Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

FINAL DEGREE PROJECT

TITLE: Aircraft-to-Aircraft separation based on Reinforcement Learning

DEGREE: Aeronavigation Engineering

AUTHOR: Weronika Prawda

ADVISOR: Cristina Barrado Muxí

DATE: September 5th, 2022

Title: Aircraft-to-Aircraft separation based on Reinforcement Learning

Author: Weronika Prawda

Advisor: Cristina Barrado Muxí

Date: September 5th, 2022

Abstract

Air traffic has been increasing and with it the workload of air traffic controllers. Despite the pandemic, the latest figures show a rapid recovery and forecast exponential growth. This indicates the need to modernise air traffic control and the technology used, which is already being developed and implemented by organisations like SESAR, like applying AI to air traffic control (DART). A support tool with automatic conflict avoidance would be a great step to address the problem of possible overcapacity of air traffic controllers.

This document describes two possible implementations of a conflict avoidance tool. The approach is to use Deep Reinforcement Learning to select actions that avoid conflict and help the air traffic controllers to take faster and better decisions. The basis for both approaches is a simple 2D airspace simulator and the same policy applied to all the aircraft.

The first proposal is a stand-alone DQN algorithm (DRL) that has a 7.06% improvement in the number of simultaneous conflicts compared to the original environment without applying a policy.

The second approach is a DQN algorithm that incorporates transfer learning of the rules of the air, and it is called by the acronym DRLT. It resulted in a degradation compared to the original environment, with a 6% increase in unremembered conflicts.

Nevertheless, Deep Reinforcement Learning has shown a decrease in decision time and the idea of reusing the same strategy for all aircraft has solved the problem of unpredictability issue that some reinforcement learning solutions had. The proposal could be a good start for a self-separation tool for unmanned aircraft but still needs future improvements in results. It is not suitable for air traffic controllers or piloted vehicles due to the increased workload it would suppose.

Título: Separación de aeronave a aeronave basada en el aprendizaje por refuerzo

Autor: Weronika Prawda

Tutor: Cristina Barrado Muxí

Fecha: 5 de Septiembre 2022

Abstract

El tráfico aéreo ha ido en aumento y con él la carga de trabajo de los controladores aéreos. A pesar de la pandemia, las últimas cifras muestran una rápida recuperación y pronostican un crecimiento exponencial. Esto indica la necesidad de modernizar el control del tráfico aéreo y la tecnología utilizada, que ya está siendo desarrollada e implementada por organizaciones como SESAR, por ejemplo, la aplicación de IA al control del tráfico aéreo (DART). Una herramienta de apoyo con prevención automática de conflictos sería un gran paso para abordar el problema de la posible sobrecapacidad de los controladores de tránsito aéreo.

Este documento describe dos posibles implementaciones de una herramienta para evitar conflictos. El enfoque es utilizar el aprendizaje por refuerzo profundo para seleccionar acciones que eviten conflictos y ayuden a los controladores de tránsito aéreo a tomar mejores y más rápidas decisiones. La base para ambos enfoques es un simple simulador de espacio aéreo 2D y la misma política aplicada a todas las aeronaves.

La primera propuesta es un algoritmo DQN (DRL) autónomo que tiene una mejora del 7,06 % en el número de conflictos simultáneos en comparación con el entorno original sin aplicar ninguna política.

El segundo enfoque es un algoritmo DQN que incorpora transferencia de aprendizaje de las reglas del aire, y se denomina por el acrónimo DRLT. Resultó en una degradación en comparación con el entorno original, con un aumento del 6% en los conflictos no recordados.

Sin embargo, el aprendizaje por refuerzo profundo ha mostrado una disminución en el tiempo de decisión y la idea de reutilizar la misma estrategia para todas las aeronaves ha resuelto el problema de imprevisibilidad que tenían algunas soluciones de aprendizaje por refuerzo. La solución podría ser un buen comienzo para una herramienta de autoseparación para aeronaves no tripuladas, pero aún necesita futuras mejoras en los resultados. No es apta para controladores aéreos ni para vehículos pilotados por el aumento de carga de trabajo que supondría.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

$\boldsymbol{a}$	action
$\underline{\boldsymbol{a}}$	joint action: array or tensor of actions from all the agents in a multi-agent environment.
$\boldsymbol{A}, \boldsymbol{A}(\boldsymbol{s})$	action space: set of actions that an agent can take
$\boldsymbol{d_{alert}}$	alert detection range
$\boldsymbol{d_{CPA}}$	distance in the closest point. Distance between two aircraft when they reach the closest point of approach
$\boldsymbol{d_{ij}}$	distance between two flights
$\boldsymbol{d_{min}}$	conflict detection range
$\boldsymbol{d_{target}}$	distance to the target
$\boldsymbol{e}$	experience: a tuple of state, reward, next state and done
$\boldsymbol{G}$	goal function. also called cumulative reward
$\boldsymbol{O}$	probability of an observation in a Partially Observable Markovian Decision Process
$\boldsymbol{P}$	matrix of transition probabilities
$\boldsymbol{p}$	probability
$\boldsymbol{r}$	reward
$\boldsymbol{R}$	reward function
$\boldsymbol{s}$	state
$\boldsymbol{s'}$	next state
$\underline{\boldsymbol{s}}_t$	joint state: array or tensor of states from all the agents in a multi-agent environment.
$\boldsymbol{S}$	observation space: set of all the states an environment can have.
$\boldsymbol{t}$	time step
$\boldsymbol{t_{CPA}}$	time to the closest point. Time left to reach the closest point between two aircraft.
$\boldsymbol{T}$	maximum number of time steps in an episode.
$\boldsymbol{V}$	value function
$\boldsymbol{v_i}$	Airspeed of flight i.
$\boldsymbol{v_{r_{ij}}}$	relative airspeed between two aircraft, normalised with maximum airspeed in the environment.
$(\boldsymbol{x_E}, \boldsymbol{y_E})$	entry point of the aircraft
$(\boldsymbol{x_i}, \boldsymbol{y_i})$	position of the aircraft in a 2D Euclidean space.
$(\boldsymbol{x_T}, \boldsymbol{y_T})$	exit point of the aircraft
α	learning rate
$\boldsymbol{\beta_{ij}}$	relative angle between the heading of the ownship and the intruder

γ	discount rate
δ	drift
ε	epsilon or exploration rate. Controls the balance between exploration and exploitation probability during the training of a model.
θ	weights of the neural network
μ	behavioural policy
ν	Value added to the airspeed in the corresponding action.
Π	target policy
π	policy
π_*	optimal policy
χ_i	track
$\boldsymbol{\psi}$	bearing
AI	Artificial Intelligence
ATC	Air Traffic Control
ATCO	Air Traffic Controller
ATM	Air Traffic Management
CPA	Closest Point of Approach
CUDA	Compute Unified Device Architecture
DART	Data-Driven Aircraft Trajectory Prediction Research
DRL	Deep Reinforcement Learning
DRLTL	Deep Reinforcement Learning with Transfer Learning
DQN	Deep Q Networks
GPU	Graphics Processing Unit
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
ML	Machine Learning
MRP	Markov Reward Process
RL	Reinforcement Learning
RoA	Rules of the Air
SESAR	Single European Sky ATM Research
SI	International System of Units
TD	Temporal Difference
TL	Transfer Learning
UAV	Unmanned Aerial Vehicle

INTRODUCTION

The air traffic is slowly but surely recovering from the 2019 health crisis impact [3] that interrupted its exponential growth. Its serious increase not only creates a challenge for the airports, like recently where airports were forced to implement capacity restrictions and demand to the airlines to stop selling tickets [13], but also puts pressure on the Air Traffic Management (ATM) and Air Traffic Control (ATC), with excessive concentration of traffic in some sectors.

Moreover, aviation is expected to change into a more sustainable model, which is hard when the traffic increases and the emissions also increase. That is why there's a need for a more efficient air traffic control system that could optimise the trajectories of the aircraft while dealing with risen demand.

That leads to the necessity of a modernisation of the ATC systems to be prepared for what the future holds. SESAR Joint Undertaking was established with this goal, and by uniting the most important European organisations intends to integrate the EU ATM systems. It is a complete pipeline of research, development and implementation of the newest technology for traffic management.

SESAR has proposed many solutions to be able to lead with this overwhelming forecast, focusing on complexity management, automation and enhanced airspace management [21]. For example, the DART project that started in 2017 and applied machine learning methods for aircraft trajectory prediction to increase predictability and faster decision making.

The response to the limited cognitive capacity of humans when processing copious amounts of information related to ATC could be artificial intelligence (AI), like the DART project proved.

From the field of Artificial Intelligence and Machine Learning, 3 types of learning are available. The first one is supervised learning, which learns from a given dataset and can predict these same variables later. The second one is unsupervised learning, used to segment and group data, detecting similarities and differences. And lastly, reinforcement learning (RL) which works with no previous data and by a punishment-reward system learns a policy [26].

What got reinforcement learning its popularity was a paper by a company called Deepmind [16], that together with the increase in computational power and appearance of open-source software, RL achieved extraordinary results in many fields.

Artificial intelligence methods have been making its way to the aviation industry, with organisations like Eurocontrol organising challenges to implement RL to aircraft conflict avoidance, which is the origin of this final degree project. The use of RL for aircraft separation is a widely studied subject, but it comes with a lot of challenges: complexity of the traffic, scalability, reliability, safety... To add

up, most of the papers are simplified scenarios so there is no real applicable solution that exists nowadays.

The aim of this project is to develop a RL model for in-cruise aircraft conflict avoidance in an airspace sector. This tool, which would ideally be used as an auxiliary tool for ATCs or as a self-separation system, would assess the best decisions needed to be taken in order to preserve the safe separation distance with other aircraft.

This project has two parts, the first one being the development of a policy that would direct the aircraft with conditional statements following the Rules of the Air, developed by Lidia Fuentes Coll [6], and a second part where this experience is taken to train a neural network in an intent to further improve the results.

The project is divided into six chapters that describe the concepts necessary to understand the process and the results of this work. The outline of the document goes as follows:

- Chapter 1: Reinforcement Learning

  This section introduces the mathematical and conceptual basics to understand the principles of reinforcement learning mentioned and used in this project.

- Chapter 2: Deep Reinforcement Learning

  This chapter explains how neural networks fit into reinforcement learning and describes the DQN algorithm.

- Chapter 3: Tools, Methods, and Metrics

  The programs, framework and tools used to develop the projects are stated, in addition to the details of the code and the metrics used to monitor and measure success in this project.

- Chapter 4: Results

  The final simulation parameters that were used and the results of the project accompanied by graphs and comparisons are described in this part of the document.

- Chapter 5: Conclusions

  The final evaluation of the project is done while revisiting the initial objectives as closure of the document.

CHAPTER 1. REINFORCEMENT LEARNING

1.1. Introduction

Reinforcement learning is an extensive subject, thus in this document only the most important concepts to understand the development of the project are described. However, for a more in-detail introduction to reinforcement learning, books like [14] or [22] are recommended to the reader as a good starting point.

1.2. Agent-Environment Dynamics

Reinforcement learning consists of an interaction between two elements: the *agent* and the *environment*. An agent is the component that makes the decisions whereas the environment is the domain that the agent interacts with. The environment has a set of rules or kinematics that the agent cannot control, but it can act in the environment and change its status. The main goal is for the agent to learn how to take good actions and reach an objective.

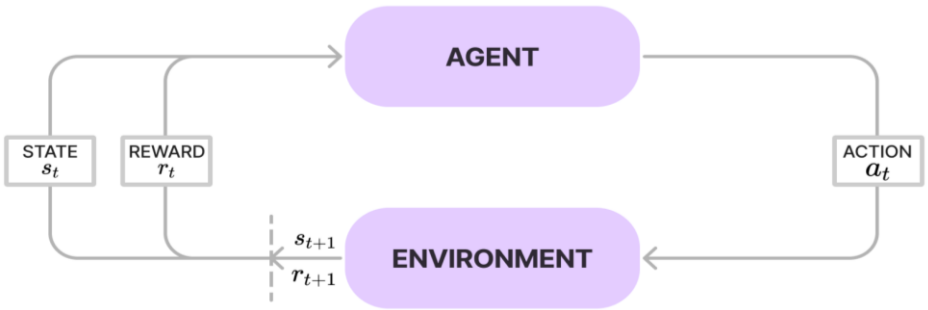

Fig. 1.1 Agent and environment interaction.

The state s_t or observation is a parameter about the status of the environment and the agent itself. The agent bases its decisions on the current state, contained in a state space S, and chooses an action from a set of actions allowed, called the action space $A(s_t)$. After acting on the action chosen in the current timestep t, a transition takes place, which makes the timestep increase to $t + 1$. The response of the environment is a next state s_{t+1}, caused after taking the action, and a reward r_{t+1}, which is telling the agent the quality of the decision that was just taken. The objective is to maximise the reward.

The choices taken are based on a policy π_t, which is a strategy to reach the goal. It ties the elements together and gives the agent a probability to choose a

certain action, based on the state and the next state provided $p_{ss'}(s_{t+1}|s_t, a_t)$ [15].

This process of interaction is represented in figure 1.1 where given a state an agent takes an action, and the environment sends the reward and the next state as feedback. All the process takes place in a timestep t and is repeated until a certain number of steps or a determined threshold is reached.

In multi-agent systems, where the environment interacts with more than one agent, there are different approaches to this interaction, which are going to be addressed in appendix A.

1.3. Markov Decision Processes

Decisions in stochastic learning environments can be formalised using Markov Decision Processes (MDP) [22].

MDP are based on the Markov Property, which states that the future states cannot depend on the past given the present observations.

Then, random states that satisfy the Markov property can be chained together into a sequence, called a Markov Process. Each state has a probability of transitioning into another state. The matrix of probabilities and the states can describe the dynamics of the environment in its entirety. Formally, a Markovian Process is a tuple with a finite set of states S and a matrix of transition probabilities. In equation 1.1 the state transition probability is formalised, and when computed for every combination of states can be joint into a matrix of transition probabilities (see equation 1.2).

$$p_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s] \quad \textbf{(1.1)}$$

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ & \cdots & \cdots & \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} \quad \textbf{(1.2)}$$

In order to get closer to reinforcement learning, it is necessary to define the reward in these processes. That's where the **Markovian Reward Processes** (MRPs) [22] come in. To add a value judgement to the Markovian Process, a reward function and a discount factor are added to the tuple. For each state of the agent, a reward function defines a numeric reward value, and the discount factor determines the importance of the immediate or future reward. Its value ranges from 0 to 1, with 0 caring only for the immediate reward and 1 caring only for the future reward. The discount factor varies according to the final goal of the reinforcement learning model.

The discount factor and the immediate reward function are related as seen in equation 1.4.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \textbf{(1.4)}$$

Where G is the goal function, also known as the cumulative reward, R is the reward function and γ is the discount factor. In the final expression, R_{t+k+1} is the reward received by the agent at timestep k by moving to the next state.

To go one step further, there is a need to define the agent that takes the decisions. The **Markovian Decision Processes** [22] adds a set of actions A for the agent to choose from, to the Markovian Reward Process. For this reason, the transition probability (1.5) and the reward function (1.6) have a slight change, as the actions are added to the equation.

$$p_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a] \quad \textbf{(1.5)}$$

$$r_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \quad \textbf{(1.6)}$$

Depending on the current state, a policy will dictate what actions the agent takes.

1.4. Value Functions

The grand majority of RL algorithms help to determine the next best action by an estimated value function. A value function relates the actions and the state an agent is in, to approximate the return that the agent will get in the future. Each policy has its own value function. As mentioned before, the policy gives a probability of choosing an action for each state.

A value function determines how good the state is, given the policy. Mathematically, the expression looks like the one in equation 1.7.

$$V_\pi(s) = E_\pi[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_t = s] \quad \textbf{(1.7)}$$

This is also called the state-value function.

Similarly, an action-value function (see equation 1.8) can be defined, where the output is the value of an action taken under a policy, in a state s.

$$Q_\pi(s,a) = E_\pi[\sum_{t=0}^{T-1} \gamma^t r_t \mid s_t = s, a_t = a] \quad \textbf{(1.8)}$$

1.5. Bellman's Equation

Richard Bellman took the sum of future rewards and broke it into the immediate current reward and the future rewards, pointing out the recursion of the value function and summed this property in the famous Bellman's equation [1]. The Bellman's optimality equation expresses that for a policy π, the value of a state can be determined by the expected reward of the next state and the value of the next state with a discounted factor, and all of that summed over the policy probabilities.

This concedes to optimising the value functions locally, putting it together and still have an optimal result. Once the function is optimised, a maximised policy can be extracted from the values. The objective was to have all the sub-problems and its value functions in order to reach the best policy. This concept gave way to dynamic programming and lastly to reinforcement learning.

Ultimately, there's a need of approximating this value function from the experience of the agent. That's when Monte Carlo methods are used to determine the average value functions for each state and each action, by taking random actions for each state and memorising the return. However, this approach is only good for simple problems, and the functions are parametrized and adjusted in order to match the output. That's why Temporal Difference methods are used instead, which will be explained in detail in section 1.6.1.

The final goal is to optimise the policy and to maximise the value function. The value function for a policy π at a given state s is the expected cumulative reward discounted by the factor gamma [22].

Assuming that the policy is the optimal policy, the value function is converted to equation 1.9.

$$q_*(s,a) = E[R_{t+1} + \gamma max_{a'} q_*(s',a')] \quad \textbf{(1.9)}$$

To find the optimal policy two algorithms can be used. On the one hand exists the value iteration algorithm, which consists of optimising by iterating on the value function, maximising given an action a. The function needed to maximise is the future reward. Value iteration assumes that the value of the next state is known and relies on the Bellman's optimality equation.

On the other hand, policy iteration is a two-step iteration process that locks in a policy, and iteratively updates the value. Once the average value is fixed, the policy is updated. Once the policy is improved, the steps are repeated.

As a result of combining the ideas of the state-value (equation 1.7) and the action-value (equation 1.8) functions, where the result is an expected return given an action a and a state s, the equation in 1.10 is obtained.

It is called the Quality function or Q-function [1]. This function has the next state return implicit in the function, so it allows a policy-free learning where these next states are unknown. The result values of this function are called Q-values.

In this equation R_{t+1} is the received reward and $argmax_{a'}Q(s', a)$ finds the value of a that gives the maximum value to the function $Q(s', a)$, and then this value is put into the $Q(s', a)$ function, discounted by a factor γ. As pointed out before, there's a clear recursion in the function.

$$Q(s, a) = R_{t+1} + \gamma Q(s', argmax_{a'}Q(s', a)) \quad \textbf{(1.10)}$$

1.6. Q-learning

Q-learning is a RL algorithm based on value iteration. The algorithm is implemented by creating a table that related the pairs of state and actions with a reward. Thus, when a Q-table is initialised, the agent chooses and performs the actions, the reward is computed and the cell corresponding to that state and action is updated with the new value of the reward.

This algorithm is an off-policy method, meaning that the optimal policy is unknown, so two policies are used.

The first policy, called the local policy μ, is used to learn and explore, and the second one, called the target policy π, is updated less, based on the first one. In the end, the target policy becomes the optimal policy.

Q-learning is based on the Q-function, and uses temporal difference to estimate the expected value of an action a in a state s. As it is just a temporal difference method applied to the Q-function, first the work describes the TD method to better understand Q-learning.

1.6.1. Temporal Difference Methods

As the name suggests, temporal difference (TD) methods consider the time in which the agent takes the actions. Opposed to Monte Carlo methods, TD methods sustain that there is a correlation between the reward and the time in which an action was performed.

This procedure adds a correction term to the value function, which instead of averaging all the trajectory, giving the same importance to each step as in a Monte Carlo method, gives more weight to recent experiences. Using the temporal difference means that the value function can be approximated even if the sequence of events, also known as the episode, is not yet finished.

Hence, to formulate the update of the value function using TD methods mathematically, the expression from equation 1.11 is used.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad \textbf{(1.11)}$$

The part $[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ is what is known as the TD error, which is the difference between the discounted approximated value of the next state $\gamma V(s_{t+1})$ and the value of the current state $V(s_t)$, with the actual reward from the transition r_{t+1}added up [8]. The multiplying factor α is a part that determines the rate of learning.

Learning rate is a hyperparameter that needs to be tuned depending on the use case. A small learning rate is helpful for convergence to one value, but it also slows learning down as the value is not updated enough. It is essential to find the right balance between a small and a large learning rate for every problem.

Q-learning takes this approach and applies TD to the Q-function as can be seen in the equation 1.12.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad \textbf{(1.12)}$$

In this equation, the equivalent to the TD error would be the expression $r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ where the r_{t+1} is the reward of the transition, $max_a Q(s_{t+1}, a)$ is the best Q-value for the next pair of state and action and the last component $Q(s_t, a_t)$ is the Q-value for the current state and action. With more and more iterations, the values get more accurate as they are based on real observations.

The process of learning the best policy and discovering the Q-values for the pairs of states and actions is guided by a trade-off between exploration and exploitation. There are lots of exploration techniques such as Thompson Sampling [20], Curiosity based [7], etc. but the Epsilon Greedy technique [16] was used for this case as it is proved to work in diverse environments, including multi-agent settings [23, 25].

1.6.2. The Epsilon-greedy

Epsilon greedy [16] consists of modelling the trade-off with a parameter named epsilon that determines the probability of choosing random actions versus the actions dictated by the Q-value function.

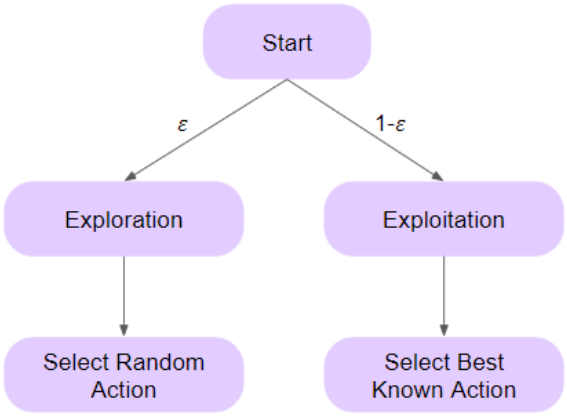

Fig. 1.2 Diagram representing the epsilon greedy strategy.

A random number between 0 and 1 is generated, and if it's less than the epsilon, a random action is chosen. On the contrary, if the random number exceeds the epsilon value, the best-known action is selected. The decision diagram of this process is represented in figure 1.2.

Usually, the epsilon value is updated and at first it is high as the agent needs to explore the options by trial and error (exploration). Further onto the training the policy needs to be adjusted, thus more actions are taken under the policy (exploitation). In this case, rather than defining one epsilon, a maximum and minimum probability of exploration is defined accompanied by an exploration decay parameter $\Delta\varepsilon$, to better control the trade-off between exploration and exploitation. At the end, a value of the epsilon is computed each learning step following the equation 1.13.

$$\varepsilon = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) * e^{-\Delta\varepsilon \cdot N} \quad \textbf{(1.13)}$$

In equation 1.13, the difference between ε_{max} and ε_{min} is multiplied by $e^{-\Delta\varepsilon \cdot N}$, with $\Delta\varepsilon$ being the exploration decay, and N being the current number of steps in the training. This is all added to the minimum epsilon ε_{min}. As the exponential is negative, the exploration starts with ε_{max} and ends with ε_{min}.

1.6.3. Pseudo Algorithm

The steps described above can be summarised in the pseudo algorithm from figure 1.3. The Q-function is initialized randomly and for each episode an action is chosen based on the epsilon greedy strategy. Once the action is performed and the new state and reward are obtained, the Q-function is updated. This process lasts until the next state is the desired state, or for a maximum number of steps T, like in figure 2.4.

Require: discount factor γ, learning parameter α
  initialize Q arbitrarily (e.g. $Q(s,a) = 0, \forall s \in S, \forall a \in A$)
  for each episode **do**
    s is initialized as the *starting* state
    repeat
      choose an action $a \in A(s)$ based on an exploration strategy
      perform action a
      observe the new state s' and received reward r
      $Q(s,a) := Q(s,a) + \alpha\left(r + \gamma \cdot \max_{a' \in A(s')} Q(s',a') - Q(s,a)\right)$
      $s := s'$
    until s' is a goal state

Fig. 1.3 Q-learning algorithm [24].

CHAPTER 2. DEEP REINFORCEMENT LEARNING

2.1. Use of Neural Networks in Reinforcement Learning

In complex systems, where the number of states is close to infinite, some elements of RL cannot be approximated and saved in tables. That's when neural networks come in and deep reinforcement learning (DRL) starts. DRL is the branch of RL that uses neural networks to approximate either the value function, the policy function, the state transition function or the reward function [14].

Neural networks are based on the human brain structure, with processing nodes and connections that vary in intensity. To decide if a neuron is activated or not, an activation function is needed. In simple words, it transforms the input in the node to an output signal depending on the importance that is given to that information. The node adds the inputs multiplied by the weights, and bias if there is any, and puts it in the activation function to obtain an output that will be sent to the next layer.

Neural networks are arranged in layers and weights (see w_{jm} and w_{mk} values in figure 2.1) that control the connections between the neurons. Normally the networks are feed-forward, which means that the connections only go from the input layer to the next layers. The weights are updated using backward propagation during training, when the error signal is sent from the output layer to the input layer, passing through the hidden layers, to adjust and strengthen the connections that worked and weaken the ones that did not. In figure 2.1 a diagram of this process is represented, where input signals and error signals are represented by purple arrows in opposite directions.

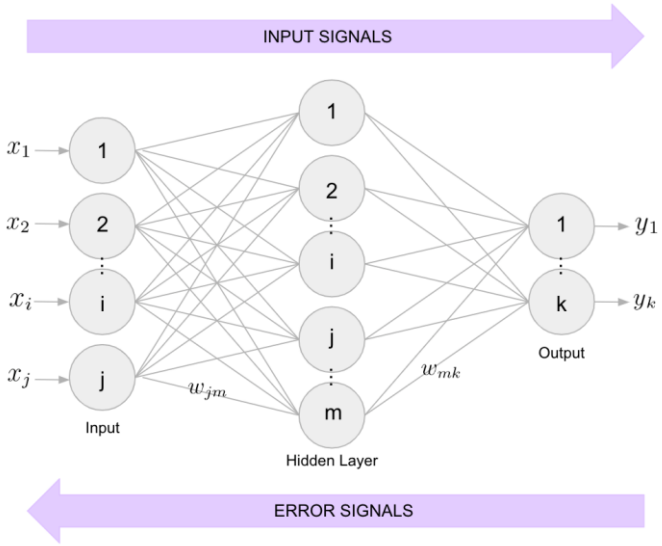

Fig. 2.1 Diagram of a feed-forward neural network with backpropagation.

The idea of using neural networks for RL dates to 1955 when Farley and Clark [5] adjusted the parameters of linear threshold functions representing policies using early neural networks. Since then, neural networks have been appearing now and then in literature in topics related to RL However, it was not until Werbos [24] when the concept as we know today appeared, although much simpler. Werbos used a backpropagation model to approximate the value of the policy and value function through TD methods showing promising results.

Nevertheless, what made DRL explode in popularity was a paper published by DeepMind in 2015 [16] that showed models that could beat humans in Atari games. More about this paper is described in the following chapter.

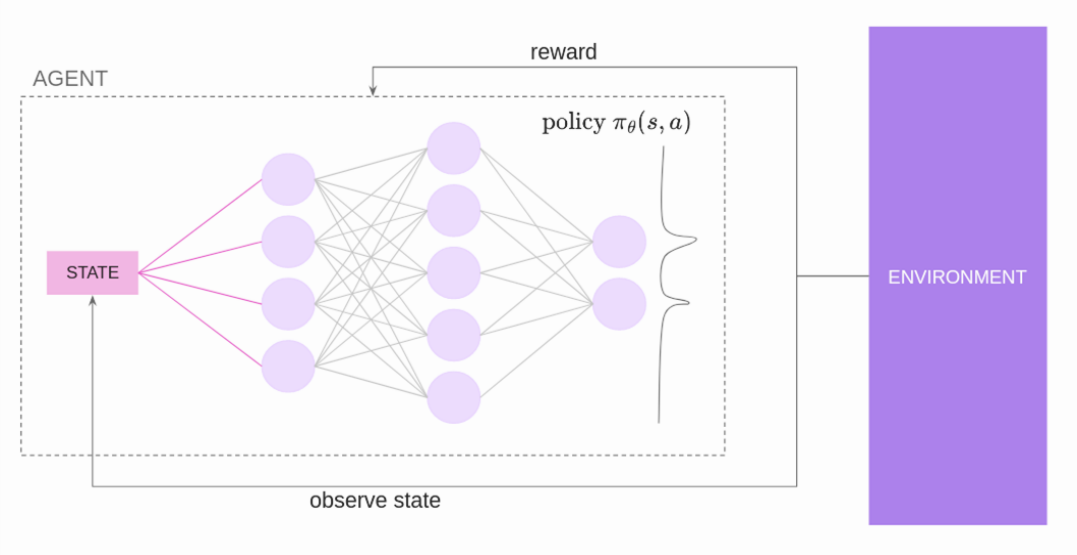

Fig. 2.2 Diagram of a simple deep reinforcement learning architecture.

A simple diagram of a DRL architecture is shown in figure 2.2. Here the agent inputs the states and gets an action through the neural network that follows the policy π. The parameter θ represents the weights of the neural network.

In short, DRL allows to apply RL to much more complex environments, but complexity is proportional to the computational power and time needed for the training process. By now, uncountable algorithms have been developed to suit many environments and a fair share of them is available in open-source platforms.

2.2. Deep Q-learning

As mentioned above, Mnih et al. or popularly referred to as the name of the company, Deepmind's paper [16] about DRL in Atari games was ground-breaking. In these games, the model learned its strategy through fully connected convolutional neural networks and the pixels of the screen as an

input. With a few hours of training, it could get scores comparable, if not better, than the best players of the game. This kind of performance and stability was never seen before and got the interest of many, which led to even more advances in the field.

An essential contribution of the DeepMind's paper was the storage of past experiences in what's called a Replay Buffer and updating the target network less times than the behavioural one.

The replay buffer is a storage space where experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ are saved in order to train the network with them. Instead of learning the current information in the timestep, this information gets saved in the replay buffer, and they are used for training in batches of random samples. What was highlighted in the paper is that random sampling helps to avoid biases that may occur in sequential environments, where the first few experiences may not be representative of the actual situation.

To further explain the second innovation, it should be noted that the target network and the policy network can lead to divergence when the same, as they are used both for computing the target value and the predicted value and these can differ. However, using two networks, one for the policy and one to update the target, solves this issue, as the target value stays locked for some timesteps. This locked interval is determined by a hyperparameter. The policy neural network weights are then copied or averaged to the target network.

Deep Q-Networks algorithm, popularly known by its acronym DQN, was first used by Mnih et al. [16]. It is a policy-free algorithm with a deep neural network used to estimate the Q-value function $Q(s, a, \theta)$. As a neural network is used, the Q function depends also on the weight's parameter θ, like in equation 2.1.

$$Q(s, a, \theta) \approx Q^{\pi}(s, a) \quad \textbf{(2.1)}$$

In Deep Q-learning the error used for backpropagation is the loss function, which is the sum of the squared differences of the Q-values and the target values, which is expresses as in equation 2.2.

$$L(\theta) = E[(r + \gamma max_a, Q(s', a', \theta) - Q(s, a, \theta))^2] \quad \textbf{(2.2)}$$

The Q-function is represented by adding the weights parameter θ. If the loss is minimised, it means that the network predictions and the target match. In order to minimise it, an optimization function is needed.

An optimization function's goal is to reduce the error in training. Lots of optimizers are available in open-source libraries, but the one that stands out the most in the community is Adam [12]. It combines different techniques of optimization into one method. It has been the most used one since its

appearance in 2014, and although it may have some problems in some use cases [19] there are improved versions of it that solve them [27].

The DQN algorithm is summarised in figure 2.3. It represents how the memory is filled with transition tuples stored in the replay buffer. The neural network learns by sampling this buffer randomly and getting batches that are used to update the policy network by using backpropagation of the loss to adjust the weights. Actions are selected through the epsilon greedy strategy, and they are executed in the environment, which starts the sequence again. Throughout the process, every determined period of steps, the weights from the target network are updated by using a soft or a hard approach.

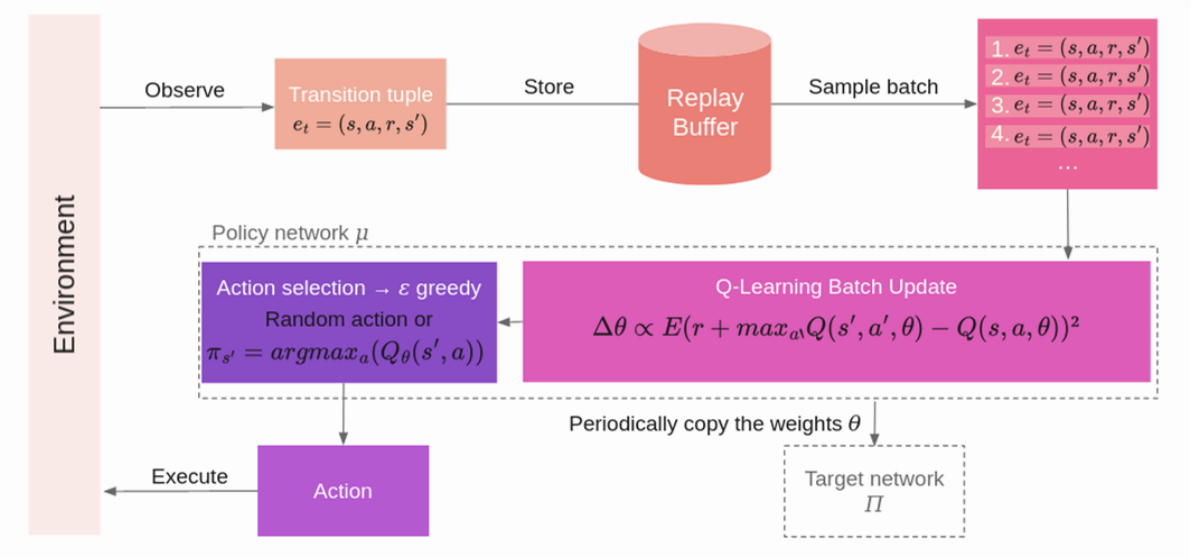

Fig. 2.3 DQN algorithm diagram.

The process is also summarised in the following pseudo algorithm from figure 2.3 used to code the learning function of this project.

Algorithm 1 Deep Q-learning with Experience Replay

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

Fig. 2.4 Deep Q-learning with Experience Replay algorithm [16].

CHAPTER 3. TOOLS, METHODS AND METRICS

In the last two chapters the RL basic concepts and the mathematical reasoning behind it has been introduced. Moreover, the neural networks paper in DRL has been explained, to finish off with the DQN algorithm, which is what is going to be used in this project.

The DQN algorithm will be used to train a centralized ATC that will learn to manage the actions of each plane with the goal of minimising conflict. The environment used is a simple 2D airspace simulator.

Since the Rules of the Air (RoA) are a set of rules all aircraft need to follow (see section 3.2.6) to avoid unpredictability in the airspace. To apply them to the RL training, transfer learning will be used.

Transfer learning consists of reusing knowledge from related tasks to update the current model, to train the neural network with already proven to work decisions.

Consequently, the RoA have been applied in the first part of the project by Lidia Fuentes Coll [6]. In this part of the work, a DRL policy has been developed and a comparison between the DRL policy on its own and the policy including transfer learning (DRLT) has been made.

3.1. Tools

3.1.1. OpenAI Gym

OpenAI Gym is a developer’s environment for RL and contains lots of RL environments, like the Atari games from DeepMind's paper. The airspace simulator used in this project was created in this environment [2].

3.1.2. Pytorch

Pytorch [18] is an open-source DRL framework in which the project is developed. Pytorch is a pythonic framework launched in 2016 by Facebook with the promise of a flexible, easy to use and fast tool. It kept its promise mainly by having two features, the first one being CUDA support, which takes advantage of the power of graphic processing units to train the RL models. The second one is by using dynamic computational graphs, that makes changing the order of operations possible at runtime.

Pytorch provides the functions necessary to define custom deep neural network models, its activation functions and optimizers among other functionalities. The

base of the framework is a class called Tensor, a multidimensional array where input and output data of the neural networks are stored. It is similar to a NumPy array but is optimised for GPU processing.

Although it is still a young competitor compared to other alternatives like Keras, its popularity arises among the research field. It was chosen for this project for its pythonic nature and thus simplicity in use. Even though it is simple to use, it has many useful functionalities implemented.

3.1.3. Weights and Biases

Weights and Biases (also known as WandB) is the platform used to visualise the progress of the training and save it. It is a cloud-based application that allows users to represent the desired parameters in graphs, group them in a dashboard, allows in-depth comparisons and report creation. Moreover, it has the capability to send a notification through Slack when the run is finished, it lets you save the console output and the last commit in GitHub that corresponds to the training run code.

The first considered platform for graphical representation of the training was Tensorboard due to its popularity, but WandB has wider features. It facilitates the use of different PCs with cloud storage of the information, allows sharing projects with a team, the comparison of runs is much easier and has many graph types to choose from.

3.2. Methods

3.2.1. The Environment

The environment for this project was provided by Eurocontrol as part of the Innovation Challenge 2022. As aforementioned, it was developed in OpenAI Gym, and it includes the initialization, and the step, reset, close and render functions as well as place to fill the observation, reward and resolution functions.

The environment is a multi-agent 2D environment with the number of flights as an input. The size and shape of the airspace sector is variable, and the minimum and maximum area are constants that can be defined. The shape of the sector changes each episode, which finishes when all the flights get to an exit point of the sector, or the number of steps reaches its maximum.

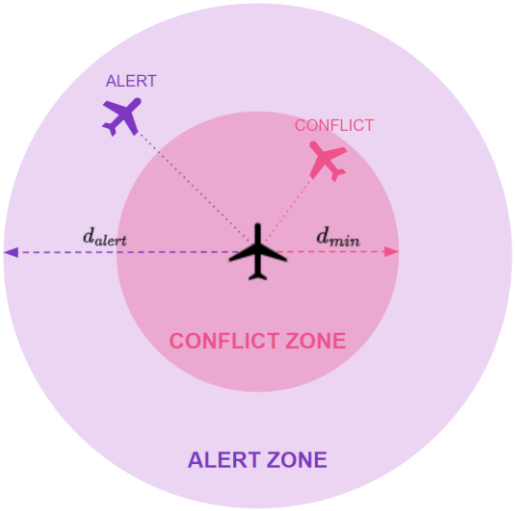

Fig. 3.1 Alert zone and conflict zone with its respective radiuses.

Two parameters defined for conflict detection and resolution are the minimum distance d_{min} and the alert distance d_{alert}. The minimum distance is the radius which determines the conflict area of a flight, this is, the area not to enter, and the alert distance is the same but for the alerts. Both areas are represented in figure 3.1. Note that $d_{min} < d_{alert}$.

3.2.1.1. Flights

The flights are the agents of the environment that are initialised with a random speed in a particular range and an arbitrary heading.

Each flight is defined with the following parameters:

- position: two-dimensional coordinates in Euclidean space (x_i, y_i)
- speed v_i
- track χ_i

And these parameters change every timestep following equation 3.1 for changes in x axis, 3.2 for changes in the y axis, 3.3 for the changes in speed and 3.4 for the change in track. In these equations Δt is the timestep, $\Delta\chi$ is the track change and Δv is the speed change.

$$x_i(t+1) = x_i(t) + v_i(t) sin\chi_i(t)\Delta t \quad \textbf{(3.1)}$$

$$y_i(t+1) = y_i(t) + v_i(t) cos\chi_i(t)\Delta t \quad \textbf{(3.2)}$$

$$v_i(t+1) = v_i(t) + \Delta v_i \quad \textbf{(3.3)}$$

$$\chi_i(t+1) = \chi_i(t) + \Delta\chi_i \quad \textbf{(3.4)}$$

Moreover, there are several elements defined for each episode: the entry point (x_E, y_E), the exit point, also called the target (x_T, y_T), the bearing ψ, the drift δ and the distance to the target d_{target}.

The rendering function plots the environment simulation and represents the parameters in the way symbolised in figure 3.2

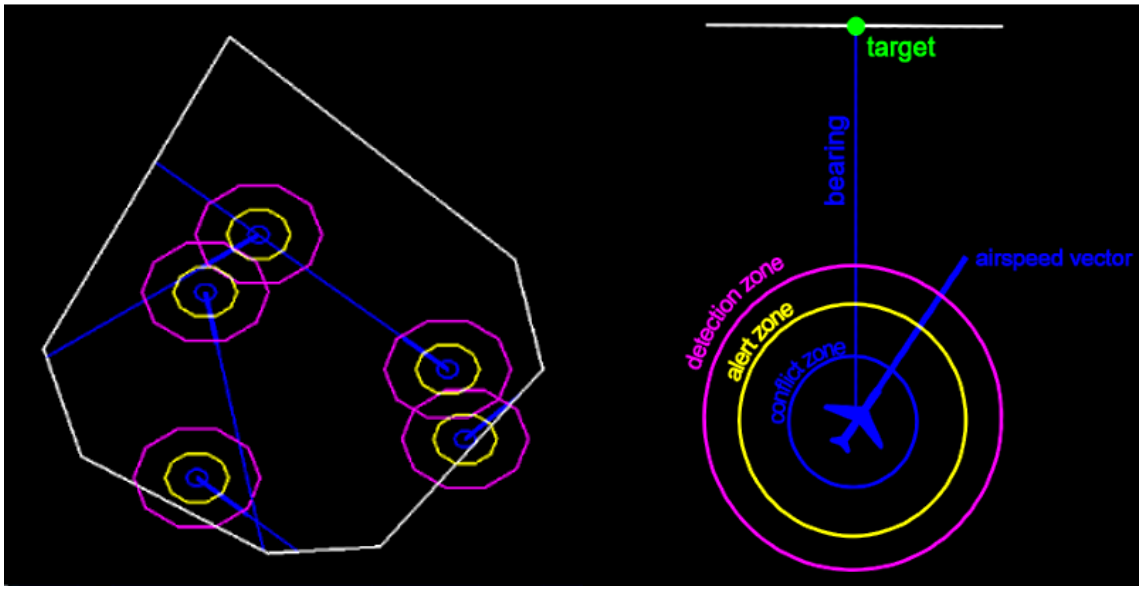

Fig. 3.2 Environment rendering with the airspace sector and the flights on the right and an individual sector with the components of the representation named.

It is worth mentioning that the position of the aircraft is contained in the airspace sector, and they start and finish the simulation in its border. When the aircraft exits the sector, it is not considered anymore in the episode.

The reinforcement learning model will be a cooperative environment that applies the same reward function for all the agents. Air traffic control is one of the most popular examples in cooperative multi-agent environments, since the flights need to cooperate in order to maintain separation between them. A DQN algorithm will be used, and the states, actions and reward will be shaped in a way that corresponds to the policy following the Rules of the Air to make the learning easier.

The code has been developed in Python and is available on GitHub in this link: https://github.com/vera15380/conflict-avoidance-TFG-WP.git. The code is accompanied by an explanation of its architecture in appendix B.

3.2.2. DQN

The DQN algorithm has been implemented on top of the OpenAI Gym environment to reduce the number of conflicts through reinforcement learning.

Although it is a multi-agent environment, the DQN algorithm is applied individually to each aircraft since the observations made it possible to use the same neural network for all the aircraft, because of using relative values. The actions are then joint into one tensor and passed to the environment.

3.2.2.1. Replay Buffer

The replay buffer is modelled as a double-ended queue with a maximum capacity as an input. The functions to append transitions to the buffer, to extract a sample batch of random indexes and to pop the last transition are written and available to use.

In order to append a transition, a tuple containing the state, the action, the reward and the next state needs to be created first.

3.2.2.2. Neural Network

A full dense two-layered feedforward network is used with the input size equal to the observation space shape and the output size equal to one as it only has to output one action. The hidden neurons are an input of the function and can be adjusted to find better performance.

```
class NeuralNetwork(nn.Module):
   def __init__(self, n_obs_individual, n_output, n_hidden):
       super().__init__()
       self.input = nn.Linear(n_obs_individual, n_hidden)
       self.hidden = nn.Linear(n_hidden, n_hidden)
      self.hidden_2 = nn.Linear(n_hidden, n_hidden)
       self.output = nn.Linear(n_hidden, n_output)

   def forward(self, x):
        x = self.input(x)
        x = self.hidden(x)
        x = self.hidden_2(x)
        x = self.output(x)
        x = F.relu(x)
        return x
```

3.2.2.3. Action Selection

The action selection function is based on the epsilon-greedy strategy, so the equation 1.13 is used to update the value of the exploration rate with the exploration range and decay. When a random number generated each step is more than the exploration rate, the policy action is chosen, but when it is less, a random action is chosen.

```
def select_action(self, obs):
   self.exploration_rate = self.exploration_min + (self.exploration_max - \
    self.exploration_min)     *     math.exp(-1.     *     self.n_steps     *
    \self.exploration_decay)
```

```
    if random.random() > self.exploration_rate:
        q_eval = self.policy_net.forward(torch.Tensor(obs))
        action = q_eval[0].max(0)[1].cpu().data.item()
    else:
        action = random.randint(0, self.action_size - 1)

    self.n_steps += 1
    return action
```

The returned action is joined to actions from all the flights in a list and finally given to the environment to perform a step in the main.py file.

3.2.2.4. Learning Function

The learning function is based on the pseudo algorithm provided in section 1.6.3. First a sample of the replay buffer is taken and divided into states, actions, rewards and next states. These arrays are converted to tensors, and the unsqueeze function is used to get the correct dimensions of the tensors, in order to be able to operate with them.

Then, the expected Q-values are extracted from the local network and the Q-values for the next states are taken from the target network.

The Q-targets are computed with Bellman's equation and then the loss is computed as the smooth L1 Loss [18]. The loss is minimised, and the backward propagation is made. The target network is updated every chosen number of steps `self.target_update` with a soft update.

```
# Getting sample from replay buffer
states, actions, rewards, next_states =
self.replay_buffer.sample(self.memory_sample_size)

# Convert the batches to torch tensors
rew_s = torch.FloatTensor(rewards).unsqueeze(-1)
obs_s = torch.FloatTensor(np.array(states))
actions_s = torch.LongTensor(actions).unsqueeze(-1)
next_obs_s = torch.FloatTensor(np.array(next_states))

# Get expected Q values from local model
q_policy = self.policy_net(obs_s).gather(1, actions_s)

# Get max predicted Q values (for next states) from target model
q_next = (self.target_net(next_obs_s).detach()).max(1)[0]

# Compute Q targets for current states
q_target=rew_s+self.gamma*q_next.view(self.memory_sample_size, 1)

# Compute the loss and minimise it.
loss = self.loss_func(q_policy, q_target)
self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
```

```
# Update target network
if self.n_steps % self.target_update == 0:
     self.soft_update()
return loss
```

3.2.3. States

In order to describe the situation of the flight, states in numeric form are needed, and preferably normalised for better performance of the neural network. Each flight will have its own states and the observations describing the status of other flights near it.

The state of the ownship contains the distance to the target d_E, the distance to the closest flight $d_{closest}$, both divided by the minimum distance d_{min}, the track χ_i, the bearing ψ_i, and the normalised airspeed $\overline{v_i}$, which is defined by the expression in equation 3.5 as in [2].

$$\overline{v_i} = \frac{v_{max} - v_i}{v_{max} - v_{min}} \tag{3.5}$$

The vector for these observations will result in equation number 3.6.

$$o_i = \begin{bmatrix} d_E/d_{min} \\ d_{closest}/d_{min} \\ \chi_i \\ \psi_i \\ \overline{v_i} \end{bmatrix}^T \tag{3.6}$$

The observations implemented regarding other aircraft are local and relative observations. The idea behind it was to reflect the conflict classification by the Rules of the Air, which are explained more in detail in section 3.2.6. The representation is done through dividing a certain range around the aircraft into 6 sectors, with 0º corresponding to the heading of the aircraft, and symmetrically from there:

- 0º-15º range corresponding to the head-on conflicts.
- 15º to 110º corresponding to the converging conflicts.
- 110º to 180º corresponding to the overtaking conflicts.

These sectors are represented in figure 3.3 where the flight is in the centre and the detection zone is painted depending on the type of the potential conflict sector.

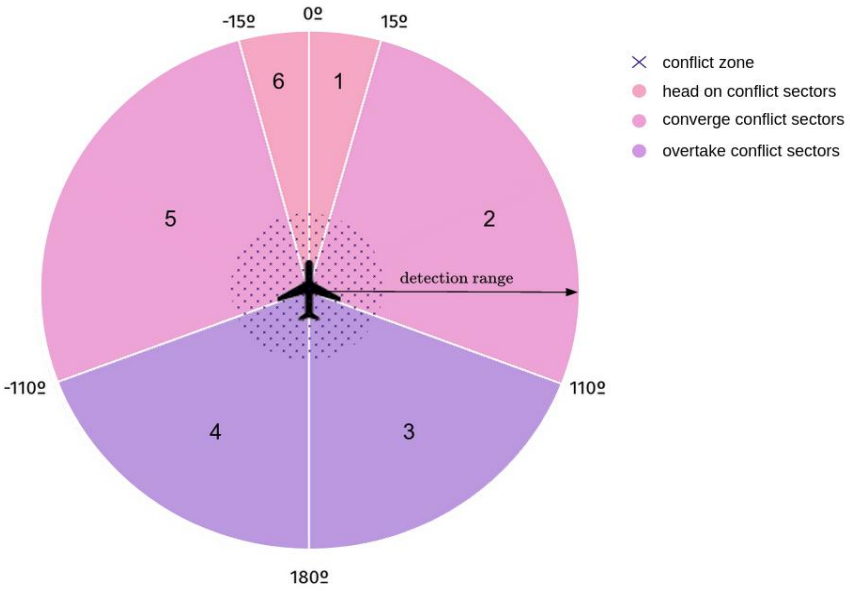

Fig. 3.3 Sector graphic representation.

Note that from now on the ownship will be the flight in the centre of the detection zone and intruders will be the ones that trespass this zone.

From each of these sectors, the closest intruder is chosen. The observations are a matrix where each row corresponds to one intruder per sector, starting from 0º and going clockwise. At the end, also some observations about the ownship are concatenated to this matrix. The observations for each row are formalised in equation 3.7.

$$s_{ij} = \begin{bmatrix} d_{CPA_{ij}}/d_{min} \\ t_{CPA_{ij}}/60 \\ \beta_{ij} \\ d_{ij}/d_{min} \\ v_{r_{ij}} \end{bmatrix}^T \quad \textbf{(3.7)}$$

Where d_{CPA} is the minimum separation that will occur between two aircraft and t_{CPA} is the time until the aircraft will be in that position. They are computed using the formulas found in [17]. β_{ij} is the relative angle between the heading of the ownship and the intruder, d_{ij} is the distance between both flights, d_{min} is the conflict range and $v_{r_{ij}}$ is defined as in equation 3.8.

$$v_{r_{ij}} = \frac{v_{max} - (v_i - v_j)}{v_{max} - v_{min}} \quad \textbf{(3.8)}$$

The observation matrix for all the sectors will be the one in equation 3.9.

$$s_i = \begin{bmatrix} d_{CPA_{ij}}/d_{min} & t_{CPA_{ij}}/60 & \beta_{ij} & d_{ij}/d_{min} & v_{r_{ij}} \\ d_{CPA_{ik}}/d_{min} & t_{CPA_{ik}}/60 & \beta_{ik} & d_{ik}/d_{min} & v_{r_{ik}} \\ d_{CPA_{il}}/d_{min} & t_{CPA_{il}}/60 & \beta_{il} & d_{il}/d_{min} & v_{r_{il}} \\ d_{CPA_{im}}/d_{min} & t_{CPA_{im}}/60 & \beta_{im} & d_{im}/d_{min} & v_{r_{im}} \\ d_{CPA_{in}}/d_{min} & t_{CPA_{in}}/60 & \beta_{in} & d_{in}/d_{min} & v_{r_{in}} \\ d_{CPA_{io}}/d_{min} & t_{CPA_{io}}/60 & \beta_{io} & d_{io}/d_{min} & v_{r_{io}} \end{bmatrix} \quad \textbf{(3.9)}$$

For better understanding let's follow with a situational example from figure 3.4.

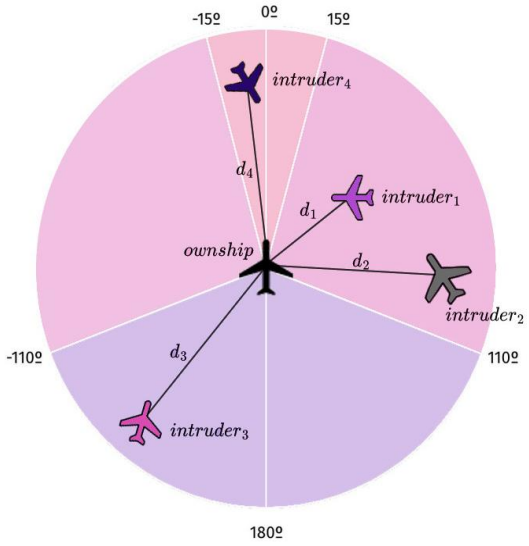

Fig. 3.4 Situational example of an ownship and 4 intruders penetrating different sectors from the observations.

The distribution of aircraft found in figure 3.4 contains an ownship and 4 intruders, intruder 1 and 2 in sector 2; intruder 3 in sector 4; and intruder 4 in sector 6. However, as $d_1 < d_2$, only intruder 1 will be considered in the observations for this particular ownship in that sector. The rows of the state matrix occupied by the intruders are highlighted in figure 3.5. The other rows will be filled with high numbers to give the sensation that the flight is far away.

$$s_i = \begin{bmatrix} d_{CPA_{ij}}/d_{min} & t_{CPA_{ij}}/60 & \beta_{ij} & d_{ij}/d_{min} & v_{r_{ij}} \\ d_{CPA_{ik}}/d_{min} & t_{CPA_{ik}}/60 & \beta_{ik} & d_{ik}/d_{min} & v_{r_{ik}} \\ d_{CPA_{il}}/d_{min} & t_{CPA_{il}}/60 & \beta_{il} & d_{il}/d_{min} & v_{r_{il}} \\ d_{CPA_{im}}/d_{min} & t_{CPA_{im}}/60 & \beta_{im} & d_{im}/d_{min} & v_{r_{im}} \\ d_{CPA_{in}}/d_{min} & t_{CPA_{in}}/60 & \beta_{in} & d_{in}/d_{min} & v_{r_{in}} \\ d_{CPA_{io}}/d_{min} & t_{CPA_{io}}/60 & \beta_{io} & d_{io}/d_{min} & v_{r_{io}} \end{bmatrix} \begin{matrix} \\ \\ \rightarrow intruder_1 \\ \\ \rightarrow intruder_3 \\ \\ \rightarrow intruder_4 \end{matrix}$$

Fig. 3.5 The observation matrix with the intruders' rows highlighted in the corresponding colour.

This matrix is joined with the ownship states. The overall matrix will then be flattened, converted to a tensor and passed to the neural network.

3.2.4. Reward

The reward function has been a real challenge as it could not deviate from the main goal of the project, but it must influence the behaviour of the aircraft so that they end up following a similar route to the one that was planned initially and get to the target. After many versions and attempts, a simple reward function was what worked best for this case.

The reward function considers the current track of the aircraft and if it has any conflicts or alerts in a timestep.

The overall function is formalised below, in equation 3.10.

$$R_i = k_{track_i} - k_{alert_i} \cdot g_{alert_i} - k_{conflict_i} \cdot g_{conflict_i} \quad \textbf{(3.10)}$$

$$\text{where } k_{track} = 0.1 \text{ if } \chi = \psi \text{ else } 0$$

$$k_{alert} = 1$$

$$g_{alert} = \frac{d_{alert} - d_{closest}}{d_{alert}} \text{ if } d_{closest} \leq d_{alert} \text{ else } 0$$

$$k_{conflict} = 10$$

$$g_{conflict} = \frac{d_{min} - d_{closest}}{d_{min}} \text{ if } d_{closest} \leq d_{min} \text{ else } 0$$

If the track coincides with the bearing (so the aircraft is going to the target), there's a bonus of $k_{track} = 0.1$. If the aircraft has a conflict or is on alert, there is a penalty. The penalty is divided into two factors: k and g. The k is the constant that determines what importance to give the alerts and the conflicts. in

this case the alerts have an importance of 1 and the conflicts of 10. The g is the gravity factor, that computes the penetration distance of each zone (alert or conflict) and divided by the radius of the respective zone (d_{alert} or d_{min}).

3.2.5. Actions

The action space consists of N discrete actions, which are:

- Stay with the same configuration.
- Point to target, so the track equals the bearing $\chi_i = \psi_i$.
- Change track with angle α, α defined by the expression in equation 3.11.

$$\alpha = \{x | x \epsilon \pm K \cdot \Delta\alpha\} \, where \, K = \{y \epsilon \mathbb{N} | y \leq N - 2\} \qquad \textbf{(3.11)}$$

- Where N is the number of discrete actions and $\Delta\alpha$ is the angle change interval allowed. If $\Delta\alpha$ is equal to 15º for example, then the turns would be of $[\pm 15º, \pm 30º, \ldots, 15º \cdot K]$
- Change airspeed $\pm v$ knots.

The number of its action and its resolution is presented in table 3.1.

Table 3.1. Equivalence of the action number in the code and the resolution.

Action number	Resolution
0	No change in speed or track.
1	Change track to bearing
2	Turn +15º
3	Turn +30º
4	Turn +45º
5	Turn +15º
6	Turn +30º
7	Turn +45º
8	Change speed +5 kt
9	Change speed -5kt

3.2.6. Rules of the Air Policy

A policy was developed as part of another final degree project [6] with the use of solely conditionals and not RL. This policy follows the Rules of the Air, which classifies the conflicts in 3 categories and has different resolutions that must be followed in each case. Moreover, this policy also considers the right-of-way rule,

that gives preference to one of the aircraft so only one aircraft must manoeuvre in case of a potential conflict situation.

The 3 categories and the actions that should be taken in each case are about to be specified, according to the general operating flight rules [4]. The classification depends on the relative angle from the ownship in which the conflict takes place, which can be seen in figure 3.3.

- Head-on. Both aircraft should modify the heading to the right.
- Converging. the aircraft on its right has the right-of-way.
- Overtaking. The aircraft that is being overtaken has priority and the one overtaking should deviate its heading to the right.

This policy uses a prediction of the area where the loss of well-clear will take place, computes the turn angle necessary for a safe avoidance manoeuvre both to the left and right and chooses the largest angle of the two in order to keep away from the conflict. The original policy had a continuous angle output, but this project adapted it to a discrete version with limited performance.

In the beginning, the policy had no turn limitation, with the aircraft having the possibility to turn maximum 180º and with a continuous angle. In order to adapt it to a discrete algorithm, the actions have been limited to turns of $\pm 15º, \pm 30º$ and $\pm 45º$. The conflict resolution capacity of the policy has not been affected much by it, as the aircraft that turned more than 45º before, now do it too, only in more steps. However, precision has been lost as the actions are now discrete and the angles must be rounded up. Moreover, the new action options also include airspeed changes, which is not considered in the RoA policy.

3.3. Metrics

Like in any project, it is necessary to establish the metrics in which the success of the DQN training will be measured. In this way, a glimpse of the status of the simulation and if the objectives established at the introduction are reached is going to be based on facts.

The utmost important objective to measure the success of is the conflict resolution, so metrics for it will be the most relevant.

Conflicts and other metrics will be displayed in plots. To see the evolution the graphs will have in the x-axis the number of steps and the y-axis is the metric mentioned in each section. A graph setup in WandB for each graph is similar and can be seen in figure 3.6, and most of the line plots are smoothened by applying an exponential moving average in order to get a better understanding of the evolution of the training even though some results may oscillate quite a bit at first. The original line appears as a faded-out colour and the smoothed-out version is a more vivid colour.

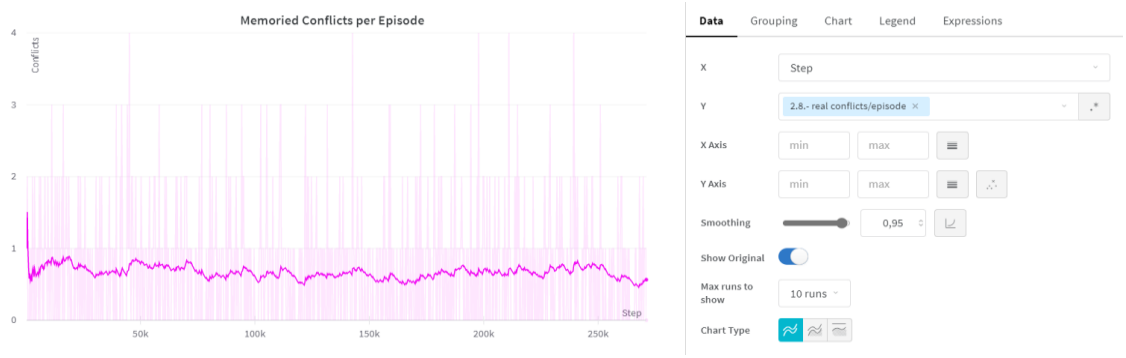

Fig. 3.6 Graph setup interface in WandB with a sample graph.

Table 3.2 presents the metrics that will be commented in more detail in sections 3.3.1 and 3.3.2 with its corresponding name in the code.

Table 3.2. Metrics and their name in the code.

Metric	***Name in the code***
Success rate (evaluation)	`Success_rate_eval`
Reward average	`avg_rew_ep`
Aircraft Conflicts	`n_real_conflicts_episode`
Simultaneous conflicts	`n_conflicts_ep`
Minimum separation distance	`critical_distance`
Number of turns	`n_turns_ep`
Distance left to target	`distance_left_to_target`
Extra distance flown	`extra_distance`

3.3.1. Metrics Used for the Evaluation of the Training

3.3.1.1 Success rate of the evaluation

The success rate of the evaluation will be the percentage of episodes that are successful (with no conflicts) after training the DQN network and giving no exploration options to the neural network.

3.3.1.2 Averages

Since in the evaluation the exploration process is gone, the metrics to measure its performance will be the averages of the metrics mentioned above related to the training.

Moreover, these parameters will be compared between runs to tune the hyperparameters (see section 4.2) and obtain the simulation with the best results. The graph to compare these results will be a parallel representation of the most relevant parameters. For example, in figure 3.7 we can see a

comparison between the results of training with different learning rates, just two for the purpose of the example.

The first vertical axis of the graph represents the hyperparameter that is currently being compared, in this case the learning rate. The following four vertical axes are the success rate of evaluation (%), the average reward per episode, the average number of turns per episode and the average aircraft conflicts per episode, correspondingly. The graph represents the relationship between the values and allows a quick evaluation of the quality of the simulation while varying a chosen hyperparameter.

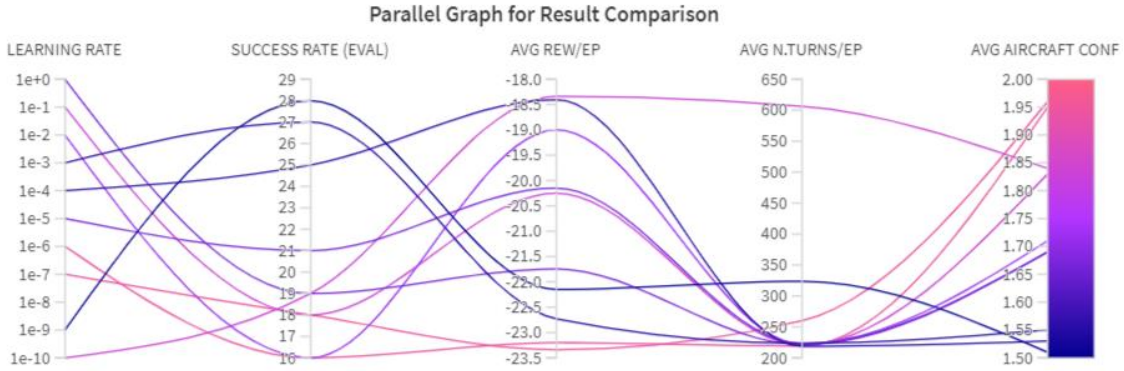

Fig. 3.7 Parallel graph example from WandB.

The results of the tuning will be detailed in section 4.2.

3.3.2. Metrics Used to Understand the Evolution of the Training

3.3.2.1 Reward per episode

The reward is one of the most important metrics in RL since it determines what are the good and bad decisions that an agent can take.

To compute the reward per step, the equation 3.10 is used. Its average is the sum of all the rewards for the flights that are not done in the episode and dividing the sum by the number of these flights.

The reward for one episode is the sum of the average reward obtained by the agents each step.

The reward graph is essential as it tells if the agents are learning or not. Ideally, at first the reward has low values and rises until reaching a stable value.

```
rew_without_nan = [x for x in rew if np.isnan(x) == False]

    if len(rew_without_nan) != 0:
```

```
        rew_average = np.average(rew_without_nan)
…
rew_episode += rew_average
```

Looking at the code, where `rew` is the array of rewards for each flight, first the empty reward from the flights that are already done are deleted, and then the average of the array is computed. At the end of each step, the reward average is added to the episode reward.

3.3.2.2 Aircraft conflicts for episode

The aircraft conflicts for one episode will be the number of the pairs of flights that violated each other's conflict areas, without considering that this conflict may last more than one step, so if the conflict is not resolved in the next timestep, it does not add it to the number of aircraft conflicts, it just adds conflicts that are new and were not considered before.

3.3.2.3 Simultaneous conflicts each episode

This parameter measures the pairs of flights that are in conflict in each step and episode of the simulation.

However, in contrast to the aircraft conflicts, this metric resets its memory each timestep, so a conflict that appeared in one step will be counted again if it is not solved in the next step. The conflicts each episode are the sum of the number of conflicts per step.

To further explain the difference between the first and the second metric, the code snippet below is used as a guide.

```
for i in range(self.num_flights - 1):
  if i not in self.done:
     for j in range(i + 1, self.num_flights):
        if j not in self.done:
   distance=self.flights[i].position.distance(self.flights[j].position)
   distance_NM = distance * u.m
              if distance < self.min_distance:
                 self.conflicts.update((i, j))
                 self.n_conflicts_step += 1
                 self.n_conflicts_episode += 1
                 self.matrix_real_conflicts_episode[i, j] = True
```

Several conflict metrics are defined: `n_conflicts_step`, `n_conflicts_episode` and `matrix_real_conflicts_episode`. The first two correspond to the metric described in this subsection, adding up 1 when a conflict appears, without caring what flights are in conflict and if it was there

before. The loop for detecting the conflicts does not run along the pairs of flights twice, so conflicts between aircraft i and j are not being repeated.

In contrast to this method of counting conflicts, `matrix_real_conflicts_episode`, that corresponds to the first metric, is a boolean quadratic matrix with a size equal to the number of flights. At first the whole matrix is False, and if a conflict is detected between flight i and flight j, the cell `matrix_real_conflicts_episode[i,j]` turns True. So, during an episode only one conflict between a pair of flights can be counted, even if it repeats itself over many steps. At the end of the episode the True values are counted, and it gives the number of aircraft conflicts during that period.

3.3.2.4 Minimum separation distance for episode

The minimum distance for an episode will be the lowest distance between two aircraft detected during an episode and it will be measured in nautical miles (NM).

3.3.2.5 Number of turns taken per episode

The number of turns appears as an attribute of the class flight. It is updated every step, so if a flight has turns it turns the parameter to 1. Then the attributes are summed up to obtain the number of turns each step, and at the end of the episode, the number of turns each episode.

It is an important factor as if it's too high it gives us the alert that the aircraft may be spinning around and not advancing to the target.

3.3.2.6 Distance left to target per flight for episode ending

After each episode, if there are any aircraft left that are yet to arrive at the target, the distance left for them to finish is added and saved. This informs if the aircraft are finishing the episodes and advancing to the target, and it is useful to detect spinning aircraft in the same place during all the steps of the episode.

3.3.2.7 Extra distance per flight for episode ending

This parameter tells if the deviation of the original route was significant or not. It is the difference between the distance from the entry point to the exit point in a straight line and the distance made by the flight.

CHAPTER 4. RESULTS.

4.1. Simulation Setup

The simulation parameters chosen for the simulations are the ones presented in table 4.1. The number of flights is 10 as it represents a high workload for an air traffic controller (ATCO). Several values of the other parameters have been tried and the ones that optimised the results appear in the table and will be used during all the training.

Table 4.1. Simulation parameters.

Number of flights	10
Area range	[15625, 40000] NM²
Speed range	[400, 500] kt
Maximum episode length	350 steps
Minimum distance	5 NM
Alert distance	15 NM
Detection radius	30 NM
Angle change $\Delta\alpha$	15º
Speed change Δv	5 kt
Discrete actions number	10
Exploration range [$\varepsilon_{min}, \varepsilon_{max}$]	[0.1, 1]
Target update	10

4.2. Hyperparameter Tuning

With the parameters from table 4.1 steady, in addition to the ones that can be seen in table 4.2. the hyperparameters of the DQN have been tuned. In order to do this, the hyperparameter tuned in that moment was the only variable parameter. The other ones stayed constant during the different runs of the code. The value with the best results from each hyperparameter is selected and they are put together in one simulation to validate if they work altogether.

Table 4.2. Parameters for hyperparameter tuning simulations.

Episodes	1000
Maximum replay buffer capacity	100

Batch size	64
Gamma γ	0.75
Tau τ	0.1
Learning rate α	0.001
Exploration decay $\Delta\varepsilon$	0.00001
Hidden neurons	128

Hyperparameter tuning can be tricky as oftentimes parameters depend on each other and work better in certain combinations. There is software that facilitates parameter tuning, however, open-source alternatives have not been found for this project, so the decision of doing it by this type of comparison has been made. While evaluating the results different aspects apart from the average reward must be considered.

Tables with average results from the evaluation of each parameter appear in appendix C.

4.2.1. Gamma

Although gamma is not considered a parameter in some research works, but more as a parameter that depends on the purpose of the project, its impact on the training has been checked. As mentioned before, it sets the importance of the long-term reward rather than the instant reward, with 0 being only caring about the immediate reward and 1 only caring about the long-term outcome.

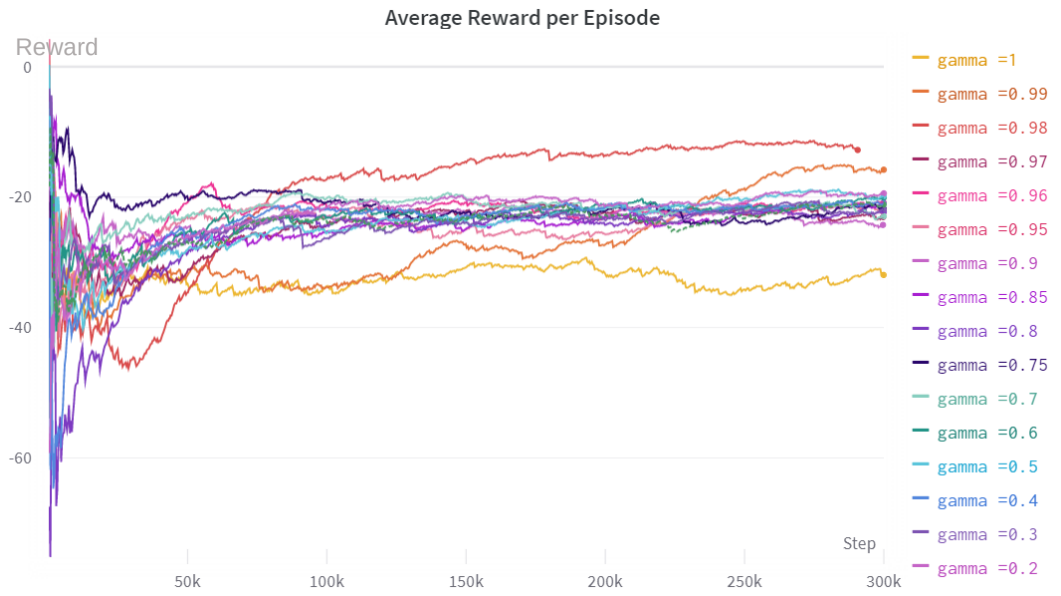

Fig. 4.1 Reward per episode graph comparison depending on gamma.

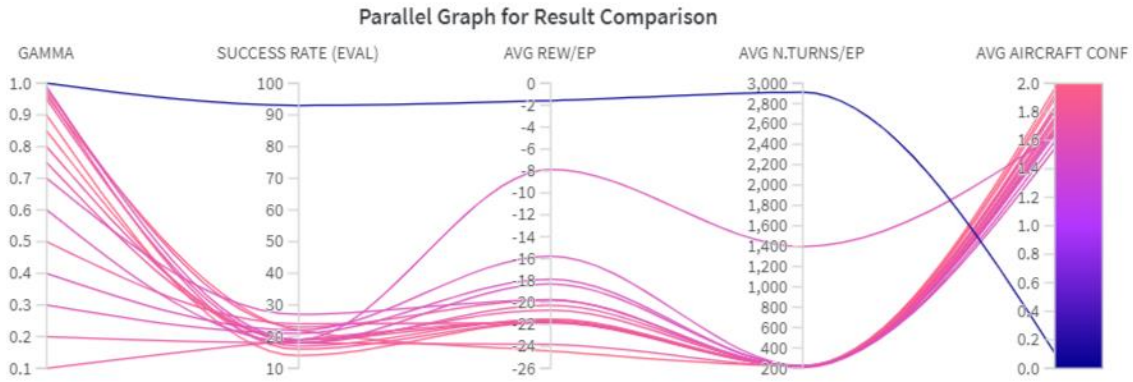

Fig. 4.2 Parallel graph of gamma, success rate, average reward, average number of turns and average real conflicts per episode.

From figure 4.2, it can be seen that the success rate was the highest for γ=1.00, but looking at the other columns, the number of turns taken is an abnormally high value, which indicates that the aircraft were just spinning in place. In figure 4.1 the two gamma values that outperform the rest in terms of reward are γ=0.98 and γ=0.99. However, looking at the next graph γ=0.98 stands out the most if γ=1.00 is discarded: with a higher success rate, less conflicts and a better reward.

4.2.2. Tau

Tau is the soft update coefficient that indicates the intensity of the update of the target network. A τ=0 would mean that no update is done, τ=0.5 would mean an average between the values of the policy and target nets, and τ=1 equals to a hard update, so copying the weight averages from the policy to the target neural network.

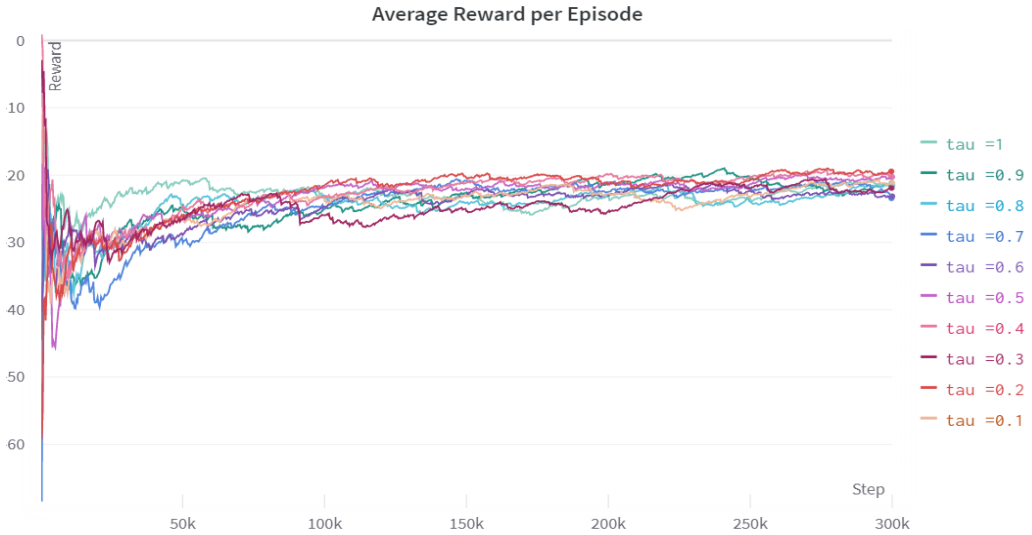

Fig. 4.3 Reward per episode graph comparison depending on tau.

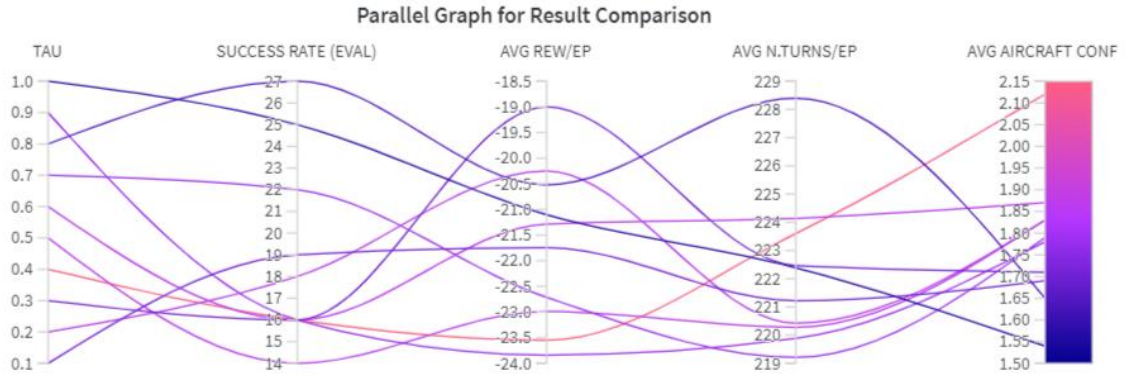

Fig. 4.4 Parallel graph of tau, success rate, average reward, average number of turns and average real conflicts per episode.

Without so much difference as gamma, tau values τ=1.0 and τ=0.8 are the most promising referring to figure 4.3. Looking at figure 4.4, the decision to make a hard update is taken, as it leaves fewer real conflicts and a better learning curve than τ=0.8, while having nearly the same success rate and average reward.

4.2.3 Hidden neurons

The hidden neurons are the number of nodes that each hidden layer of the neural network will have. Contrary to the input and the output layers, this one can be adjusted. If the neurons are overestimated, the network will act from memory and not by generalising. On the other hand, if the number chosen is too small. it will not be able to learn the patterns of the resolution as it will be too complex [10].

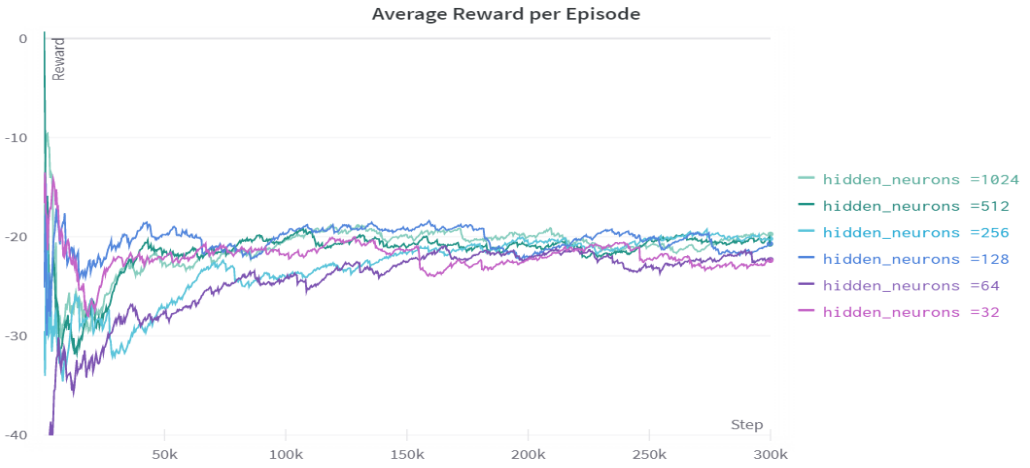

Fig. 4.5 Reward per episode graph comparison depending on the hidden neurons.

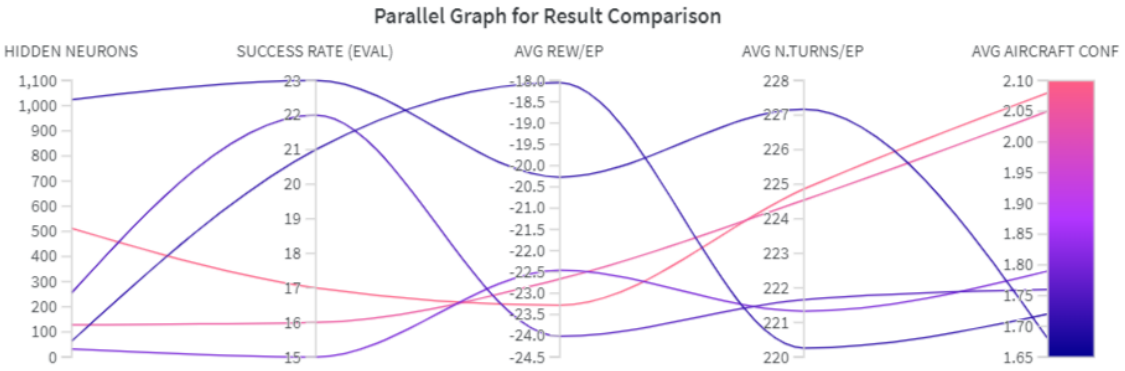

Fig. 4.6 Parallel graph of the hidden neurons, success rate, average reward, average number of turns and average real conflicts per episode.

With the results in hand, considering the graph of the reward during training in figure 4.5 and the averages of the evaluation from figure 4.6, 64 hidden neurons have been selected as the number of nodes of the hidden layers of the neural network. The reward was the highest in evaluation, the curve of the reward during training was correct and the number of conflicts diminished better than in other simulations.

4.2.4 Learning Rate

The learning rate defines how quickly the model stops learning and is considered adapted to the problem: if it takes big or slow steps to optimise the loss function. If the learning rate is too small, it will take lots of time to find the optimal solution. If it's too big, it can lead to sudden changes and disruption in the learning process, and the neural network to diverge.

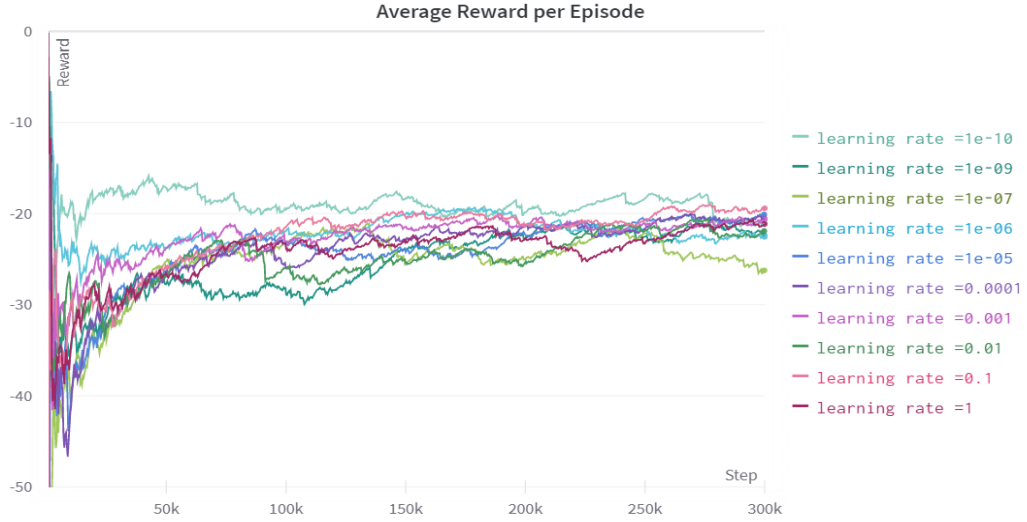

Fig. 4.7 Reward per episode graph comparison depending on the learning rate.

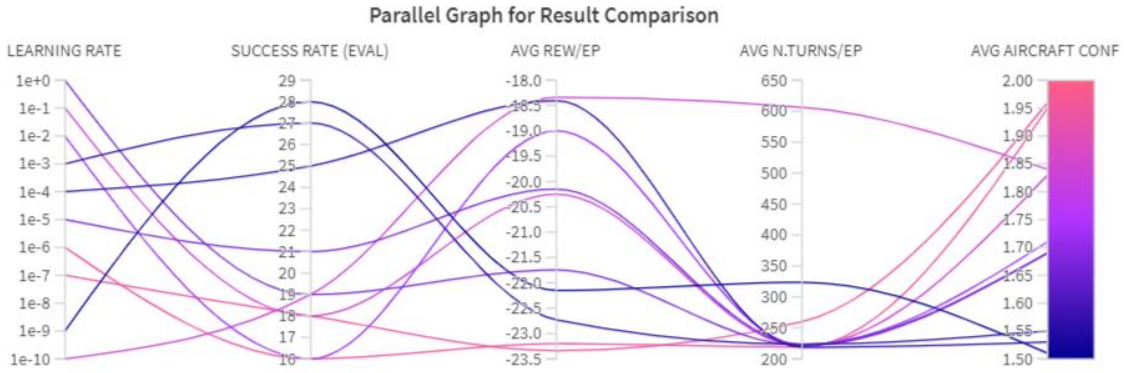

Fig. 4.8 Parallel graph of the learning rate, success rate, average reward, average number of turns and average real conflicts per episode.

The value of the learning rate with the best results without being too small is α=0.0001. Other good results are α=0.1 and α=1e-10 according to figure 4.7Fig. 4.7, as they stabilise at a higher reward value. The second value is discarded as the learning would be too slow, and between α=0.1 and α=0.0001, the latter gets better results in evaluation according to figure 4.8.

4.2.5 Batch Size

The batch size is the number of samples taken from the replay buffer and trained to the neural network at once. Smaller batch sizes could lead to instabilities in the model, but the larger ones tend to generalise poorly [11].

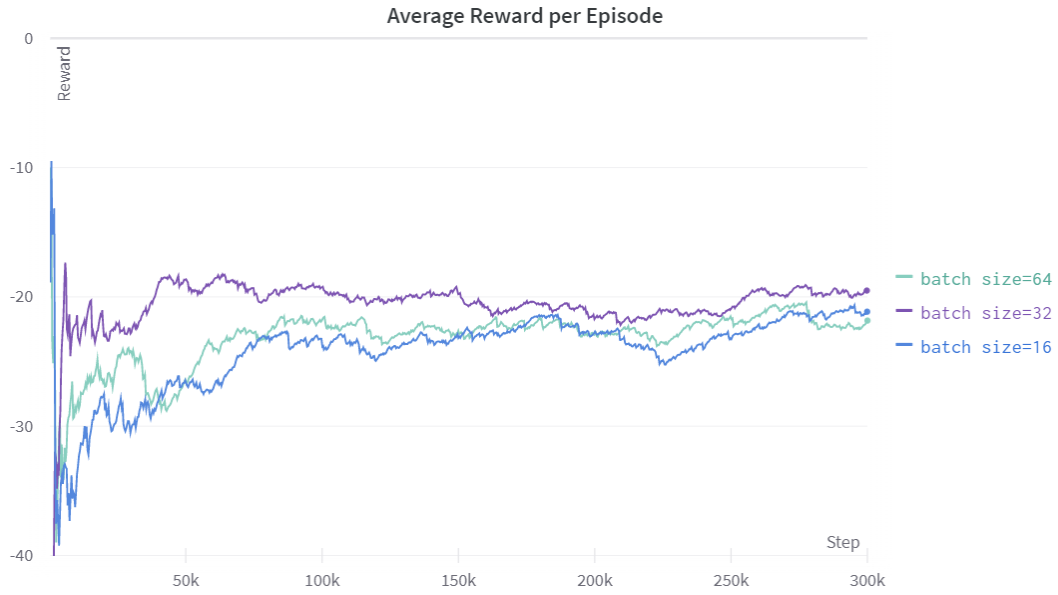

Fig. 4.9 Reward per episode graph comparison depending on the batch size.

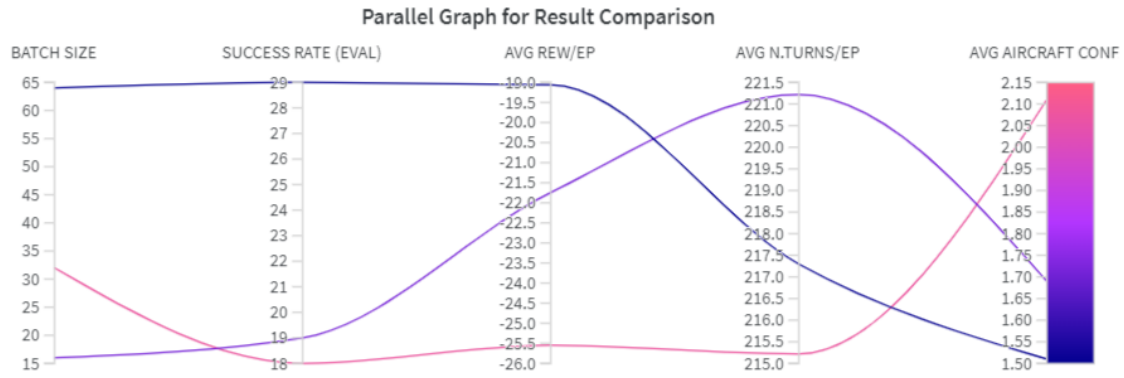

Fig. 4.10 Parallel graph of the batch size, success rate, average reward, average number of turns and average real conflicts per episode.

The batch size maximum number that could be tested is 64 as the memory size and the maximum episodes of the simulation limit it. From these runs, a batch size of 64 was the one with the best results overall. From figure 4.9 it can be told that it is the one with the highest reward throughout training, and from figure 4.10 the results

After the parameter tuning the results have been wrapped up in table 4.3, which collects the parameters that wil be used in the simulations described next.

Table 4.3. Parameters after hyperparameter tuning.

Episodes	50000
Maximum replay buffer capacity	10000
Batch size	64
Gamma γ	0.98
Tau τ	1
Learning rate α	0.0001
Exploration range [$\varepsilon_{min}, \varepsilon_{max}$]	[0.1, 1]
Exploration decay $\Delta\varepsilon$	0.0001
Hidden neurons	64
Target update	10

4.3. Deep Reinforcement Learning Results

The DRL approach is the one only using a DQN algorithm. The results of the training and the test are going to be presented below.

In figure 4.11 the reward evolution throughout the training can be seen, that stabilises at -10. The simultaneous conflicts graph seen in figure 4.12 shows a decrease in conflicts oscillating between 10 and 20 conflicts. The minimum

separation seen in figure 4.13 distance has variable values, but the average does not meet the requirement of conflict avoidance of a minimum separation distance of 5 NM.

Fig. 4.11 Average reward per episode. DRL.

Fig. 4.12 Simultaneous conflicts per episode. DRL.

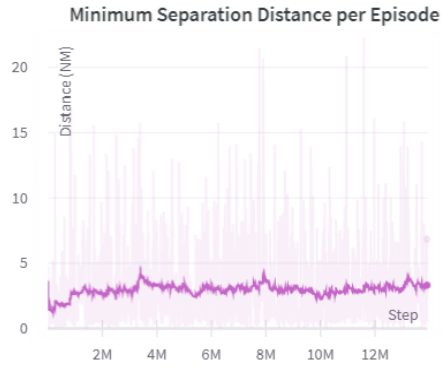

Fig. 4.13 Min. separation distance per episode. DRL.

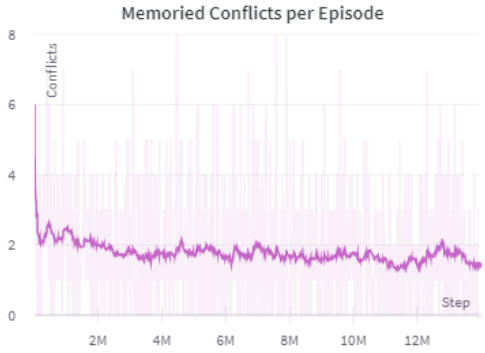

Fig. 4.14 Aircraft conflicts per episode. DRL.

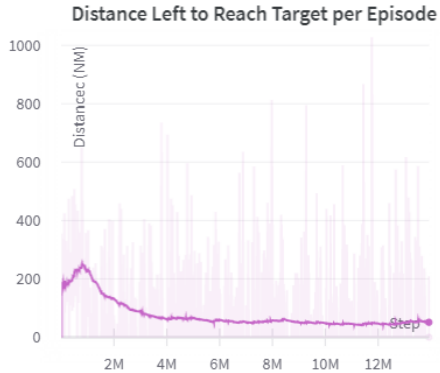

Fig. 4.15 Average distance left to reach target per episode. DRL.

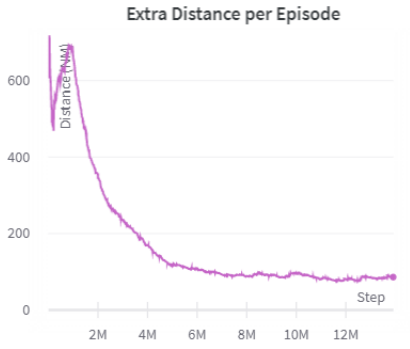

Fig. 4.16 Extra distance per episode. DRL.

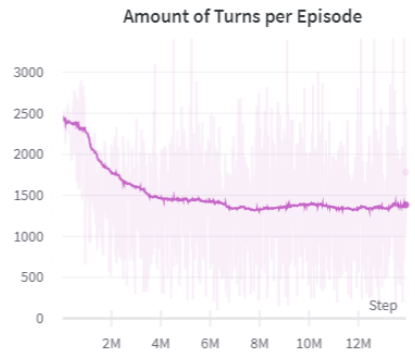

Fig. 4.17 Number of turns per episode. DRL.

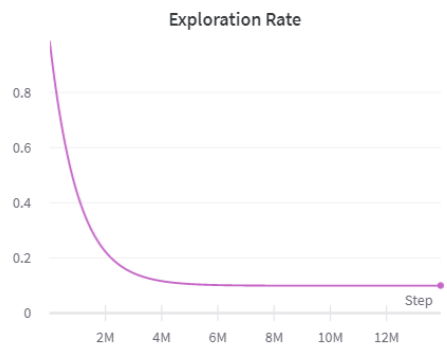

Fig. 4.18 Exploration rate. DRL.

The aircraft conflicts in figure 4.14 do not decrease notably, but the resulting average is of 1.78. As for the distance left to the target by the flights that did not finish the episode (see figure 4.15), it reaches a peak that decreases once the learning rate stabilizes (see figure 4.18). The same happens with the extra distance seen in figure 4.16, it follows the shape of the exploration rate graph from figure 4.18 except for the peak from the start.

The turns per episode diminish during training but are variable depending on the situation. However, the value is not too high to discard this evaluation from possible aircraft turning too much and spinning around, an issue that has been found in some simulations.

4.4. Reinforcement Learning with Transfer Learning Results

In all the graphs below related to Deep Reinforcement Learning with Transfer Learning (from now on referred to as DRLT policy) there is a clear separation from where the policy stopped applying the RoA to when only DQN decisions were applied. In figure 4.19 the reward had positive values of 10 until reaching the point of change that diminishes the reward to -20.

Fig. 4.19 Average reward per episode. DRLT.

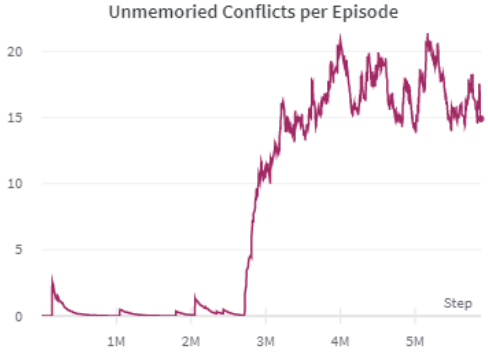

Fig. 4.20 Simultaneous conflicts per episode. DRLT.

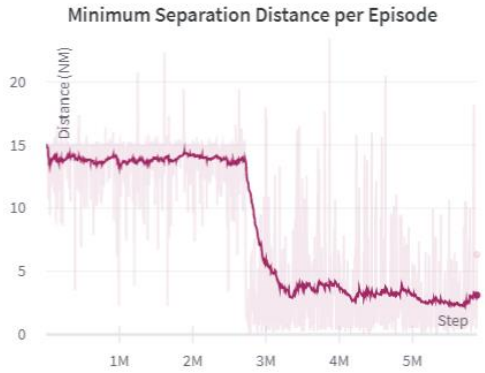

Fig. 4.21 Min. separation distance per episode. DRLT.

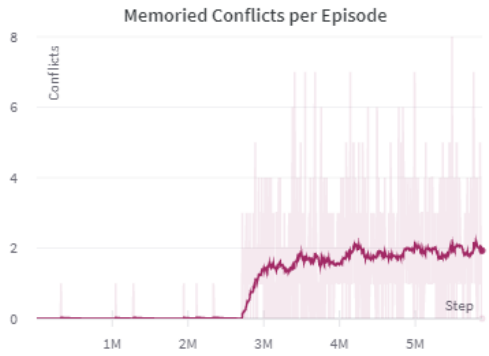

Fig. 4.22 Aircraft conflicts per episode. DRLT.

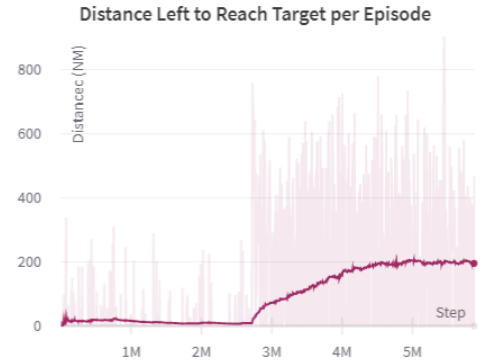

Fig. 4.23 Average distance left to reach target per episode. DRLT.

Fig. 4.24 Extra distance per episode. DRLT.

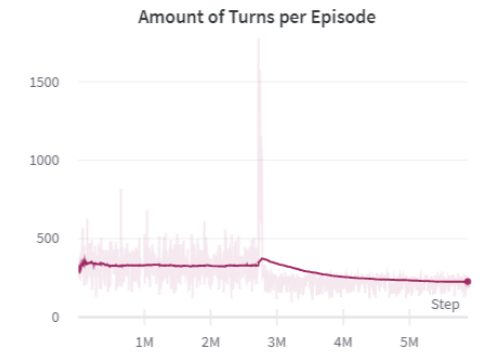

Fig. 4.25 Number of turns per episode. DRLT.

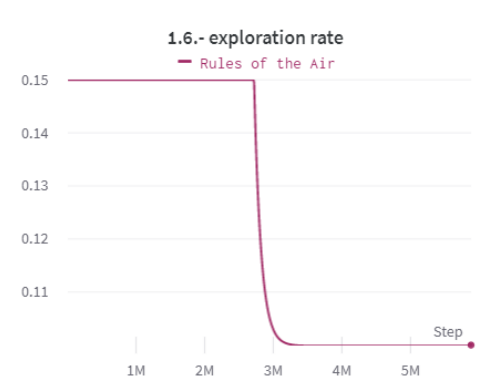

Fig. 4.26 Exploration rate. DRLT.

The simultaneous conflicts graph seen in figure 4.20 shows nearly no conflict until DQN is applied, which makes the conflicts rapidly increase conflicts. The minimum separation seen in figure 4.21 exceeds the average of 5 NM for conflict avoidance but it is quickly lost.

The aircraft conflicts in figure 4.22 do not inherit the values of the RoA policy. As for the distance left to the target by the flights that did not finish the episode (see figure 4.23), the extra distance from figure 4.24 and the number of turns in

figure 4.25, they are increased abruptly when the DQN point is reached. The exploration rate is only used once the TL starts and decreases fast to avoid the loss of learning with random choices (see figure 4.26) but it is not achieved.

To sum up, the transfer learning of the RoA did not improve the results of the DRL solution.

4.3 Comparison of DRL, DRLT, RoA and the baseline simulations

The two approaches studied, DRL and DRLT, are going to be compared to the RoA policy developed in [6] and the baseline environment which has no policy applied.

In figure 4.27 the average reward per episode for the 4 policies is represented, where it can be clearly seen that the RoA has the best results. The DRL is the second-best reward at the end of the simulations, and DRLT had good results at first that decay with the DQN application and ends up at the same level as the baseline environment.

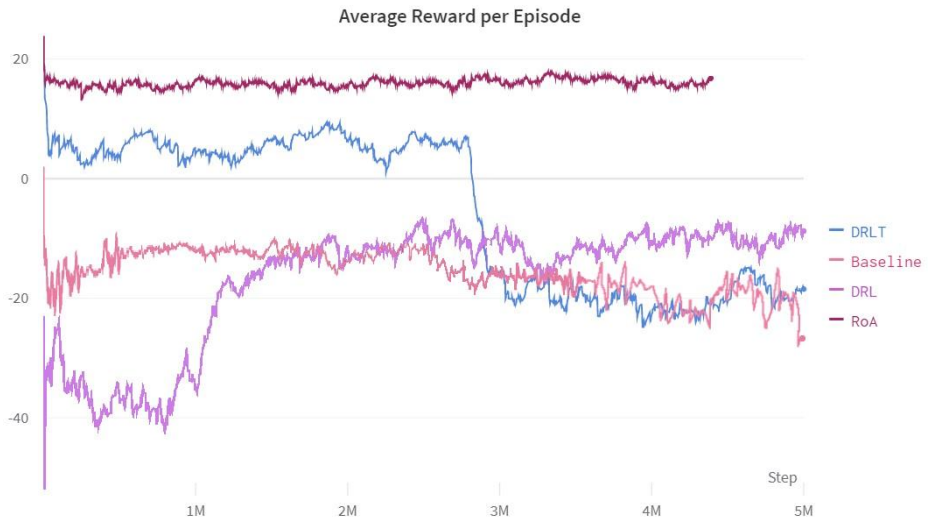

Fig. 4.27 Average reward per episode comparison between DRLT, DRL, RoA and the baseline environment.

As for the evaluation of DRL, the values in table 4.4 show an increase in reward and the conflict avoidance compared to the baseline. Nonetheless, these results are not an extreme enhancement and could not be applied to a real ATC problem. As for a visual evaluation, in some episodes the aircraft avoid conflict perfectly but oftentimes the turns that they make generate conflict that did not exist at first. When the aircraft are in conflict, they show that they can get out of it quickly.

Table 4.4. Evaluation results and comparison with the baseline environment without applying any policy, with green representing an improvement and red a worse result than the baseline environment.

Metrics (avg/episode)	*Baseline*	*DRL*	*Baseline Difference*	*DRLT*	*Baseline Difference*
Reward	**-17.026**	-11.565	↑ 32.07%	-24.935	↓ 46.45%
Dist. left to target (NM)	**$7.78 \cdot 10^3$**	$1.80 \cdot 10^5$	↑ 2222.01%	$7.05 \cdot 10^5$	↑ 8973.79%
Extra dist. (NM)	**0.000**	71.895	↑ 71.895	163.254	↑ 163.25
Min. separation dist. (NM)	**3.176**	3.186	↑ 0.03%	2.814	↓ 11.40%
Nº of turns	**0.00**	1299	↑ 1299	217.01	↑ 217.01
Aircraft conflicts	**1.81**	1.78	↓ -1.66%	1.73	↓ -4.42%
Simult.conflicts	**15.73**	14.62	↓ -7.06%	16.68	↑ 6.04%
Success rate (%)	**19**	19	0%	15	↓ 21%

As for the evaluation of DRLT, the values in table 4.4 show a decrease in reward and the conflict avoidance compared to the environment without applying any policy. As for a visual evaluation, in some episodes the aircraft avoid conflict however most of the times conflict is not avoided and even more conflicts are generated by the actions of the aircraft.

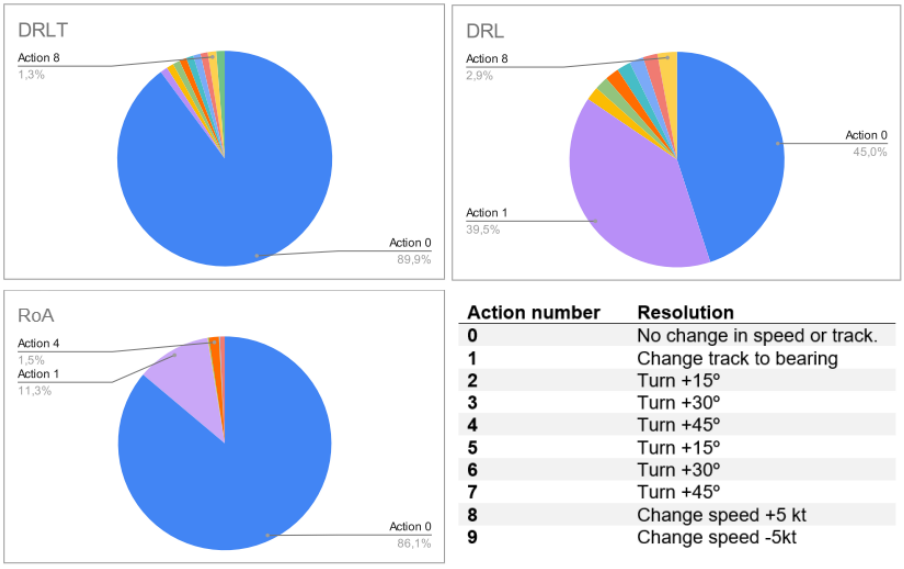

Fig. 4.28 Percentage of actions chosen using DRL, DRLT and RoA with the legend of action resolutions corresponding to the action numbers.

The difference in results is traduced to a difference in the actions taken by the agents. Whilst RoA uses “action 0” 86.1% of the time, followed by “action 1” with 11.3%, DRLT prioritises “action 0” above all and the rest of the actions are chosen with equal probability, and DRL choses “action 0” and “action 1” with nearly the same probability.

CHAPTER 5. CONCLUSIONS

The project has demonstrated the possibility of conflict reduction by DRL, concretely a DQN algorithm applied to a flight simulator. The implementation is a shared policy that uses the same neural network for all flights. The airspace simulator represented a high-capacity sector with flights in cruise.

DRL alone reduced simultaneous conflicts by an average of 7.06% compared to the original environment without a conflict avoidance strategy. The reward was 32.07% higher. However, the success rate and the minimum separation distance did not change significantly compared to the baseline environment.

DRLT was developed to improve these results and apply the RoA to the DRL policy. The RoA policy showed an improvement of 99.41%. This run corresponds to the RoA policy by Lidia Fuentes with two modifications: turns limited to ±15º, ±30º and ±45º and speed changes were included in the actions. Despite the promising results of the RoA, transferred learning did not improve them; on the contrary, they were worse than with DRL, where the simultaneous conflicts increased by 6.04% and reward decreased by 46.45% with respect to the baseline.

The execution of the DRL policy was much faster than using RoA, at 1 second per iteration versus 12 seconds per iteration, thus it can be concluded that DRL improves the decision time. As for the feasibility of the solution, the DRL decision model is not suitable to be applied to piloted aircraft or as a complementary tool for the ATCOs. In the comparison between the actions taken by different policies, the case of DRLT, almost 90% of aircraft take no action (action 0). However, in the case of DRL, these actions are more spread over different turns, so that only in 45% of the cases action 0 is taken. This would lead to an impossible workload for the controller and for the pilot.

In contrast, this could be a good solution for unmanned aerial vehicles (UAVs), where fully autonomous conflict avoidance could be implemented, and self-separation would be possible. Furthermore, since only one neural network is used for all aircraft, exactly the same strategy is used for all flights, reducing the unpredictability of using a different policy for each flight, a very important aspect in aviation.

Nevertheless, further improvements need to be made to achieve full conflict avoidance (100% conflict resolution) and a possible implementation proposal. This could be the implementation of different algorithms, such as the actor-critic algorithms, or increasing the complexity of the neural network by adding more layers (currently set to a minimum of 2). In addition, a more realistic model of the environment could be developed by adding the wind component, making the environment a 3D model, or gradual turns that takes more than one step. Finally, a supercomputer would have improved the results by allowing more training.

BIBLIOGRAPHY

[1] Bellman, R. and Stuart E. D. *Applied Dynamic Programming*. Princeton University Press, Princeton (1962)

[2] Dalmau Codina, R. and Allard, E. “Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning.” *10th SESAR Innovation Days (SID)*. Ed. Network (NET) Research Unit. 1-8 Bretigny-Sur-Orge (2020).

[3] EUROCONTROL. *Aviation Intelligence*. 20 August 2022: <https://www.eurocontrol.int/Economics/DailyTrafficVariation-ACCs.html> accessed 20 August 2022.

[4] FAA. “Legal Information Institute.” 27 July 2004. *Part 91 - General operating and flight rules:* <https://www.law.cornell.edu/cfr/text/14/91.113> accessed on 10 August 2022.

[5] Farley, B. and Wesley C. “Simulation of Self-Organizing Systems by Digital Computer.” *Transactions of the IRE Professional Group on Information Theory*. 76-84 (1954): <https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/stamp/stamp.jsp?tp=&arnumber=1057468> accessed 1 August 2022.

[6] Fuentes, L. “Air Traffic Control using separation algorithm based on Rules of the Air.” Universitat Politècnica de Catalunya, EETAC, 29 June 2022.

[7] Girdhar, Y., Whitney D. and Dudek G. “Curiosity Based Exploration for Learning Terrain Models.” *2014 IEEE International Conference on Robotics & Automation (ICRA)*. Hong Kong: Hong Kong Convention and Exhibition Center, 578-584 (2014): <https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/stamp/stamp.jsp?tp=&arnumber=6906913> accessed 10 August 2022.

[8] J Hood, J. *Reinforcement Learning: Temporal Difference (TD) Learning* (2021): <https://www.lancaster.ac.uk/stor-i-student-sites/jordan-j-hood/2021/04/12/reinforcement-learning-temporal-difference-td-learning/> accessed 24 July 2022.

[9] Jang, K, et al. “Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles.” *the 10th ACM/IEEE International Conference*. Montreal, 291-300, (2019).

[10] Kalagirou, S. A. “Designing and Modeling Solar Energy Systems.” *Solar Energy Engineering.* 1-14 (2014).

[11] Keskar, N. S., et al., “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.” *International Conference on Learning Representations (ICLR)*. San Juan, 1-16 (2016).

[12] Kingma, D. P. and Ba, J. “Adam: A Method for Stochastic Optimization.” *ArXiv* , 1-15 (2014): <https://arxiv.org/pdf/1412.6980.pdf> accessed on 20 June 2022.

[13] Leffer, L. “One of the World's Busiest Airports Will Cap Passenger Numbers at 100,000 Per Day.” *Gizmondo* (2022): <https://www.msn.com/en-us/travel/news/one-of-the-worlds-busiest-airports-will-cap-passenger-numbers-at-100000-per-day/ar-AAZuIdw> accessed on 20 July 2022.

[14] Li, Y. “Deep Reinforcement Learning.” *ArXiv.* 1-150 (2018): <https://arxiv.org/pdf/1810.06339.pdf> accessed on 22 July 2022.

[15] de Luca, G. *ML Policy Reinforcement Learning* (2020): <https://www.baeldung.com/cs/ml-policy-reinforcement-learning> accessed on 12 June 2022.

[16] Mnih, V., et al. “Human-level Control through Deep Reinforcement Learning.” *Nature*. issue 518, 529-533 (2015): <https://www.nature.com/articles/nature14236> accessed on 14 July 2022.

[17] Pham, D., et al. “A Machine Learning Approach for Conflict Resolution in Dense Traffic Scenarios with Uncertainties.” *Thirteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2019)*. Vienna: AI Team, 1-12, (2019): <https://dr.ntu.edu.sg/bitstream/10356/146568/2/ATM_Seminar.pdf> accessed on 20 March 2022.

[18] PyTorch Contributors. *Smooth L1 Loss*. (2020): <https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html> accessed on 10 June 2022.

[19] Reddi, S. J., Kale S. and Kumar, S. “On the Convergence of Adam and Beyond.” *ArXiv* , 1-23. (2019): <https://arxiv.org/pdf/1904.09237.pdf> accessed on 21 June 2022.

[20] Russo, S. and Van Roy, B. “An Information-Theoretic Analysis of Thompson Sampling.” *Journal of Machine Learning Research, 1-30* (2016): <https://www.jmlr.org/papers/volume17/14-087/14-087.pdf> accessed on 10 August 2022.

[21] SESAR JU. *SESAR Joint Undertaking*. (2018). <https://www.sesarju.eu/index.php/projects/dart> accessed on 20 July 2022.

[22] Sutton, R. S. and Barto, A.G. *Reinforcement Learning: An Introduction*. The MIT Press, Bradford (2015).

[23] Tse, H- T. and Leung, H. “Exploiting Semantic Epsilon Greedy Exploration Strategy in Multi-Agent Reinforcement Learning.” *ArXiv*, 1-8

(2022): <https://arxiv.org/pdf/2201.10803v2.pdf> accessed on 10 August 2022.

[24] Werbos, P. J. “Generalization of Backpropagation with Application to a Recurrent Gas Market Model.” *Neural Networks: the official journal of the International Neural Network Society* 1.4, 339-356 (1988): <https://www.researchgate.net/publication/223074905_Generalization_of_Backpropagation_with_Application_to_a_Recurrent_Gas_Market_Model> accessed on 16 July 2022.

[25] Wunder, M., Littman, M. L. and Babes, M. “Classes of Multiagent Q-Learning Dynamics with epsilon-greedy Exploration.” *International Conference on Machine Learning*. Haifa, 1167-1174 (2010).

[26] z_ai. *The Good, the Bad, and the Ugly: Supervised, Unsupervised and Reinforcement Learning*. (2020): <https://towardsdatascience.com/the-good-the-bad-and-the-ugly-supervised-unsupervised-and-reinforcement-learning-2ccf814c6bab> accessed on 22 August 2022.

[27] Zaheer, M., et al. “Adaptive Methods for Nonconvex Optimization.” *Advances in Neural Information Processing Systems,* 31, 9793–9803 (2018).

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEX

TITLE: Aircraft-to-Aircraft separation based on Reinforcement Learning

DEGREE: Aeronavigation Engineering

AUTHOR: Weronika Prawda

ADVISOR: Cristina Barrado Muxí

DATE: September 5th, 2022

APPENDIX A. MULTI-AGENT REINFORCEMENT LEARNING

Multi-agent reinforcement learning (MARL) takes place when the environment is shared between two or more agents. This adds a difficulty to the training as the actions of one of the agents can influence the decisions of the other ones.

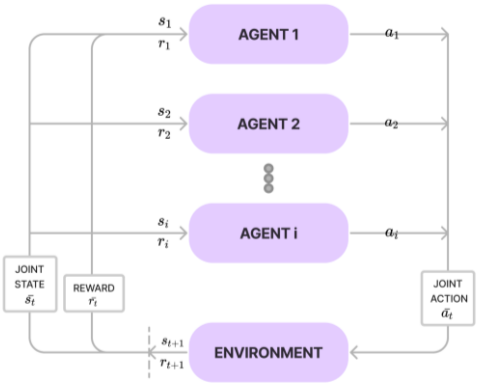

Fig. A.1 Multi-agent interaction with the environment, where the actions, the states and the reward are joint.

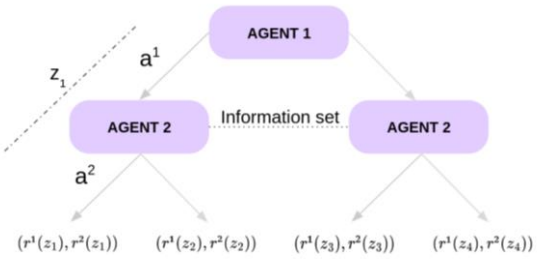

Fig. A.2 Multi-agent interaction with the environment, extensive-form approach.

In a multiagent setup, several agents share the environment in which they are in. So, for this case the diagram shown before could look like the one in figure A.2, where the actions of each agent form a joint action $\underline{a_t}$ and the environment returns a joint $\underline{s_t}$ state and reward $\underline{r_t}$ to the agents. These joint elements normally are in the form of an array where each element corresponds to each agent. Another type of system would be an extensive-form approach where agents take decisions alternately and receive their individual reward at the end, like in figure A.2.

If the setup is multi-agent, MDPs (see section 1.3 of the document) are called Markov Games [3] that are represented by states' set S, action sets for each agent A_i, an associated reward function and with the goal to maximise the sum of discounted rewards (see equation A.1).

$$G_t = E\left\{\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right\} \qquad \textbf{(A.1)}$$

The agents can interact with each other and with the environment in various ways in line with the tasks that they are supposed to do. They may compete for the reward, collaborate to maximise it, or a mix of the two, generally in teams.

To better understand these three approaches, let's dive into more specific examples.

- *Cooperative environments*
  Cooperative situations are also called multi-agent MDPs or Markov games [3]. In these environments the agents usually share the same reward function and goal, so formally: $R^1 = R^2 = \cdots = R^N = R$.

  The Q-value function for the agents is identical, so there are ways to approach it in a centralised manner, where all the agents are engulfed as one decision maker and all the actions are taken simultaneously. However, normally this approach leads to not considering the impact of the actions of the other agents in the environment.

  Another viewpoint would be to have agents with different reward functions and considering a team-average reward [1], where the final goal is to optimise the average of the reward functions as the long-term goal, where its function would look like the following equation.

$$R(s,a,s') := N^{-1} \cdot \sum_{i \epsilon N} R_i(s,a,s') \; for \; any \; (s,a,s') \in S \times A \times S \qquad \textbf{(A.2)}$$

  This helps with the decentralisation of the processes and brings certain individuality to the agents.

  This individuality brings the need for communication between agents, as they no longer take decisions as one, so they no longer have the state of each agent being considered while choosing the actions. On one hand, the communication can be very simple, in the form of sharing information from other agents or sharing the reward. On the other hand, it can reach more complexity, such as the prediction of the moves of each agent in order to decide, called Intention sharing [5] or using meta-learning to do so [4].

  These setups have several broad examples, from air traffic control to the game Overcooked.

- *Competitive environments*
  In competitive environments agents compete for a goal. Normally they are formalised as a zero-sum game, where the reward should be 0 for any state, $\sum_{i \epsilon N} R_i(s,a,s') = 0 \; for \; any \; (s,a,s')$. Oftentimes these games are associated with one-vs-one games, where the reward of one agent is donated to the other, so when one loses, the other one wins the same number of points.

- *Cooperative-competitive environments*

These environments are a mix of both cooperative and competitive strategies, normally associated with team versus team setups. In this case there are no restrictions to the goal or a typical modelling of the reward, It could be just two cooperative environments competing with each other or something more entangled. One example of this setting could be a predator-prey ecosystem.

There are many algorithms to choose from in the case of multi-agent reinforcement learning, but they are difficult to use, and the available codes are specific to one environment. However, they can be adapted if understood well.

In contrast to a single-agent problem, there are a few challenges that these algorithms face, the first one being scalability.

When the agent number increases, the action and observation spaces also increase, which leads to a need of more computational resources and memory in order to perform the training and use the reinforcement learning model. This is especially notable for centralised systems, where all the information from all the agents must be processed at once. Decentralised systems could be the response to that problem, but they have other issues.

The second challenge is the non-stationarity of the environment. Opposed to a single-agent setup where the agent is only concerned with its actions, now the agents must consider their actions and the actions of the other agents, which constantly reshape the environment and make it non-stationary. This leads to problems regarding the inaccuracy of the policy where the outcome is very much dependent on the other agents' states. This issue is typical of decentralised environments without good communication.

The last issue is the partial observability. Most of the algorithms that exist assume the possibility of having a complete image of the environment, which is not always the case in real life.

This problem can be avoided by modelling a partial observation Markov decision process, where the probability of the observations O is added to an MDP tuple $(S, A, T, R, \Omega, O, \gamma)$.

REFERENCES

[1] Kar, S., Moura, J.M., Poor, H.V.: “QD-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovation”s. IEEE *Transactions on Signal Processing* 61(7), pp. 1848–1862 (2013

[2] Kim, Woojun, Jongeui Park and Youngchul Sung. «Communication in Multi-Agent Reinforcement Learning: Intention Sharing.» *International Conference on Learning Representations (ICLR)*. (2021).

[3] Owen, Guillermo. *Game Theory*. 2nd. New York: Academic Press, (1982). Online. <https://openlibrary.org/books/OL3484527M/Game_theory#details>.

[4] Raileanu,R., Denton, E., Szlam, A. and Fergus, R.. Modeling others using oneself in multi-agent reinforcement learning. ArXiv preprint, *ArXiv*:1802.09640 (2018)

[5] Zhang, Kaiquing, Zhuoran Yang and Tamer Basar. «Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms.» *ArXiv* (2021).

APPENDIX B. CODE ARCHITECTURE

The entire code can be found in vera15380/conflict-avoidance-TFG-WP (github.com).

The code is divided into different files depending on the functionality.

- Calcs.py is where the auxiliary functions that are used in more than one file are stored, for example the t_{CPA} function.
- Definitions.py is where the airspace and flight classes are defined.
- In env.py the reward, the resolution of the actions and the observation functions can be found, aside from the update, step and render functions.
- In the rl.py file all the classes related to reinforcement learning are saved. This includes the replay buffer class, neural network class and the DQN algorithm with the action selection, update, and learning functions.
- The units.py file contains the conversions from the International System of Units (SI) to units used in aviation.
- In the wandb_graphs.py there are functions to represent graphically and store information about the training and evaluation, in Wandb and in Tensorboard in case of failure of one of the two.
- The main.py file contains the setup of constants for the training and the loops to train with the Rules of the Air and DQN.

Figure B.1 explains the content of the different files found in Github and figure B.2 goes into detail of what can be found in the file atcenv.py.

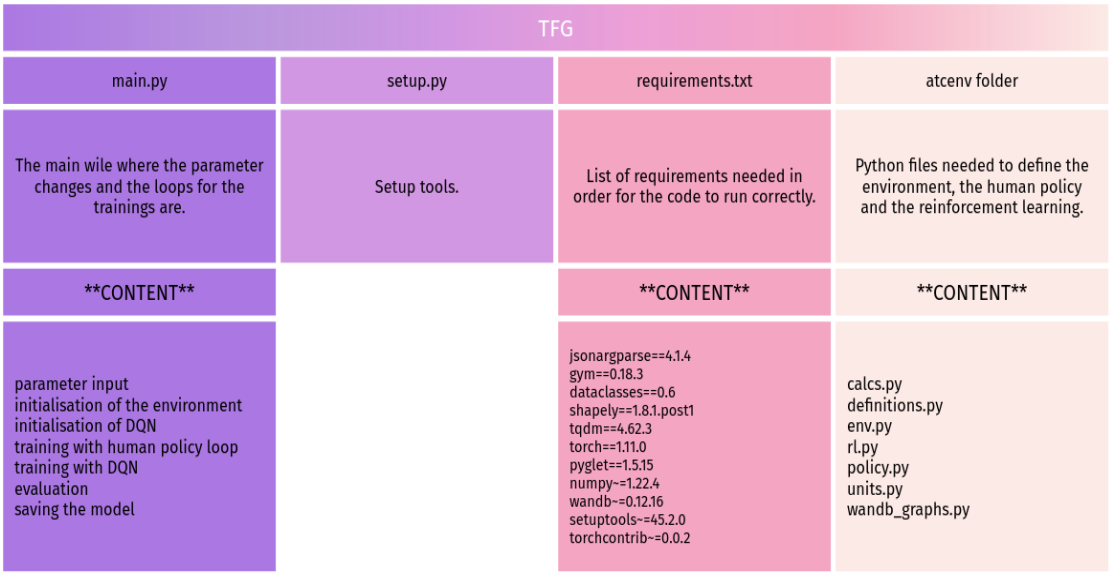

Fig. B.1. Diagram of the content of the code (first layer).

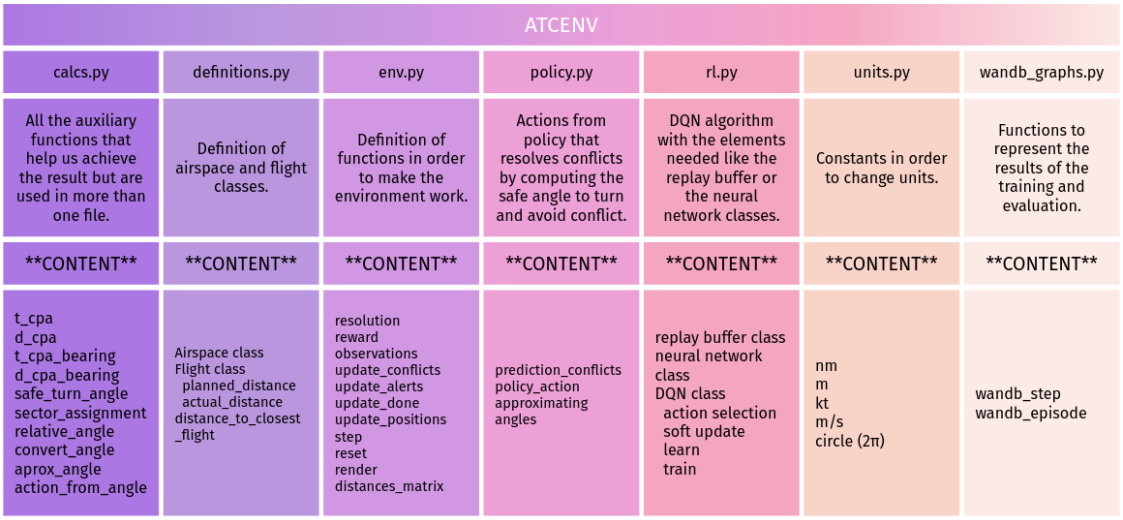

Fig. B.2 Diagram of the content of the atcenv folder and its code.

APPENDIX C. HYPERPARAMETER TUNING RESULTS

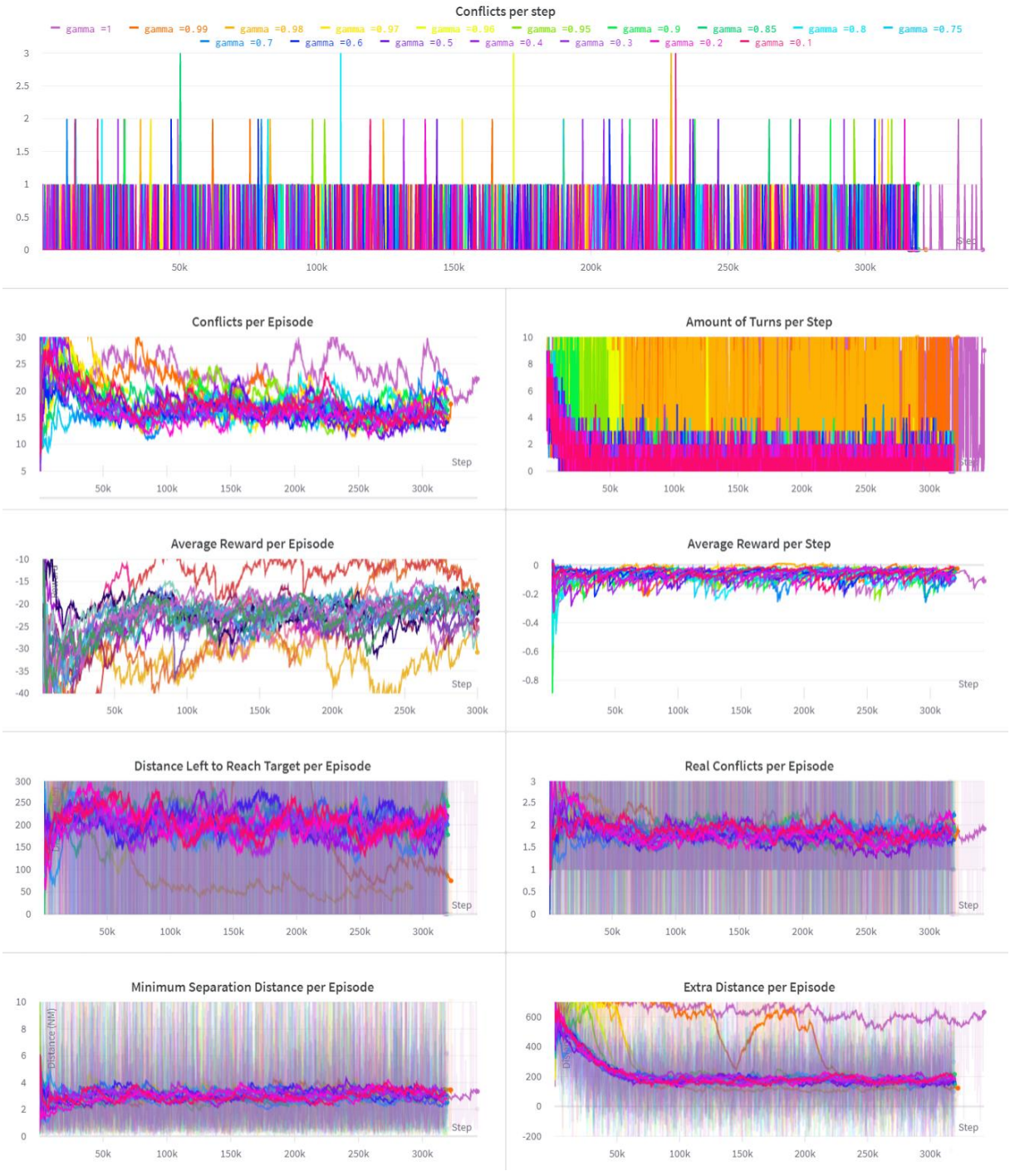

Fig. C.1 Graphs of hyperparameter tuning gamma.

Table C.1. Average results of the evaluation depending on gamma.

Gamma value	Conflict/episode	Distance to target/ep (NM)	Extra distance/ep (NM)	Min. Separation/ep (NM)	N. turns/ep	Real conflicts/ep	Reward/ep	Success rate eval
γ=0.10	15.80	705764.6	183.88	2.82	221.22	1.69	-21.7443	19
γ=0.20	15.45	714396.2	185.36	2.92	220.42	1.83	-20.2489	18
γ=0.30	13.73	736444	178.69	3.31	217.54	1.66	-17.8747	21
γ=0.40	12.63	733071.3	157.71	3.62	221.27	1.54	-15.7749	22
γ=0.50	15.15	753911.2	150.74	3.29	217.46	1.74	-21.8553	24
γ=0.60	13.17	718265.7	182.08	3.05	224.9	1.64	-18.3068	18
γ=0.70	13.44	744588	136.44	3.66	214.98	1.59	-19.7417	27
γ=0.75	16.26	679347.8	179.68	3.18	219.94	1.76	-20.6940	19
γ =0.80	16.41	756728.5	162.84	2.84	219.84	1.72	-21.5975	19
γ=0.85	16.87	707665.7	162.00	2.83	218.72	1.89	-21.5066	17
γ=0.90	17.25	726309.1	197.92	2.73	225.10	1.96	-21.7886	14
γ=0.95	17.27	710292.6	192.44	2.91	220.02	1.82	-23.8251	18
γ=0.96	18.49	726481.3	186.01	2.89	223.03	1.92	-24.4192	22
γ=0.97	15.95	727029.6	178.26	2.69	220.29	1.80	-21.7772	16
γ=0.98	**14.87**	**679847.3**	**144.50**	**3.62**	**207.46**	**1.68**	**-19.7779**	**23**
γ=0.99	12.99	164353.2	59.42	2.82	1394.95	1.64	-7.85734	19
γ=1.00	0.72	1562465	1059.36	15.52	2912.11	0.11	-1.58261	93

Fig. C.2 Graphs of hyperparameter tuning tau.

Table 4.2. Average results of the evaluation depending on tau.

Tau value	Conflict/episode	Distance to target/ep (NM)	Extra distance/ep (NM)	Min. Separation/ep (NM)	N. turns/ep	Real conflicts/ep	Reward/ep	Success rate eval
τ=0.1	15.8	705764.6	183.8888	2.820306	221.22	1.69	-21.7443	19
τ=0.2	15.45	714396.2	185.3639	2.926843	220.42	1.83	-20.2489	18
τ=0.3	15.98	739725.0	182.8851	2.845074	222.47	1.71	-18.9995	16
τ=0.4	19.41	746403.7	169.4548	2.853679	223.60	2.12	-23.5515	16
τ=0.5	17.07	712537.6	156.8700	2.958805	220.28	1.83	-22.9872	14
τ=0.6	15.01	715991.9	237.5440	2.894260	224.14	1.87	-21.2840	16
τ=0.7	16.10	729329.1	193.1994	3.012887	219.21	1.79	-22.7135	22
τ=0.8	14.54	766096.3	203.3953	3.331865	228.40	1.65	-20.5196	27
τ=0.9	17.55	728369.9	179.1316	2.971296	219.88	1.78	-23.8385	16
τ=1.0	**15.16**	**744628.8**	**183.0338**	**3.573278**	**222.39**	**1.54**	**-21.1014**	**25**

Fig. C.3 Graphs of hyperparameter tuning the hidden neurons.

Table C.3. Average results of the evaluation depending on the number of hidden neurons.

Hidden neurons value	Conflict/episode	Distance to target/ep (NM)	Extra distance/ep (NM)	Min. Separation/ep (NM)	N. turns/ep	Real conflicts/ep	Reward/ep	Success rate eval
hn=32	16.41	693290.7	181.7419	2.631968	221.33	1.79	-22.4604	15
hn=64	**14.1**	**726214.9**	**180.9207**	**3.246031**	**220.26**	**1.72**	**-18.0490**	**21**
hn=128	17.47	751845.7	200.83	2.78099	224.54	2.05	-22.6656	16
hn =256	16.28	741141.4	182.8497	2.861823	221.66	1.76	-24.0137	22
hn=512	19.38	705955.2	178.4158	2.868479	224.86	2.08	-23.2801	17
hn=1024	14.38	702727.9	181.5375	3.322088	227.17	1.68	-20.2692	23

Fig. C.4 Graphs of hyperparameter tuning the learning rate.

Table C.4. Average results of the evaluation depending on the learning rate.

Learning rate value	Conflict/episode	Distance to target/ep (NM)	Extra distance/ep (NM)	Min. Separation/ep (NM)	N. turns/ep	Real conflicts/ep	Reward/ep	Success rate eval
lr =1	15.8	705764.6	183.8888	2.820306	221.22	1.69	-21.7443	19
lr = 0.1	15.45	714396.2	185.3639	2.926843	220.42	1.83	-20.2489	18
lr = 0.01	15.98	739725	182.8851	2.845074	222.47	1.71	-18.9995	16
lr = 0.001	14.92	762505.3	162.1336	3.643531	223.52	1.55	-22.728	27
lr = 0.0001	**14.06**	**704054.9**	**162.0858**	**3.706851**	**218.69**	**1.53**	**-18.4025**	**25**
lr = 1e-05	15.38	735892.8	176.3586	3.108465	222.65	1.69	-20.1481	21
lr = 1e-06	17.16	694668	174.356	2.886834	219.15	1.95	-23.2004	16
lr = 1e-07	17.9	818672.5	194.1866	3.264188	259.94	1.96	-23.3391	18
lr = 1e-09	14.62	1223299	162.2595	4.118106	323.69	1.51	-22.1512	28
lr = 1e-10	15.72	617504	178.5235	3.042452	606.47	1.84	-18.3367	19

Fig. C.5 Graphs of hyperparameter tuning the batch size.

Table C.5. Average results of the evaluation depending on the batch size.

Batch size value	Conflict/episode	Distance to target/ep (NM)	Extra distance/ep (NM)	Min. Separation/ep (NM)	N. turns/ep	Real conflicts/ep	Reward/ep	Success rate eval
batch size=16	15.8	705764.6	183.8888	2.820306	221.22	1.69	-21.7443	19
batch size=32	20.39	686766.1	183.1232	2.935089	215.22	2.11	-25.5489	18
batch size=64	**13.92**	**699969.7**	**162.0731**	**3.797281**	**217.3**	**1.51**	**-19.058**	**29**