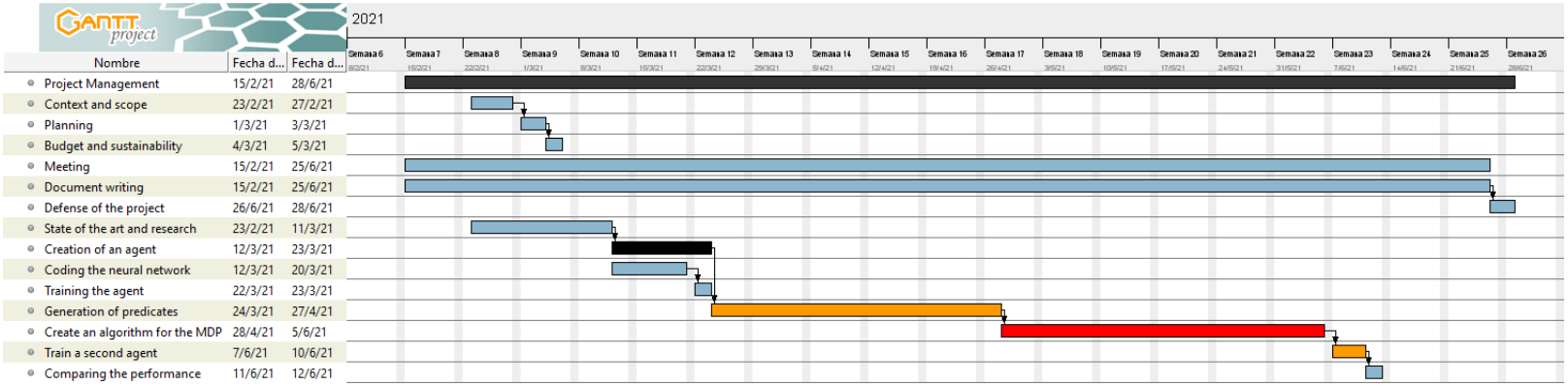


Figure A.1: Old Gantt diagram. Self made with GanttProject

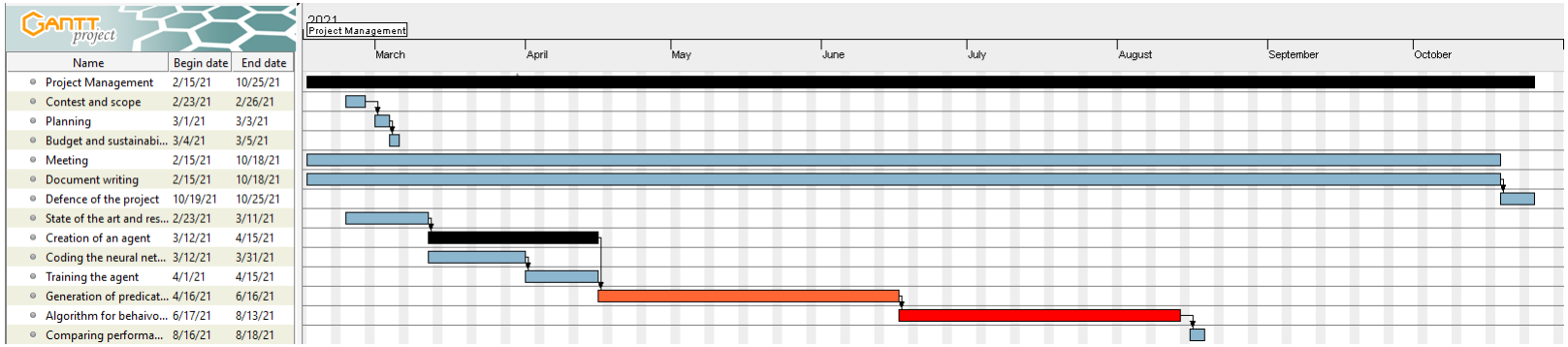


Figure A.2: New Gantt diagram. Self made with GanttProject

Appendix B

Budget

In this chapter we will discuss the cost or budget that would be required for the realization of the project. Costs can be split in several sections, staff costs, amortization, contingencies and incidentals.

B.1 Staff costs

First of all we must present the different roles that will be necessary for this project. A **project manager** is responsible for the planning of the project, also a **developer** will be required because the core of the project is to create a reinforcement learning agent so programming will be essential, finally the last role would be a **tester** so it can verify the correctness of the agent. Due to the pandemic the use of a work space won't be necessary for safety reasons, we specified in *section 2.5* that weekly meetings would be done to control the flow of the project so all of work can be done from home.

Role	**Salary**	**SS**	**Cost**
Project manager	20.26	6.08	26.34
Developer	12.2	3.66	15.86
Tester	16.1	4.84	20.94

Table B.1: Table of the staff salary. Salary from [23][22][24]

ID	Task	Time	Proj. Manager	Developer	Tester	Cost
T1	Project Management	162 h	162 h	-	-	**4214.4€**
T1.1	Context and scope	20 h	20 h	-	-	526.8€
T1.2	Planning	12 h	12 h	-	-	316.08€
T1.3	Budget and sustainability	10 h	10 h	-	-	210.72€
T1.4	Meeting	10 h	10 h	-	-	263.4€
T1.5	Document writing	90 h	90 h	-	-	2370.6€
T1.6	Defense of the project	20 h	20 h	-	-	526.8€
T2	State of the art and research	60 h	5 h	50 h	5 h	**1029.4€**
T3	Creation of an agent	90 h	-	85 h	5 h	**1437.56€**
T3.1	Coding the neural network	50 h	-	55 h	-	793.0€
T3.2	Training the agent	40 h	-	30 h	5 h	644.56€
T4	Generation of predicates	200 h	-	180 h	20 h	**3273.6€**
T5	Create an algorithm for the MDP	180 h	-	165 h	15 h	**2931€**
T6	Comparing the performance	40 h	-	40 h	-	**634.4€**
	Total Cost	732 h	167 h	520 h	45 h	**13520.35€**

Table B.2: Final task cost. Self made.

B.2 Hardware costs

As we specified in the previous section we have been using a decent laptop in order to code, train and write the report of the project, the cost of this laptop was 1400€. As hardware cost we will have to calculate the amortization of the laptop.

B.3 Amortization

In our case the amortization comes from the hardware used, as we are renting a service in order to train the only amortization is the laptop used for programming and writing, also as the software we are using is open source there is no amortization. So the formula for calculating the amortization is the following.

$$Amortization = productPrice * \frac{monthsUsed}{expectedLifeTime}$$

So the amortization will be $1400 * \frac{8.5}{40} = 297,5$€.

Source	Cost
Laptop	297,5€
Total	**297,5€**

Table B.3: Hardware costs. Self made

B.4 Contingencies

Throughout a project unexpected events may appear and those events can have an impact in the cost, having a contingency budget is necessary to cover them in case of need. A common value used in software projects is a 15% of the total cost. This is the value we calculated at the start of the project with an estimated cost of 10803.51. **Contingency cost** = 10803.51 * 0.15 = **1620.53€**

B.5 Incidentals

In section 3.3 we specified two different risks we could face on the project, also we had almost two weeks which corresponds to 54 hours of work, in one hand if programming bugs or delay appear the developer would take those hours as work, on the other hand training times may vary if the results are not what we desire, due to renting this can have an impact on the budget. We estimate that we have a high chance that more training will be necessary and less likely more hours of programming.

Problem	**Hours**	**Price**	**Chance**	**Cost**
More training	35 h	73.5€	50%	36.75€
More programming	40 h	634.4€	30%	190.32€
Total				**227.07€**

Table B.4: Cost of the possible incidentals. Self made.

B.6 Total cost

In the chart below we have a summary of all of the costs calculated previously

Source	**Cost**
Staff cost	10115.72€
Hardware cost	119.05 €
Contingencies	1620.53 €
Incidentals	227.07 €
Total	**12082.37 €**

Table B.5: Hardware costs. Self made.

B.7 Management control

Once the budget has been calculated we must have a form of controlling the budget and amortization deviations, in order to do so after a project stage has been completed we will calculate both deviations.

In order to calculate the deviations we just need to subtract the real cost of a task from the estimated, if we get a positive result we are below the estimated meaning we have spare budget that can be assigned to troubleshooting but in case the result is negative we will have to take funds from the contingency budget.

To calculate the real cost at each stage we will have to check if any price has changed and re-calculate.

As we showed in the previous chapter we had to face some delay with the most critical tasks, now we will calculate how much we have had to spend in order to complete those tasks and check if our contingency plan has covered this expense.

At this point we have deviated 135 hours from the original planning, 35 from those hours have come from training. The average price for each kWh is around 0.23 €. The power draw of the laptop when training is close to 230 Watts, so cost of more training is $35 * 0.23 * \frac{230}{1000} = 1.85$€, this cost is almost negligible

Now the remaining 100 hours are from our developer, at the time of writing the average wage has not changed.

The cost of more hours for the developer is $15.86 * 100 = 1586$€. In the end we have to add 1587.85 €to the total cost of our project, because we set a 15% as a contingency which is equivalent to 1620.53€, which means we are still within the budget we calculated.

Appendix C

Sustainability

Before doing the survey I already knew the existence of the three different dimensions that make up sustainability, environment, social and economic, but my knowledge was not as high as expected regarding the social dimension.

While the project was being organized one of the goals was to optimize the resources that could be used which has relation with the environment and the economic dimensions.

C.1 Environmental dimension

In regards to the environmental dimension the computer we are using for the writing and coding is one we already had so there is no more potential electronic waste, also because the computer is a laptop its power consumption is lower than a desktop so the amount of electricity consumed is lower.

One big impact that this project may have on the environment is the computation required to train the agents that will be created. In order to train those agents high end hardware must be used and this kind of hardware requires a huge amount of energy so our effort will be to invest more time in programming so less training time will be required.

C.2 Social dimension

Since the beginning of the degree I knew the existence of artificial intelligence which was a topic which had my attention and when i discovered what machine learning was it fascinated me, the capability to generalize a task with

a series of calculations made me want to take the computation mention and to look for a project within this area. I think this is a great opportunity to learn from talented people that work in this field.

As for the utility that this project can offer, once it is finished if positive results are achieved it will allow researchers to understand the behaviour of their agents and networks making possible the understanding of different strategies in tasks that we assumed that could only be solved in a specific way.

C.3 Economic dimension

In my opinion the calculation of the budget has been as realistic as possible, the only problem we may face are the training hours. I think that i underestimated them a bit, but nothing else from there.

As this is a research project and an extension of a PhD project it is quite hard to estimate the economic impact it may have. One benefit of knowing the behaviour of an agent or network could be that once a similar task has to be faced there is no need to invest resources into solving the problem from scratch.

The major problem regarding this type of project is that is very hard to improve the economic issues because improving the training speed which would involve less economic resources implies using better hardware which is more expensive and less energy efficient

Appendix D

Laws and regulations

In this section we will be explaining if our project may have a conflict with some regulations or laws.

In April 2021 the European Commission posted a document which is a proposal for regulating the use of high risk AI systems in the European region [1][6], even if those regulations are not in force we should take a look at them to check if our project could be under those regulations.

The EU has four categories for the risk classification of AI systems, unacceptable, high, limited and minimal [14].

For a system to be classified as high risk it needs to be used in the following cases:

- Critical infrastructures: cases where the life or health of humans could be affected
- Educational or vocational training: cases where the system could interfere with the access to studies or formations
- Safety components of products
- Employment procedures
- Essential private and public services
- Law enforcement that may interfere with people's fundamental rights
- Migration, asylum and border control management
- Administration of justice and democratic processes

The requirements for being a limited risk system is if the machine is interacting directly with its users, in this case they must be notified.

Finally there is the group where our project belongs, the minimal risk, to be in this group a system must not be used in any of the previous situations, in the case of being minimal the project must obey the Charter of Fundamental Rights and the General Data Protection Regulation.