Airport Assignment for Emergency Aircraft using Reinforcement Learning

Saketh Kamatham
San Jose State University

Airport Assignment for Emergency Aircraft using Reinforcement Learning

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Saketh Kamatham

May 2023

The Designated Project Committee Approves the Project Titled

Airport Assignment for Emergency Aircraft using Reinforcement Learning

by

Saketh Kamatham

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

ABSTRACT

Airport Assignment for Emergency Aircraft using Reinforcement Learning

by Saketh Kamatham

The volume of air traffic is increasing exponentially every day. The Air Traffic Control (ATC) at the airport has to handle aircraft runway assignments for landing and takeoff and airspace maintenance by directing passing aircraft through the airspace safely. If any aircraft is facing a technical issue or problem and is in a state of emergency, it requires expedited landing to respond to that emergency. The ATC gives this aircraft priority to landing and assistance. This process is very strenuous as the ATC has to deal with multiple aspects along with the emergency aircraft. It is the duty of the ATC to direct the aircraft to the place that is equipped with handling the aircraft. If the ATC does not have immediate answers to the requests from the pilots, then it might result in an aircraft crash and loss of life. This project aims to solve this problem by building a model using Reinforcement Learning that can map an emergency aircraft to the airport in the shortest amount of time.

Keywords: Reinforcement Learning, Air Traffic Control, Assignment, Airport Assignment, Deep Reinforcement Learning

ACKNOWLEDGMENTS

I want to thank my parents Lakshmi and Diwakar, and my Uncle Ravi for their constant support and guidance throughout my life. This would not be possible without them. Additionally, I want to thank Dr. Genya Ishigaki for his guidance and patience in helping me with this endeavor. I also want to thank Professor Faranak Abri and Professor Saptarshi Sengupta for their support and guidance for the project. Lastly, I want to thank my friends and housemates for motivating me and helping me succeed.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

The responsibility of air traffic control (ATC) is to ensure the safe and efficient movement of airplanes in the airspace. In the case of an emergency, ATC must be able to route the aircraft to the nearest airport while also ensuring the safety of the airspace. The process is very stressful on the ATC as it involves managing a lot of aircraft both on the ground and in the air. An Emergency aircraft adds on top of this stress as it requires priority assistance and any mistake from the ATC can result in catastrophic results. There is a need for a more intelligent system that can automate the handling of the airspace to ensure that all aircraft reach their intended destinations safely. In this project, I propose employing reinforcement learning (RL) to create an ATC agent capable of guiding an emergency aircraft to the airport while ensuring airspace safety.

RL is a form of artificial intelligence that involves an agent interacting with an environment and learning to attain a goal through trial and error. The agent would be in charge of directing an emergency aircraft to the airport while avoiding any obstacles in the setting of ATC. Based on its actions, it would receive feedback in the form of rewards or penalties, such as successfully guiding the emergency aircraft to the airport going out of bounds.

The intricacy of the environment is a significant barrier in designing an RL agent for ATC. To avoid accidents, the agent would need to assess various aspects in real time and make decisions swiftly. As a result, complex algorithms and advanced machine learning techniques would be required to effectively navigate the airspace and lead the emergency aircraft to the airport.

The ability of RL to learn from experience is a big advantage of employing it for this problem. RL algorithms can adapt to changing environments and optimize their decision-making. This means that the RL agent's performance may increase with time, making it more efficient and successful in directing emergency planes to the airport. Another benefit of adopting RL for ATC is its capacity to assess numerous parameters at the same time. Traditional rule-based systems frequently rely on a predefined set of rules that might be restrictive and inflexible. RL agents, on the other hand, can consider several variables in real-time, allowing them to make more informed judgments that take into account all relevant factors.

Finally, using reinforcement learning for ATC can potentially increase airspace safety and efficiency in emergency situations. We can save lives and effectively manage crucial situations by designing an RL agent that can guide emergency planes to the airport while avoiding any obstacles. While developing such an agent is difficult, the benefits it may give are substantial, making it a potential field of research for the future of aviation.

In summary, the main target of this project is to route an emergency aircraft to the nearest airport in the least amount of time without causing loss of separation with obstacles using reinforcement learning.

CHAPTER 2

Background

Machine learning, which is a sub-field of artificial intelligence is one such advancement that is improving every day. Machine learning itself has various sub-fields like supervised learning which basically means we are training a model with data that is labeled and the correct output is available when the model trains, unsupervised learning which mainly focuses on unlabeled data and tried to detect patterns among them, and reinforcement learning which is an incentive-based agent training approach which vast range of applications.

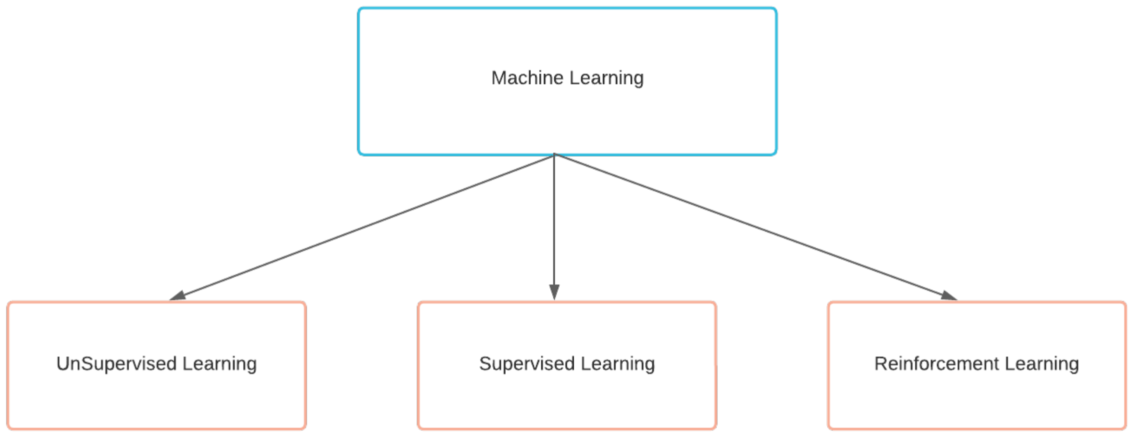


Figure 1: Three Categories in Machine Learning

2.1 Reinforcement Learning

Reinforcement learning is a subset of Machine Learning (ML) that relies on the output given by the environment to take certain actions. The reinforcement learning model gives us information about the action that would produce the highest reward if taken. In reinforcement learning, the actions that we take not only affect the immediate rewards but also have an effect on the scenario that arises. The agent is not given any information regarding which action to be taken, instead, the agent in reinforcement learning observes the state of the environment and takes actions intending to maximize the long-term reward. This interaction between the agent and the environment is modeled in a Markov Decision Process (MDP). A way to solve optimal control problems was given by Richard Ernest Bellman. He called it dynamic programming [11]. This approach used reinforcement learning for solving MDPs. Apart from the environment and the agent, reinforcement learning has various other sub-components.

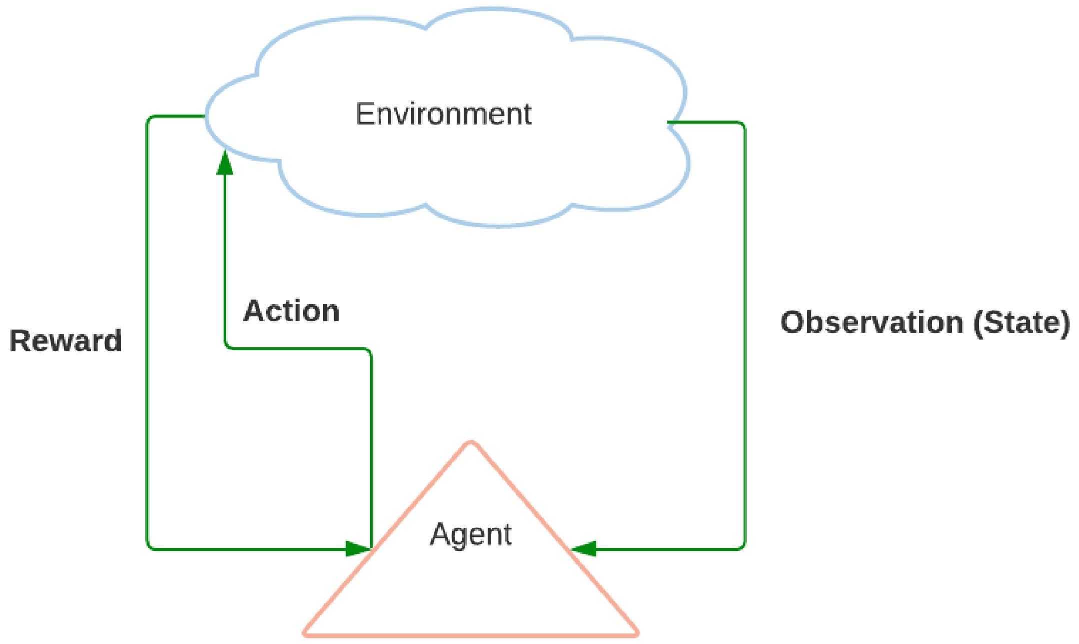


Figure 2: Reinforcement Learning Steps

2.1.1 Policy

In simple words, we can define *policy* as the way any agent will behave at any time. In a reinforcement learning environment, we have certain perceived states and actions that we can take. A *policy* is a mapping between these states and actions that we can take when we are in those states. The complexity of the policy depends on the scenario. It can be a simple function as well as a computation-extensive calculation based on the problem. The policy is considered very vital in reinforcement algorithms as using policy alone, we will be able to determine the agent's behavior [8].

2.1.2 Reward Signal

In reinforcement learning, the agent takes an *action* in the *environment* and received a *reward* from the environment. This receiving of the reward is called the *reward signal*. This reward signal is vital in deciding what the goal of the algorithm is. The main goal of the agent is to achieve maximum reward in the long run. Defining what actions will be 'bad' and 'good' for the problem can be conveyed by the reward signal.

2.1.3 Value Function

Using the *reward signal*, the agent will know what is 'good' in an immediate sense but to understand what is 'good' in the longer run, we use a $value\ function$. The value of the state can be defined as the total reward the agent can expect in the long run when it starts from that state.

2.1.4 Model

Using the model, we can understand how the environment will behave. In a broader sense, we can use a model to plan after which we can decide what our actions will be. When we use models to learn, we call it $model - based$ learning.

To understand reinforcement learning, one can consider the following example where a chess grandmaster makes a move. This move is decided based on factors like planning, analyzing what the possible next move might be, and then planning for a reply to counter this. In another example, consider a newborn calf. Initially, just after birth, the calf struggles to stand up but after a few hours, it can walk and run. In both these cases, there is an interaction between an agent who makes decisions and the environment [8].

Agent training has experienced a lot of breakthroughs. One example is Deep

Q-Learning [DQN] which incorporates deep neural networks with the aim to model the state-action function accurately. This report aims to use this technique to implement a deep learning model that is able to map an emergency aircraft to the nearest airport in the least time.

2.2 Drawbacks of Reinforcement Learning

Reinforcement learning is an effective technique for teaching machines to learn and develop through trial and error. However, there are several disadvantages to this strategy. One significant issue is the distinction between exploration and exploitation. To learn the best policy, the agent must experiment with various actions and investigate its surroundings. However, this exploration can be time and money intensive. However, if the agent only uses its current knowledge, it may lose out on finding better solutions [8].

The problem of reward shaping is another disadvantage of reinforcement learning. Only the reward signal given by the environment allows the agent to learn. If the reward indication is insufficient or misleading, the agent may have difficulty learning the best policy. Furthermore, designing a reward function can be difficult and time-consuming, and poorly designed rewards can result in unintended behaviors [8].

Finally, reinforcement learning has the potential to be computationally costly. Many iterations of trial and error are required to train an agent to execute a task, which can be time-consuming and resource-intensive. Furthermore, as the task's complexity grows, the state space and action space can become prohibitively big, making it difficult to find an optimal policy [8].

2.3 Neural Networks

2.3.1 Perceptron

A perception, which was developed in the 1950s by Frank Rosenblatt is a type of artificial neuron.

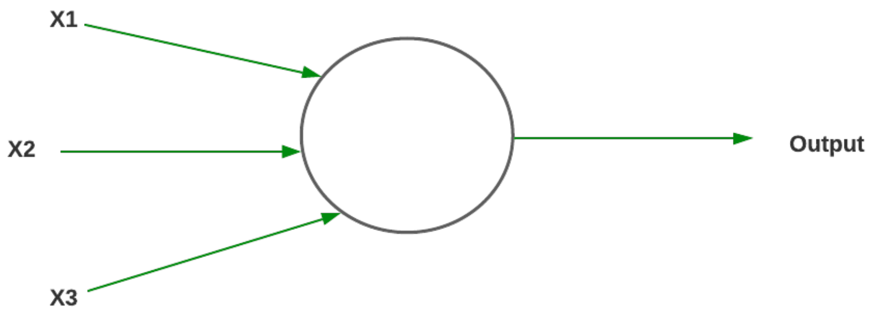


Figure 3: A simple Perceptron

A perceptron will take binary inputs to give a single binary output. From Figure 3, we can see that x_1, x_2, and x_3 are the binary inputs that give out a single binary *output*. Using the inputs, we determine the output. But there were cases where some inputs had more significance towards the output than others. For this, Frank Rosenblatt used *weights*. These *weights* are real numbers that convey how important each input is.

$$Let\ t = Threshold \tag{1}$$

and

$$Let\ w_1, w_2, w_3 = weights \tag{2}$$

then,

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} [1] \tag{3}$$

This is a very basic neuron and it is evident that a more complex network is required when working on high-intensity or human-level data.

2.3.2 Neural Network

A neural network is a collection of multiple neurons arranged in multiple layers.

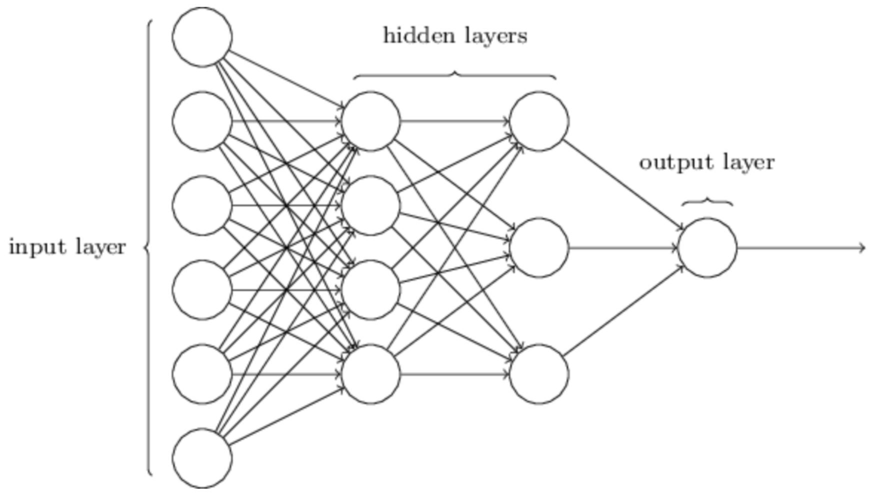


Figure 4: Neural Network Architecture [1]

The layer that is the most to the left is called as input layer followed by the hidden layer/s to its right and then the output layer which is the right-most layer. Neural networks have various applications and advantages when used. One such application is Deep Learning. A neural network that has a lot of hidden layers can be called a Deep neural Network. By using deep neural networks, complex tasks like image analysis, speech recognition, AI gaming, Natural language processing, etc can be done very easily. Neural networks are very flexible when it comes to the kind of data they can work with. They are known to work easily with almost all kinds of data. Because of this, the use cases for neural networks can range from a simple prediction to a self-driving car.

CHAPTER 3

Related Works

Air transport is one of the most used modes of transport in modern times. Air traffic is a term used to refer to the movement of aircraft in the airspace. This movement includes the aircraft taking off from the airport, the aircraft landing at the airport, and the in-between movement of the aircraft during the flight. In recent times, there has been substantial growth in the volume of air traffic. With this exponential growth in traffic, it brought many changes in the way people travel around the world. Air traffic also has a significant role in the transportation of necessary goods, emergency assistance, medical supply delivery, and many more.

The reason that there is an exponential growth in air traffic is because there has been an exponential growth in population. As per the data given by the International Aviation Transport Association (IATA), an organization that handles air travel around the world, about 178 million passengers traveled using an aircraft in the year 2022 and it is increasing every day. With this increase, there is an increase in traffic congestion, safety concerns, and accidents and it is up to the air traffic control to handle all these and ensure safe passage to all aircraft in a timely manner. The current air traffic control operations extensively depend on human controllers [3]. Maintaining the loss of separation is an important aspect that all air traffic controllers consider.

3.1 Loss of Separation

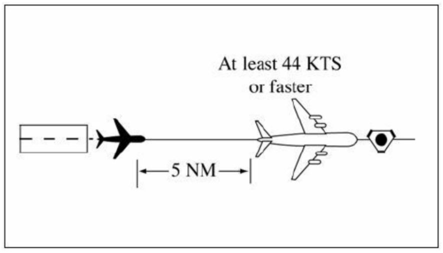
Loss of separation is a phenomenon defined when, in airspace, two aircrafts come too close the each other which violated the minimum separation distance that is decided by the authorities. This is a major safety aspect that every air traffic controller considers. The minimum separating distance is in place so that the safety of the aircraft and in turn, the airspace is maintained. Various aspects like errors by the pilot, errors by the controller, failure in communication systems, and mechanical or technical errors can cause loss of separation. It is very vital for air traffic controllers to constantly check the positions of each aircraft and make sure the separation is not lost.

Both in manned and unmanned aircraft, there is a need for an early warning for safety incidents as airspace congestion is increasing. for this problem, M. Hawley and R. Bharadwaj [3] utilized machine learning to detect statistical, safety anomalies within national airspace(NAS). This is particularly challenging as there is less availability of labeled anomaly data and using unsupervised learning would result in hard-to-interpret results.

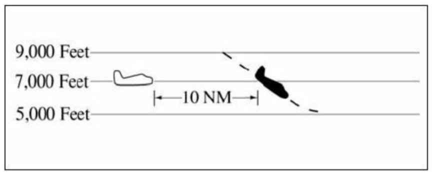
In their research, they focus on multiple aircraft anomalies. For their framework, an anomaly is defined as a possible loss of separation event. Their model works from an air traffic controller standpoint. They used the Aviation Safety Reporting System (ASRS) data which has data pertinent to anonymous safety incidents. They also used the FAA SWIM sources such as ASDE-X and TBFM data. They considered self-separation and a calculation between the aircraft to identify these anomalies. They also used the Traffic Collision Avoidance System (TCAS) in the aircraft to identify loss of separation events. TCAS usually issues two distinct advisories, one is the Traffic advisory (TA) and the other is Resolution Advisory (RA). If an RA is issued, it means a critical collision event is imminent and if a TA is issued, it means that a collision might be possible.

Using a machine learning model, three common situations where the aircraft came under the separation threshold were identified. They used a reinforcement learning algorithm to detect potential loss of separation events and the model was successful.

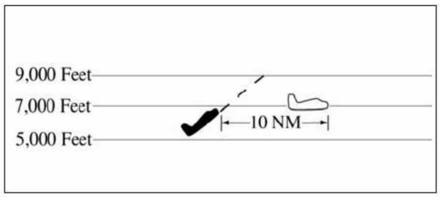
Another aspect that is very vital in airspace management is runway assignment. There have been improvements in arrival management and optimal runway assignment strategies around the world. Optimizing the runway assignment will increase the



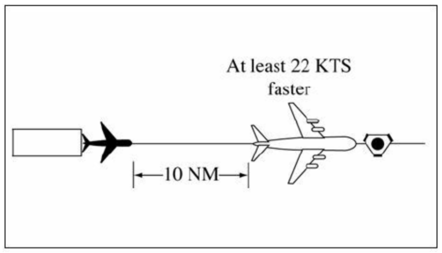
(a) a



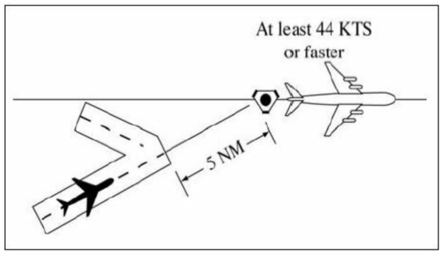
(b) b



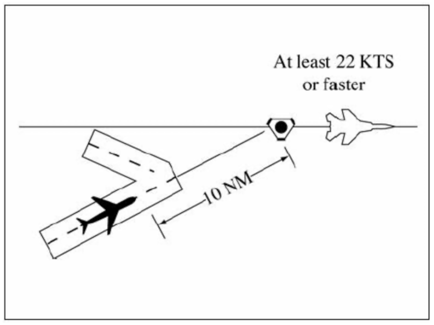
(c) c



(d) d



(e) e



(f) f

Figure 5: United States Separation Standards [2]

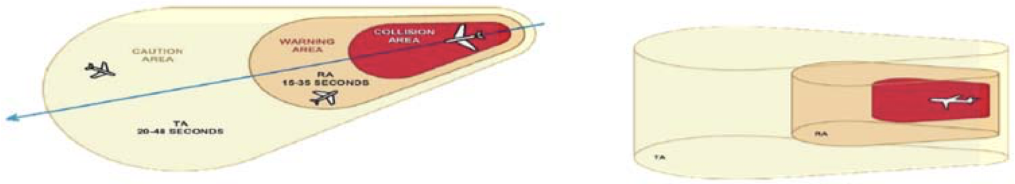


Figure 6: TCAS Envelopes [3]

utility of the airport but this in turn will also increase stress and workload on the air traffic controller. N. Yoichi et al [4] used a neural network approach to take on this problem. They have used the data from Tokyo International Airport (RJTT) for their work. Based on the direction of the wind, the RJTT airport uses two of its four available runways. To determine which runway to be assigned to the aircraft, the air traffic controller considers the direction of the aircraft's destination. But, sometimes due to the presence of traffic, non-nominal or non-optimal runways are also assigned.

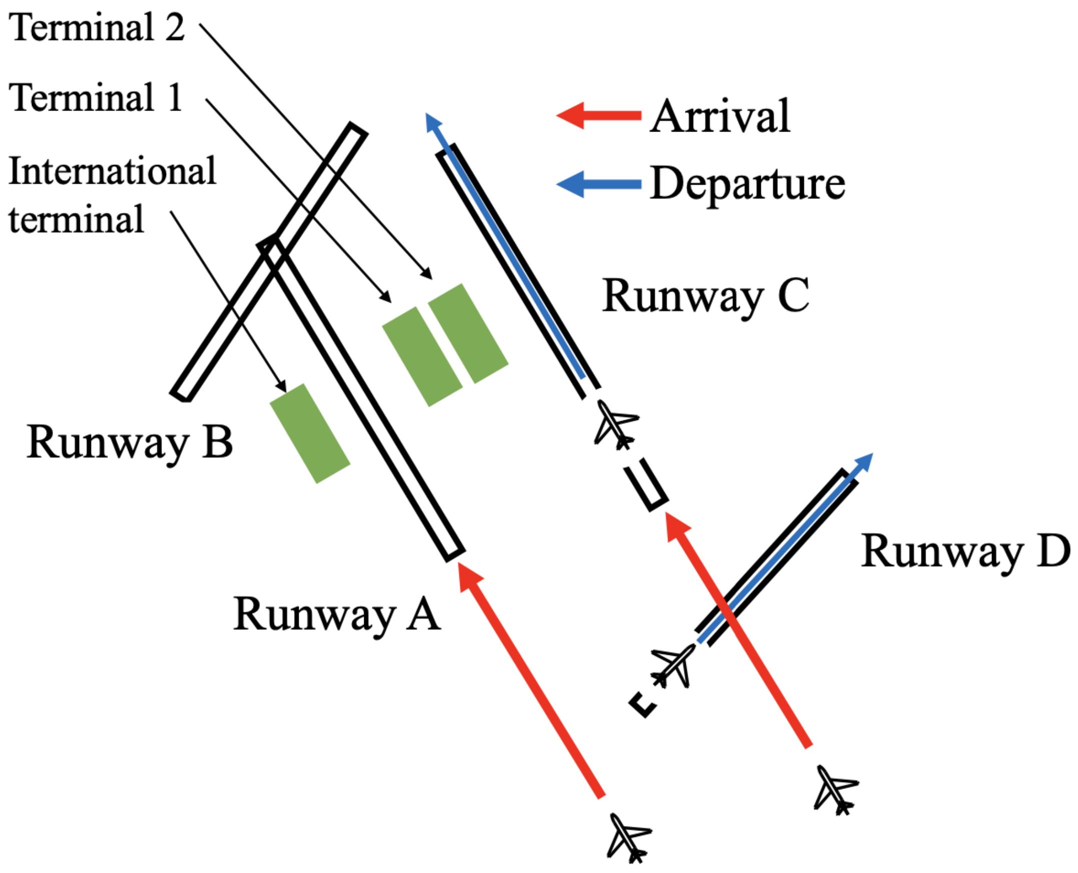


Figure 7: RJTT Airport Runways [4]

Runway changes are a practice among air traffic controllers to assign the runways more efficiently. One of the main reasons for runway swapping is traffic congestion. N. Yoichi et al [4] interviewed multiple air traffic controllers and according to them, the departure information is not taken into account because it is very difficult to estimate the traffic in 30 minutes. In their approach, they used a neural network-based model.

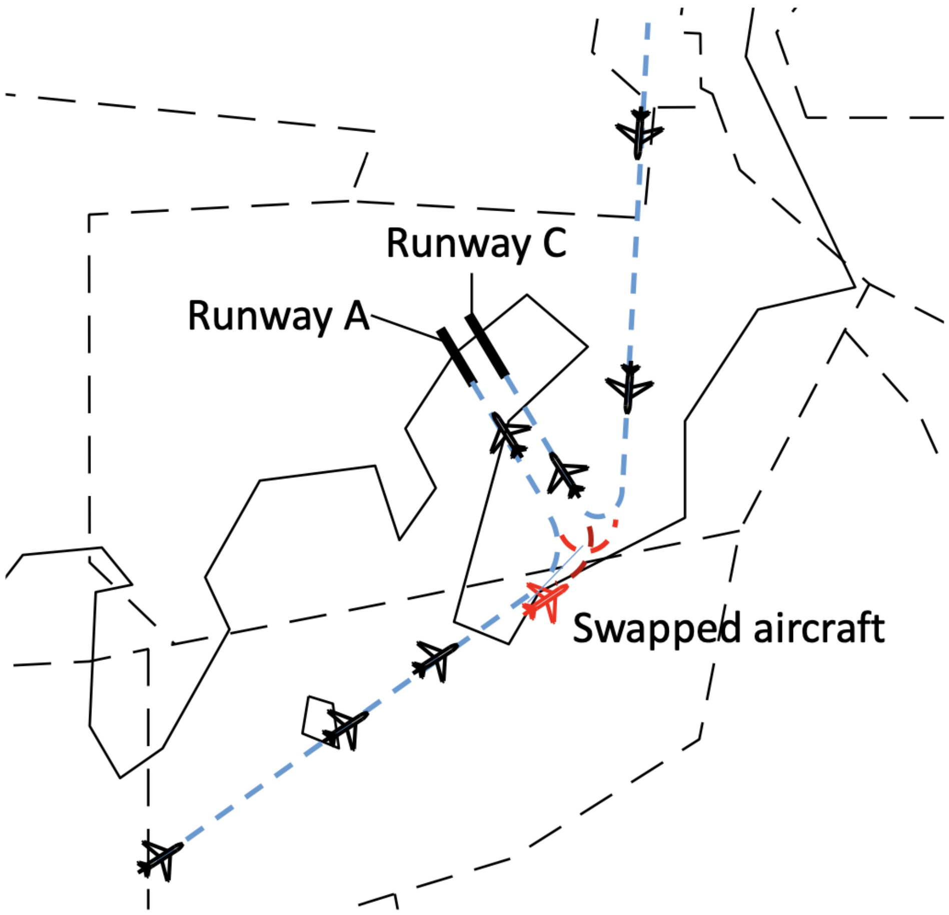


Figure 8: A Runway Swapping Scenario at the RJTT airport [4]

3.2 Multi-Layer Perceptron

In their approach, N. Yoichi et al [4] use a basic neural network model called as a multi-layered perceptron. In this, every unit receives an input designated as x. The output is denoted as z which is equal to the sum total of the weighted inputs and bias.

$$let\ u = \sum_{i=1}^{I}((W_i\ X_i)\ +\ b) \tag{4}$$

$$let\ z = \ f(u) \tag{5}$$

For the runway assignment problem, the entire data is classified either into swapped or non-swapped classes. From this, the posterior probability is derived as

$$y_k = p(d_k|x)\ = \exp(z_k)\ \sum_j \exp(z_j) \tag{6}$$

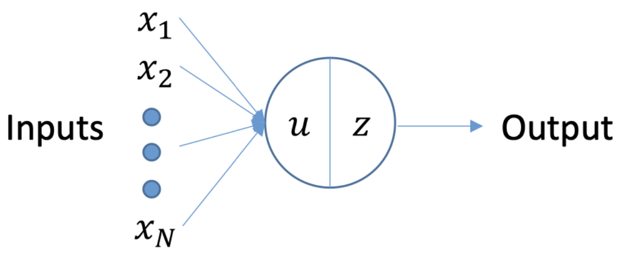


Figure 9: Inputs and Output of one unit [4]

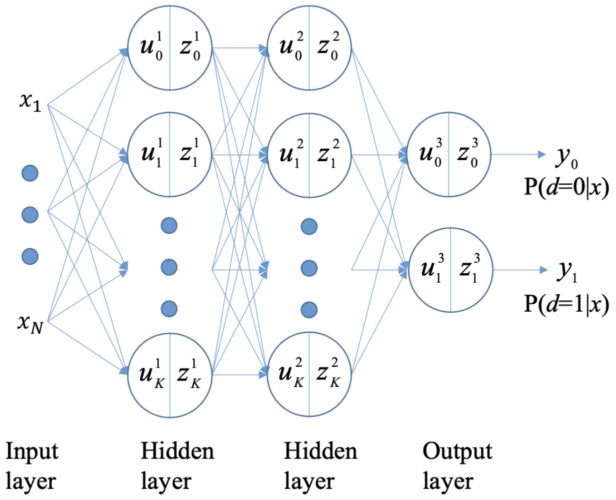


Figure 10: Multi Layered Perceptron [4]

The model divides the aircraft into two types: swapped denoted by 1 and non-swapped denoted by 0. There are four types of projected and actual assignment combinations that were predicted. they are: [0 0], [0 1], [1 0], and [1 1]. Both the [0 0] and [1 1] findings indicate that the model can accurately predict. When they define accuracy as the number of [0 0] and [1 1] divided by the total number of airplanes, the accuracy is expected to be very high in this scenario because the number of swapped aircraft is small when compared to the number of unswapped aircraft. They tried the same approach with four different types of input data and they found that terminal preference has an important role in runway assignment.

Ng, K.K.H, Lee, C.K.M [5] focused on the Aircraft Sequencing and Scheduling Problem (ASSP). ASSP is one of the most critical aspects of air transport management. This problem deals with Landing, takeoff, and aircraft movement in the airspace. The problem contains two main aspects which are Aircraft Landing Problem (ALP) and Aircraft Takeoff Problem (ATP). The main motive for the ASSP problem is that the airport ensures on-time travel or has a minor impact on airport operations. Initially, the ASSP used a First Come First Serve (FCFS) approach to decide which aircraft took off or landed.

The ALP is used to determine the optimal aircraft landing time and which runway the aircraft lands. They have a separation constraint that needs to be followed and this is essential to maintain airspace safety. While trying to land the aircraft, meeting the safety constraints, there is a change in actual landing time and it is stressful and hard for controllers to do this manually. They proposed a novel method called Modified Variables Neighborhood Search (MVNS). Their approach had a better exploration ability. The traditional VNS algorithm, though efficient may fall into the local optimum trap when performing a single swap of two variables.

To provide variation along the search space, the algorithm begins with a randomized solution. The number of unsuccessful updates by neighborhood structure operators is represented by the adaptive control parameter. A randomized recon-

```
Algorithm Architecture
Introduce maximum tolerance operator $limit$ as the maximum number of iteration $MaxIter$
                    $limit = MaxIter$
Define the stopping criteria as $iter$ reaching the $MaxIter$ or best known value $BKV$ is found
Set $iter = 0$
Set$trial = 0$
Call function: Initialization Phase
Do
    Call function: Modified Neighborhood Search Phase
    Call Function: Perturbation Phase
Until stopping criterion is met, $iter > MaxIter \,||\, f\,(best) \leq BKV$
```

Figure 11: Modified Variables Neighborhood Search (MVNS) Algorithm [5]

struction of a neighbor solution preserves solution diversity. The static model does not accurately reflect real operations, but it does provide useful insight into aircraft scheduling. Landing of the aircraft is not limited to time, but rather in space. As flight travel time increases, the main concern in aircraft landing is the reliable arrangement of the landing and maintaining a minimal turnaround time before the next departure.

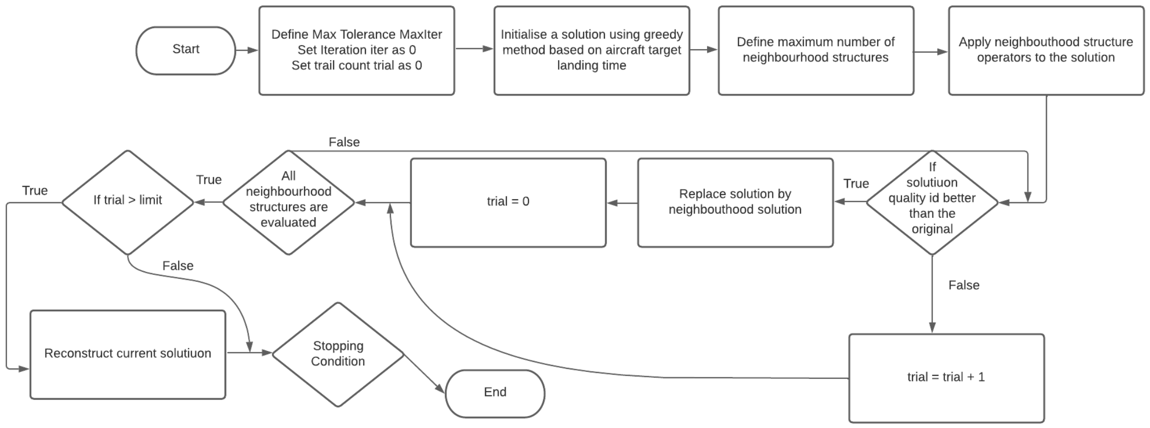

Start
Define Max Tolerance MaxIter Set Iteration iter as 0 Set trail count trial as 0
Initialise a solution using greedy method based on aircraft target landing time
Define maximum number of neighbourhood structures
Apply neighbouthood structure operators to the solution
If solutiuon quality id better than the original
True
Replace solution by neighbouthood solution
trial = 0
All neighbourhood structures are evaluated
False
True
If trial > limit
True
False
False
trial = trial + 1
Reconstruct current solutiuon
Stopping Condition
End

Figure 12: Flowchat for the MVNS Algorithm

The task of selecting how to land airplanes closing in on an airport entails allocating each aircraft to an optimal runway, calculating a sequence for landing for each runway, and arranging each aircraft's landing time. Each aircraft's runway allocation, sequencing, and scheduling must guarantee that the scheduled landing time falls within a set time window and that it meets separation time requirements with other aircraft. The goal is to maximize runway utilization. For this problem, Xiangwei et al [6] proposed a unique algorithm that is based on receding horizon control. This algorithm is called as sliding window algorithm.

In their work, Xiangwei et al [6] look at the ATC segment. The stressful task of allocating the runway to an aircraft and making sure that the aircraft lands safely and within an allocated time is dependent on the air traffic controller. This is stressful and challenging because the arriving aircraft flow is not uniform; it contains a variety of aircraft types. Wake vortices are created at the rear of all aircraft in flight. These vortices evolve chaotically and can produce severe turbulence to a closely following aircraft, even resulting in a collision. To maintain an aircraft's aerodynamic stability, the duration between one plane landing and the next plane landing must be longer than a predetermined minimum (the separation time), which varies depending on the planes involved. For solving this, they use their sliding window Algorithm.

They consider the aircraft landing problem as a multi-period decision problem and use the RHC strategy to counter this problem. RHC can be defined as a n-step ahead iterative online optimization strategy. Based on the information available, RHC optimizes the problem for the N intervals in the future but only implements the solution that is pertinent to the current interval. Based on the updated information, RHC works the same way for the next N intervals in the future. They used the RHC to determine the scheduling time and runways step by step. They found that the execution time for this RHC algorithm can't be predicted because the number of aircrafts during a congested time and a non-congested time varies. To bypass this, in their algorithm, they used a fixed size of aircrafts.

3.3 Sliding Window Algorithm

Sequence aircraft waiting to land in FCFS order in terms of their target times. Denote the ordered set of aircraft as $F\{i|i = 1, ...N\}$. A widow of aircraft W is defined, along with an incremental step size S, both in terms of the number of aircrafts. Here, r is step count, $F(r)$ shows the aircrafts that have not been swept by sliding window, $F_o(r)$ symbolises aircrafts swept by sliding window. $Q(n|r)$ record the nth aircraft in the optimized arrival sequence for $F_0(r)$, i.e., $Q(n) = i$ means that aircraft i in $F_o(r)$ turns out to be the nth aircraft in the optimized arrival sequence $F_1(r)$ represents the set of aircraft that have been swept by the sliding window and have their landing times fixed at step r.

$$\alpha_i = \text{how soon plane } i, i \in \{1 \cdots P\} \text{ lands before } T_i$$

$$\beta_i = \text{how soon plane } i, i \in \{1 \cdots P\} \text{ lands after } T_i$$

$$\delta_{ij} = \begin{cases} 1 \text{ if plane } i \text{ lands before plane } j,\ (i,j) \in (1, \cdots, P)^2, i \neq j \\ 0 \text{ otherwise.} \end{cases}$$

$$y_{ir} = \begin{cases} 1 \text{ if aircraft } i \text{ lands on runway } r\ ,\ i \in \{1 \cdots P\}, \\ \quad r \in \{1 \cdots R\} \\ 0 \text{ otherwise.} \end{cases}$$

$$z_{ij} = \begin{cases} 1 \text{ if aircraft } i \text{ and } j \text{ lands on the same runway}\ , \\ \quad (i,j) \in \{1 \cdots P\}^2, i \neq j \\ 0 \text{ otherwise.} \end{cases}$$

Figure 13: Decision Variables in SLiding Window Algorithm [6]

For the simulation, they used data from 500 aircrafts and 4 runways and compared their results with previous results. In terms of execution durations and solution values, the sliding window approach surpasses the scatter search algorithm and the bionomic algorithm given in earlier research.

All these research works help understand the aircraft landing problem more efficiently and prompted more research into reinforcement learning to solve the airport assignment problem to ease the stress of the air traffic control and to direct aircrafts to the airport as soon as possible and also make sure the safety of the airspace and other aircrafts are not put into risk. All these researches explore various approaches to solving different aspects of the airport and aircraft assignment problem. More extensive research and the use of reinforcement learning by which we can train the agent of the airspace under a particular airport can help us with aircraft assignments with more efficiency.

CHAPTER 4

Problem Formalisation

4.1 Problem Definition

An emergency aircraft needs immediate and priority attention from the ATC. any delays from the ATC may have a direct impact on the safety of the aircraft. It becomes very strenuous for the ATC as it has to manage the airspace and give priority to the emergency aircraft. As the nature and the position of the emergency is random, we can use reinforcement learning to train an agent that will guide the emergency aircraft.

In this project, we aim to deploy a reinforcement learning agent for controlling emergency aircraft in a dynamic environment. The primary objective is to navigate the emergency aircraft to the airport in the shortest amount of time. I aim to train an agent that can navigate the emergency aircraft to the airport in the least time.

4.2 Variables

4.2.1 Distance

let D be the distance between the initial point (I_e) of the emergency aircraft and the airport (A_c). Where,

$$I_e = (X_e, Y_e), \tag{7}$$

and

$$A_c = (X_a, Y_a), \tag{8}$$

So, the Distance between the initial point of the aircraft and the airport is

$$D = \sqrt{(X_a - X_e)^2 + (Y_a - Y_e)^2} \tag{9}$$

The distance between the position of the aircraft and the airport changes as the aircraft moves.

4.2.2 Position

In the real world, the position of the aircraft usually comprises three factors that are extracted using Global Position System (GPS). These factors are latitude, longitude, and altitude above sea level. All this information is relayed to the ATC by the aircraft and this helps the ATC to safely navigate an aircraft safely.

$$Position_{aircraft} = (latitude, longitude, altitude) \tag{10}$$

In this project, a few assumptions are made in establishing the position of the aircraft. The entire environment is a 2-Dimensional grid and altitude is not considered for this project. This representation is simplified compared to the real world and it will allow the agent to learn and navigate in the environment more easily.

4.2.3 Heading

The direction in which the aircraft is looking at is called the heading. In the real world, the heading is calculated by using the inertial navigation system (INS) and the aircraft's compass. The INS calculates the rotation of the earth and the acceleration by using the aircraft's equipment which gives the current position of the aircraft. Now, using the compass we get the magnetic heading which is the direction the aircraft is currently looking at with respect to the magnetic north. Using these two factors, we calculate the true heading of the aircraft.

$$H = \text{atan2}\big(\sin(\Delta\lambda) \cdot \cos(\phi_2), \cos(\phi_1) \cdot \sin(\phi_2) - \sin(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\Delta\lambda)\big) \tag{11}$$

here, H is the heading of the aircraft. ϕ_1 and ϕ_2 are latitudes of the initial and final positions. $\Delta\lambda$ is the difference between the longitudes of the initial and final points.

In this project, I am calculating the heading using the positions of the aircraft and the airport. The heading of the aircraft is a unit vector that represents the direction in which the aircaft is facing.

$$H = \frac{(aircraft\_position - airport\_position)}{||aircraft\_position - agent\_position||} \tag{12}$$

here , $||aircraft\_position - agent\_position||$ is the Euclidean Norm. The Euclidean Norm is a way to measure the magnitude of a vector in Euclidean Space.

CHAPTER 5

Approach

5.1 Emergency Aircraft Assignment

5.1.1 Airspace

Every country and every airport within has a designated section of the atmosphere above it that is under the control and supervision of the country's government. This designated section is called airspace. The main use of the airspace is so that the authorities can monitor all the aircrafts that are flying to ensure safe aircraft mobility. Airspace is divided into many classes and each of these classes has different rules. The designations to the airspace are done keeping in mind various aspects like air density, air volume, the traffic in that area, and if the airspace is purely used for military or civilian purposes [7].

The airspace is primarily classified into two distinct categories, controlled and uncontrolled. In uncontrolled airspace, the pilots are in charge of handling their own movement and safety in the airspace whereas in controlled airspace, Air Traffic Control (ATC) services are available and all pilots must have or obtain clearance for movement in that airspace. Along with being divided into classes, the airspace is also divided into six altitude zones or levels.

5.1.2 Aircraft Emergency

Managing the airspace effectively is an important aspect to make sure flight delays are minimized and passenger safety is ensured. The task of managing the airspace in a controlled environment is done by the ATC. There are many problems or challenges that the ATC faces when handling airspace. One of the main issues is air traffic. In a high air traffic scenario, managing the airspace becomes increasingly difficult for the ATC, and more air traffic leads to airspace congestion and delays. One other aspect that increases the strain and challenge for the ATC is an emergency aircraft. An emergency aircraft is an aircraft that is facing an emergency and may

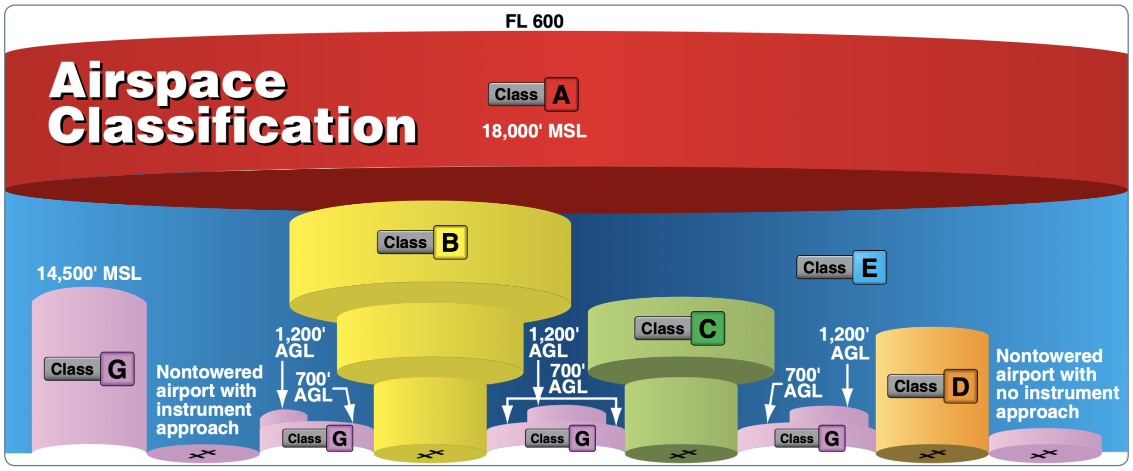


Figure 14: The Airspace Classification [7]

require immediate help and priority service from the ATC. The need for immediate service and priority increases the complexity involved in handling the airspace for the ATC.

These emergencies that occur in the aircraft are random and unpredictable. These situations can arise at any time and suddenly which requires the ATC to be swift and efficient in their response. In a case where they have multiple emergencies to deal with, handling the airspace becomes even more challenging. The ATC has to consider various aspects like weather and airport situations while handling the emergency which increases a significant amount of stress on the ATC.

5.1.3 Reinforcement Learning in Emergency Aircraft Assignment

The use of reinforcement learning in this scenario can help improve the efficiency and safety pertinent to the ATC system. The ATC system agent can learn from experience which could help in making better decisions without delay. For instance, the ATC agent can be used to improve the emergency aircraft route assignment to the airport and make sure that the aircraft reaches the airport quickly and safely.

In airspace, there are multiple aspects to consider before building a model. The first thing to consider is the emergency aircraft itself. The emergency aircraft has a current heading and an initial position. This initial position and initial heading are generated randomly in the environment. This is done to ensure that the agent does not overfit but just learns one particular route to the airport. The position of the airport is also generated at random after every episode thus giving the agent a chance to explore the entire state space and associate actions and states with outcomes.

5.2 State Space

The State space in reinforcement learning refers to the environment or space that the agent can see. It refers to the set of all possibilities of states the agents could encounter when interacting in the environment. The State Space usually conveys all the information that the agent needs to make a decision of which action to pick to reach the intended target.

The state Space can be either continuous or discrete. If the number of combinations possible is finite then the state space is discrete and in a case where the combinations are infinite, the state space is continuous.

For the air traffic environment, the state space can be defined as a dictionary containing three main components.

In a 500x500 Grid,

1. Airport position - a two-dimensional circle that represents the position of the airport.

2. Aircraft Position - a two-dimensional point that represents the position of the aircraft

3. Aircraft heading - a unit vector representing the direction the aircraft is facing.

4. Obstacle Position - a two-dimensional circle that represents the position of the

obstacle.

The airport, aircraft, and obstacle position can take values ranging between 0 and 500 and the heading is a unit vector.

5.3 Action Space

The action space in reinforcement learning refers to the collection of all feasible actions that an agent can do in a given environment. The agent's policy and the present state of the environment influence the action taken at each phase. Depending on the sort of problem being solved, the action space can be continuous or discrete. The set of available actions in a discrete action space is finite and discrete, such as choosing from a set of predefined activities. A continuous action space, on the other hand, entails picking a value from a continuous range of values, such as the amount of force to be applied in a robotic arm. The action space's size and complexity can have a major impact.

in the air-traffic scenario, the action space is

$$A_t = \{0, 1, 2\} \tag{13}$$

here,

$$A_t = \begin{cases} 0 & \text{for} \quad Aircraft\ turns\ left\ by\ 90 \\ 1 & \text{for} \quad Aircraft\ is\ straight \\ 2 & \text{for} \quad Aircraft\ turns\ right\ by\ 90 \end{cases}$$

5.4 Reward Function

A reward function is an important notion in reinforcement learning, a sort of machine learning in which an agent is trained to make decisions based on the results of its actions. The reward function sets the agent's target and gives feedback in the form of a numerical value or score to the agent. The agent's task is to achieve the greatest possible score over time by choosing actions that end up resulting in high rewards. The reward function can be programmed to reward desired conduct while discouraging undesired behavior. Designing an effective reward function, on the other hand, might be a difficult process because the agent may learn to exploit gaps in the function rather than behave optimally. The design of the incentive must be carefully considered.

In the air traffic scenario, we follow certain rules to decide the rewards. they are :

1. The Aircraft is not allowed to fly out of the sector.
2. The aircraft should not collide with obstacles
3. The aircraft should land at the airport as fast as possible.

$$R_t = \begin{cases} 100 & \text{if} \quad Airport\ landed\ at\ the\ airport \\ -100 & \text{if} \quad Aircraft\ flies\ out\ of\ sector \\ -100 & \text{if} \quad Aircraft\ collides\ with\ obstacle \\ -100 & \text{if} \quad Aircraft\ going\ out\ of\ step \\ -0.1 & \text{for} \quad Each\ timestep\ the\ aircraft\ is\ not\ landed\ at\ the\ airport. \end{cases}$$

5.5 Q-Learning

Q-learning is an off-policy Temporal Difference control reinforcement learning algorithm. In the algorithm, the policy will be modeled as an $action - value$ function.

$$Q(s, a)\ be\ the\ action\ value\ function \tag{14}$$

The action-value functions gives us the information about the willingness of the agent to take an action a given a state s [9]. The Q-learning algorithm is defined as:

$$Q(S_t, A_t) < -Q(S_t, A_t) + \alpha[R_t + 1 + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{15}$$

$$\overset{\text{Update Current Q-value}}{\boxed{Q(s_t,a_t)}} \leftarrow \underset{\text{Current Q-value}}{\boxed{Q(s_t,a_t)}} + \overset{\text{Learning Rate}}{\boxed{\alpha}} \Big(\underset{\text{Rewards}}{\boxed{R_{t+1}}} + \underset{\text{Discount factor}}{\boxed{\gamma}} \overset{\text{Estimated reward from next action}}{\boxed{\max_{a_{t+1}} Q(s_{t+1},a_{t+1})}} - \boxed{Q(s_t,a_t)} \Big)$$

Figure 15: Q-Learning Equation [8]

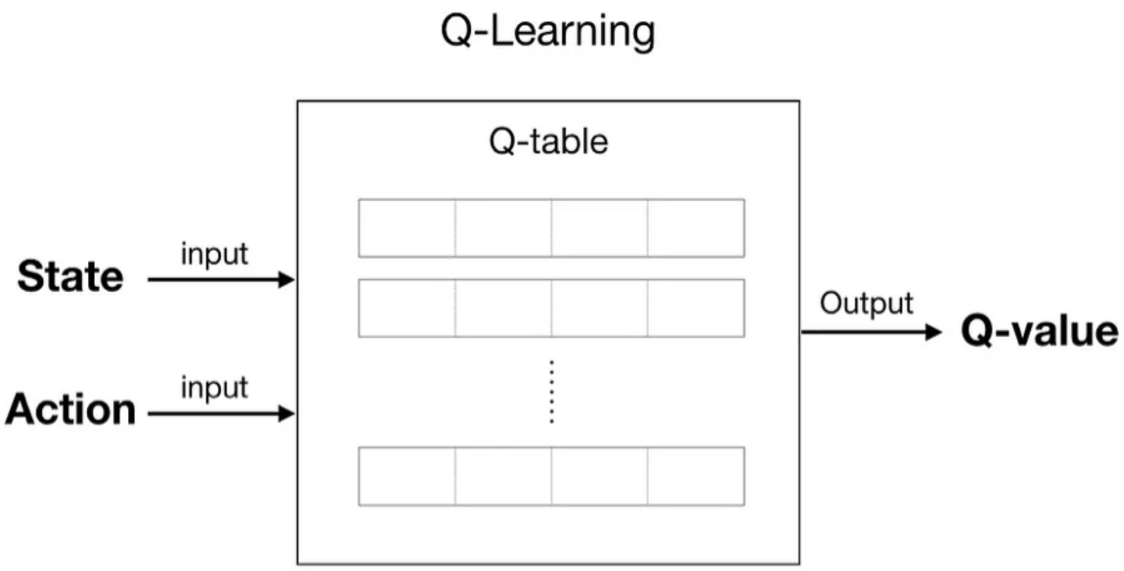
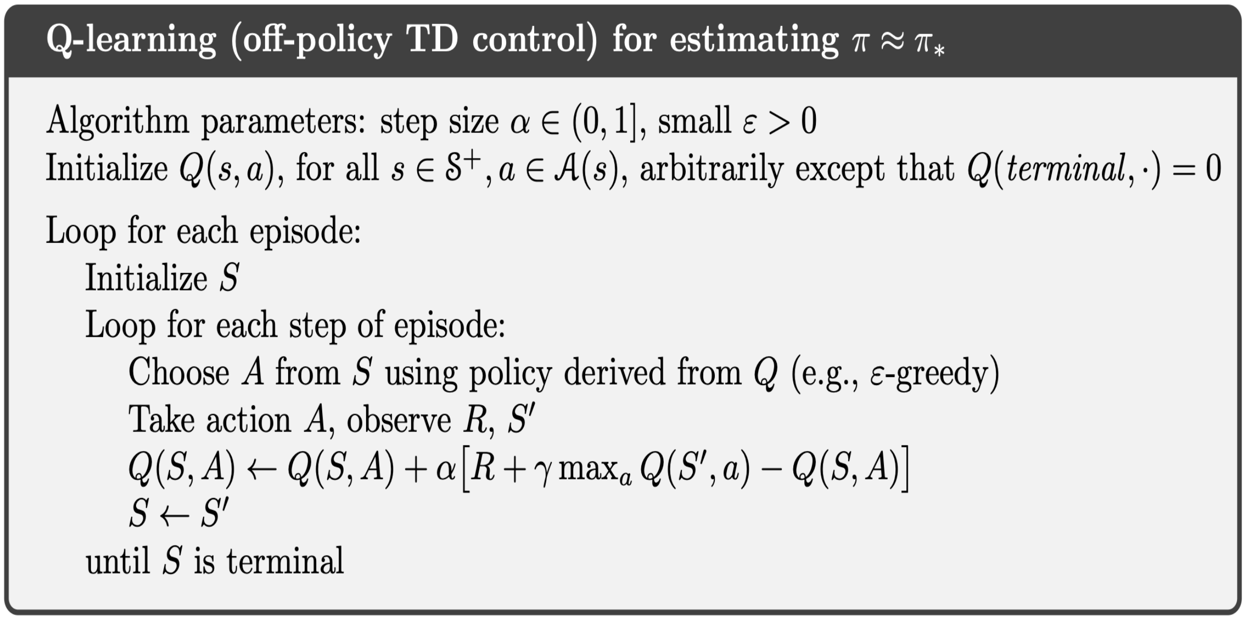


Figure 16: Q-Learning [9]

5.6 Deep Q Learning

Recent advances in computer vision and speech recognition have depended on training deep neural networks effectively on very large training sets. The most successful approaches use updates that employ stochastic gradient descent to train directly from raw inputs. It is often feasible to learn more accurate representations than manually created features by feeding sufficient information to a deep neural network [12] Using this approach, V. Mnih et al [12] came up with an approach that connected a reinforcement Learning Algorithm to a deep neural network. To



```
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Figure 17: Q-Learning Algorithm [8]

begin the testing approach, they considered Tesauro's TF-Gammon architecture. The parameters that are used to estimate a value function are updated using this architecture.

To contrast the architectural approach of the TD-Gammon, V. Mnih et al [12] used a method called *experience replay*

5.6.1 Experience Replay

Let $e_1, e_2, ..., e_n$ be episodes, where e_t is an episode at a time-step t. In the RL environment, a state, action, and reward at time-step t are denoted as S_t, a_t, and r_t, respectively.

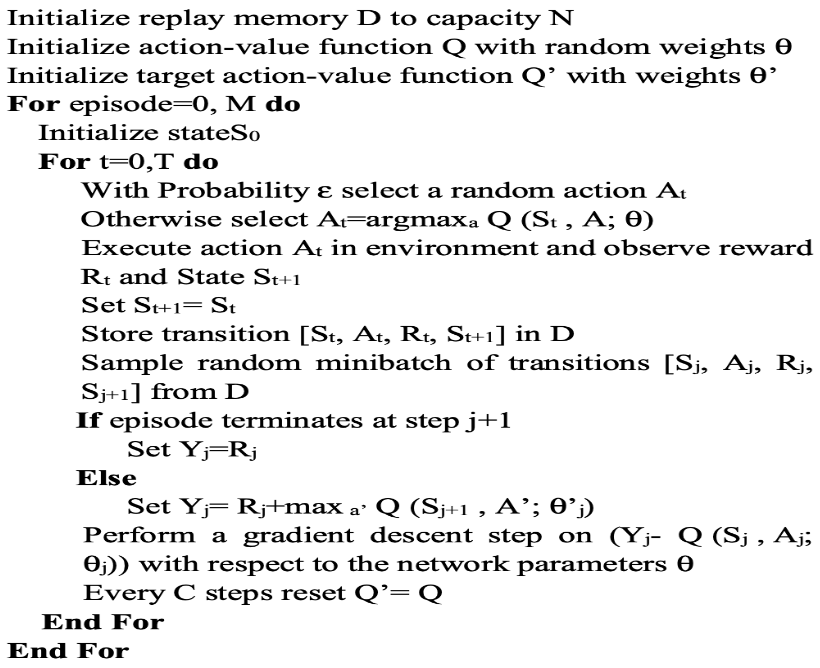
In the experience replay, we store

$$e_t = (s_t, a_t, r_t, s_{t+1}). \tag{16}$$

Hence, a data-set D is defined as

$$D := \{e_1, ..., e_n\}. \tag{17}$$

This information is pooled and stored in a *replay buffer*. After the experience replay step, we apply the Q learning updates and the agent will select actions following a ϵ greedy policy. This complete procedure is called the Deep Q Learning Algorithm.



```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function Q' with weights θ'
For episode=0, M do
   Initialize state$S_0$
   For t=0,T do
       With Probability ε select a random action $A_t$
       Otherwise select $A_t$=argmax$_a$ Q ($S_t$ , A; θ)
       Execute action $A_t$ in environment and observe reward
       $R_t$ and State $S_{t+1}$
       Set $S_{t+1}$= $S_t$
       Store transition [$S_t$, $A_t$, $R_t$, $S_{t+1}$] in D
       Sample random minibatch of transitions [$S_j$, $A_j$, $R_j$,
       $S_{j+1}$] from D
       If episode terminates at step j+1
            Set $Y_j$=$R_j$
       Else
            Set $Y_j$= $R_j$+max $_{a'}$ Q ($S_{j+1}$ , A'; θ'$_j$)
       Perform a gradient descent step on ($Y_j$- Q ($S_j$ , $A_j$;
       θ$_j$)) with respect to the network parameters θ
       Every C steps reset Q'= Q
   End For
End For
```

Figure 18: Deep Q-Learning Algorithm [10]

5.7 Approach

The agent in the environment iterates over the number of episodes to select actions and based on the reward it updates the Q-Values of the state action pair. According to its current policy, the agent observes the states and takes action. After the action is done, the agent moves to a new state and a reward is received. based on this, the agent will update the Q-value of the current state-action pair.

Initially, at the beginning of each episode, the positions of the aircraft and the airport along with the aircraft's heading are generated at random in the environment.

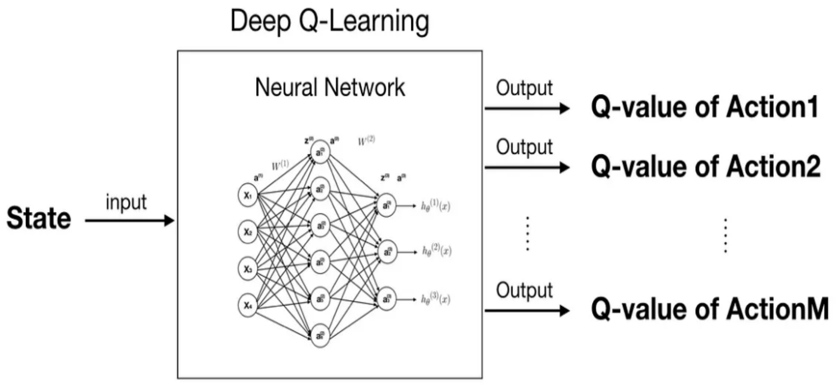


Figure 19: Deep Q-Learning Architecture [9]

The agent then calculates the optimal heading based on its current position and the position of the airport and takes action to change the heading towards the airport. At the end of every step, the agent's heading is again randomized so that the agent is not stuck in a sub-optimal policy.

For this project, the altitude of the aircraft is not considered and the speed is always constant.

Algorithm 1 Airport Assignment

```
for Number of Episodes do
    for Step Count != step limit do
        Increment the step count by 1
        Initiate the position of the aircraft randomly
        Initiate the position of the airport randomly
        Initiate the position of the obstacles randomly
        Using the position of the aircraft and the airport, calculate the relative heading between them.
        Based on the relative heading, take action and update the heading of the aircraft
        if the next state = Airport then
            reward = 100
            done = true
            End Episode
        if the next state = out of bound then
            reward = -100
            End Episode
        if the next state = obstacle then
            reward = -100
            End Episode
        reward = -0.1 for event step not at an end state.
        Reset the heading randomly
    if Step Count = Step limit then
        reward = -100
        Done = true
        End Episode
```

5.8 Exploration Method

Exploration means reaching out to states that have not been explored yet so that the environment is learned more. Deep Q-Learning has many learning exploration strategies that can be used to tell the agent how to explore.

5.8.1 Epsilon-Greedy Policy

In reinforcement learning, the Epsilon Greedy Policy is a policy that is commonly used to employ an exploration approach. This policy's fundamental tenet is to choose the action that maximizes the Q-value with probability 1-epsilon and a random action with probability epsilon. The ratio of exploration to exploitation in the policy is based on the value of epsilon. The policy gets more predictable and the agent depends more on its learned Q-values as epsilon goes down. On the other hand, it gets more exploratory as the epsilon rises, and the agent is more prone to choose random behaviors. This strategy may be useful when we want to balance Exploitation and Exploration and we want a simple implementation. [13]

5.8.2 Boltzmann Q Policy

It is also an exploration-based approach that works on the Boltzmann Distribution. In this policy, the action probabilities are proportional to the exponential of the Q-values divided by a temperature parameter. The temperature or Tau handles the degree of exploration. As the tau value is close to zero, the agent relies more on the learned Q values and vice versa. Boltzmann Policy naturally encourages exploration. This policy can be used when we want to choose actions problematically. In situations where we need to shape rewards, have a continuous actions space. or we need to tune the exploration sensitivity, this strategy is useful.[13]

5.8.3 Random Exploration

In this simple exploration strategy, the agent chooses its course of action entirely at random and without taking the Q-values into account. The agent randomly explores the state-action space without regard for the perceived quality of the acts. This strategy is very useful during initial exploration, situations where we have stochastic environments, and sparse reward environments.

5.8.4 Upper Confidence Bound (UCB)

This strategy is commonly used in Reinforcement learning and Bandit Problems. It considers both the average reward and the uncertainty associated with it. The fundamental tenet of UCB is choosing behaviors with high expected rewards while considering their exploration potential. Using the reward and the exploration count, the strategy maintains an upper confidence bound associated with each activity. The action with the highest UCB is used throughout the process. This strategy is very useful in situations where the environment dynamics are unknown, have continuous action space, or in bandit problems.[13]

5.8.5 Thompson Sampling

In this strategy, exploration and exploitation are balanced by maintaining a distribution over the action values using a Bayesian technique. Thompson Sampling selects the action with the highest sampled value by randomly selecting an action from the distribution. The fundamental principle is to give activities that are more likely to result in larger rewards a higher probability based on the observed data. Thompson Sampling uses Bayesian inference to update the action value distribution as it learns. The action values are first modeled using prior distributions, and the posterior distributions are updated when the agent engages with the environment and gathers rewards. This method is very useful where the reward distribution is uncertain, the environment is not stationary or have a small-medium action space. [13]

For the context of this problem, I have employed two very common strategies in Reinforcement learning. They are :

- Epsilon Greedy Policy
- Boltzmann Q Policy

CHAPTER 6

Environment and Simulation

For testing out the algorithm, we need an environment with all out requirements i.e the non emergency aircraft, the emergency aircraft, the airport and the airspace. For this purpose, I used open AI GYM and Python to design the environment.

6.1 OpenAI - Gym

OpenAI Gym is a toolkit that can be used in developing and later testing various reinforcement algorithms. This has a certain set of testing problems or environments that we can use to test our reinforcement learning agent. The main purpose of gym is the enablement of rapid development and experimentation pertinent to reinforcement learning algorithms. Some examples of famous open AI Gym environments are atari games and robotics games [14].

One of OpenAI Gym's primary characteristics is its standardized interface, which enables easy integration with multiple reinforcement learning methods. This interface provides a standard set of methods for interacting with the environment, such as resetting the environment, performing an action, and monitoring the subsequent state and reward. This common interface allows researchers and developers to concentrate on designing unique reinforcement learning algorithms rather than worrying about environment peculiarities.

OpenAI Gym contains tools for displaying and monitoring the performance of reinforcement learning algorithms in addition to a standardized interface. These tools can be used to track an algorithm's development during training and to compare the performance of various algorithms in the same environment. This enables identifying areas for improvement and iterating on the design of reinforcement learning algorithms simple. Overall, OpenAI Gym is a valuable resource for everyone interested in creating and comparing reinforcement learning algorithms, from researchers and academics to hobbyists and enthusiasts. One other important aspect of openAI gym is that we can also creatre custom environments using it.

We must define a class that implements the Gym environment interface in order to create a custom environment. This entails creating a collection of methods that allow an agent to interact with the environment, such as a way for resetting the environment, a method for performing an action, and a method for retrieving the current state and reward. Developers can utilize the environment class to test and evaluate various reinforcement learning algorithms once it has been defined. OpenAI Gym provides visualization tools for these algorithms' performance, allowing developers to iterate and tweak their solutions as needed.

6.2 Airspace Environment

I created a custom environment using OpenAI's Gym. For this, we first import the required packages

```
import gym
from gym import spaces
```

After importing, we create a *Airport Environment* Class. In this class, we use the __*init*__ method. The __init__ method in the Gym environment is a particular method used to initialize the environment when an instance of the environment is created. Specifically, __init__ configures the environment's initial state and any parameters or configurations required for the environment to work effectively. It may, for example, initialize variables that track the current state of the environment, specify the action and observation spaces, or load any required data or models. We declare the initial size of the environment grid, the observation spce, the action space and any other information that we need when the environment is generated. The other functions that are in the environment are

6.3 reset ()

The *reset*() method is used in a reinforcement learning environment to reset the environment to the initial state and return the initial observations. During an episode, the agent interacts with the environment and receives feedback. As soon as the episode ends, the *reset*() method is called to reset the environment and prepare for the next episode. The *reset*() method's main responsibility is to return the initial environment observations to the reinforcement agent so that it can start its exploration and environment observation anew [14]. We are resetting the agent's position and heading to the initial state which in this case is random at the start of any episode. Another important method in the environment class is the *step*() method.

6.4 step()

The *step*() method contains the entire business logic of the senario. When an episode is running, the agent takes a predefined number of steps where it performs various *actions* and receives *feedback* from the environment. One the feedback is received, the step function returns the following tuple [14]

```
(observation, reward, done, info).
```

1. Observation - a new observation that we get from the environment after an action is taken.

2. Reward - This is the reward the agent receives from the environment for an action it took from its current state.

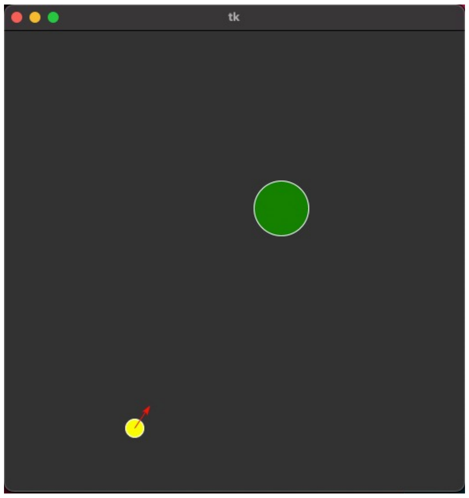
3. Done - This is a boolean value that tells us if the episode is terminated when the agent in the current state.

4. info - This has additional information and can be used when debugging.

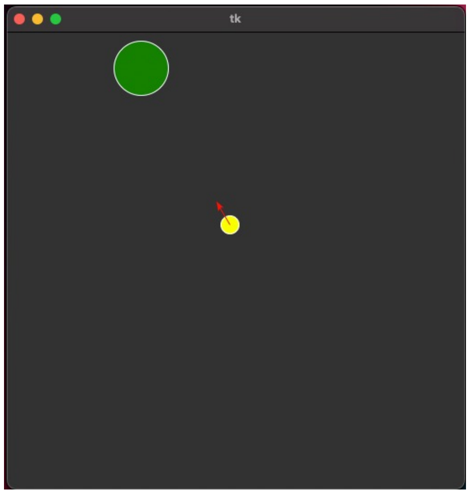
Another important method in the environment class is *render*()

6.5 render ()

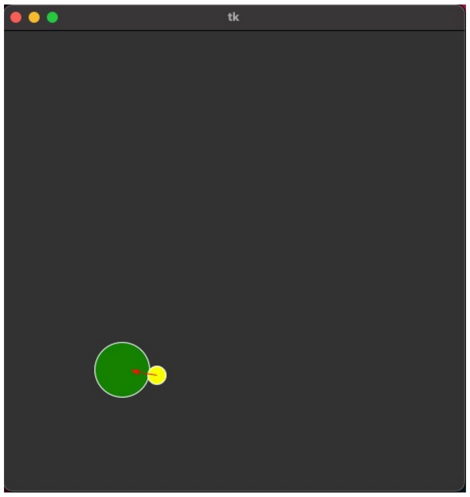
This method is used to visualize the current state and the environment. This is very useful when you want to see what's happening in the algorithm and can be very helpful when debugging. The exact behavior of *render*() changes as the environments change. There are various environments that have multiple *render*() methods that can be used to display various visualizations



(a)

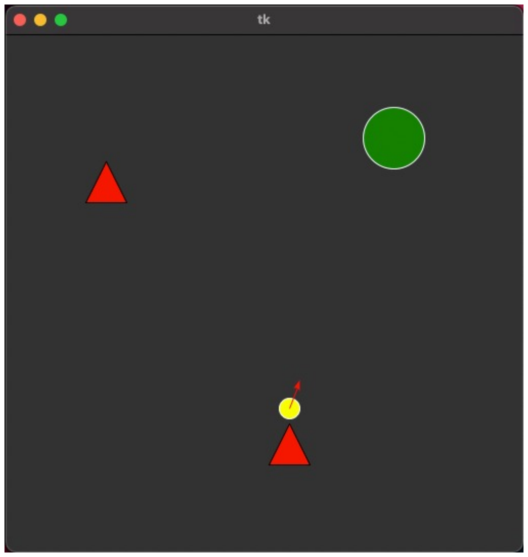


(b)

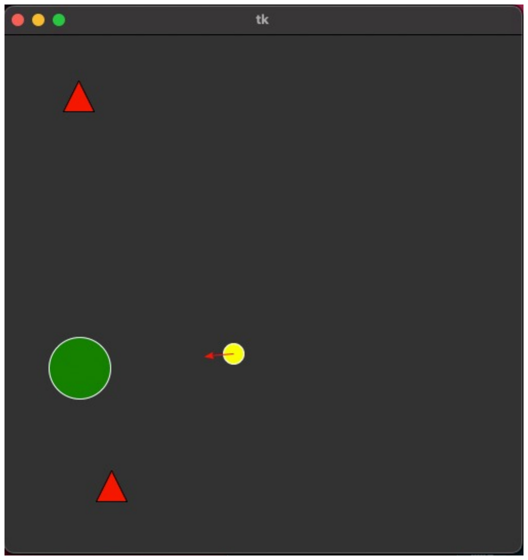


(c)

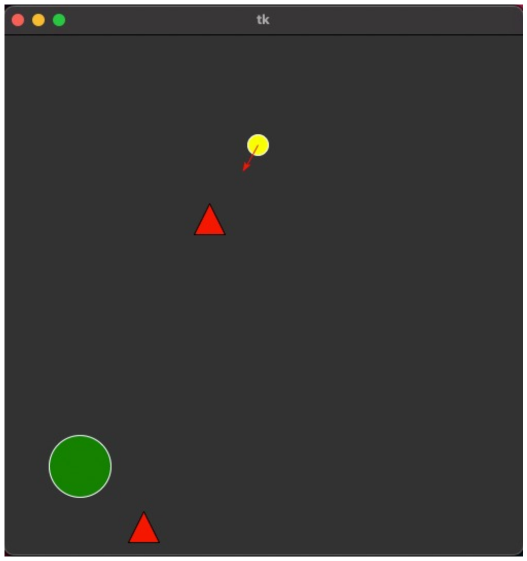
Figure 20: Aircraft and Airport



(a)



(b)



(c)

Figure 21: Aircraft, Airport and Obstacles

For the current problem, the environment is a

1. 500 X 500 grid
2. Has an airport and the airport region (The area the aircraft reaches to successfully land the aircraft) is designated in the color green.

3. Has 1 emergency aircraft designated in a yellow circle whose job is to find the path to the airport.

4. Based on the running scenario, has 2 obstacles designated in red triangles.

Figure 20(a) represents a scenario showing the airport and the aircraft at a random position. Figure 20(b) represents a scenario where the airport and aircraft are at different positions in different episodes. Figure 20(c) shows a case where the aircraft has reached the airport. Figures 21(a) and 21(b) show a case We have the airport, the aircraft, and 2 obstacles in different positions. Figure 21(c) shows a case where the aircraft is heading towards an obstacle.

CHAPTER 7

Results

7.1 Evaluation Metircs

We evaluate the RL agents based on two metrics: the number of successful landings at the airport (*Success*) and the mean episode reward, which can be translated into the time duration from the emergency incident and the landing.

7.2 Hyper-parameter Tuning

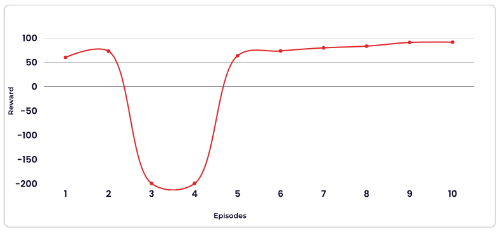
Hyperparameters have a direct effect on how the model will perform. Tuning these hyperparameters cannot be learned using existing data. Hyperparameters can be tuned by systematically searching from a range of different hyperparameters.

7.2.1 Emergency Aircraft and Airport Senario

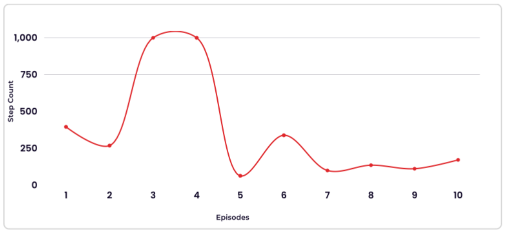
7.2.1.1 Epsilon Greedy Policy

For the Epsilon Greedy Policy, we can change the value of Epsilon to see how the performance of the model is varying. For this scenario, I have considered the following Epsilon values.

$$\epsilon = \{0.1, 0.3, 0.7, 0.9\} \tag{18}$$

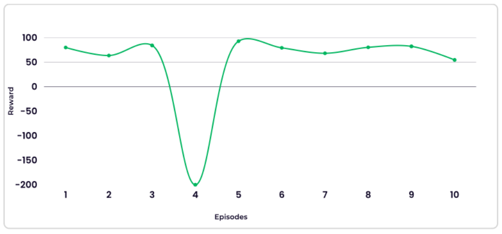


(a) Episodes vs Reward

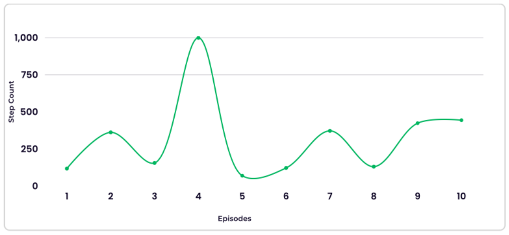


(b) Episodes vs Step Count

Figure 22: Epsilon - 0.1 for 50 Episodes

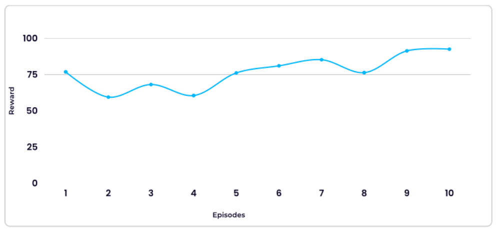


(a) Episodes vs Reward

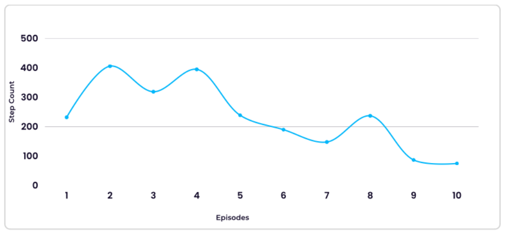


(b) Episodes vs Step Count

Figure 23: Epsilon - 0.3 for 50 Episodes

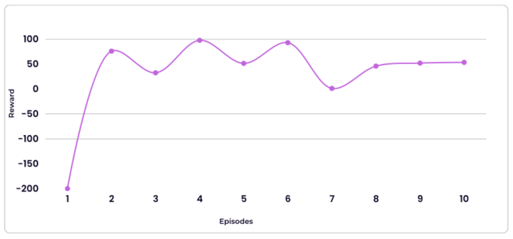


(a) Episodes vs Reward

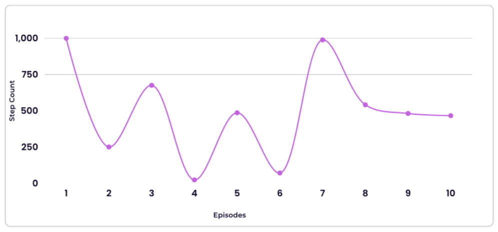


(b) Episodes vs Step Count

Figure 24: Epsilon - 0.7 for 50 Episodes



(a) Episodes vs Reward



(b) Episodes vs Step Count

Figure 25: Epsilon - 0.9 for 50 Episodes

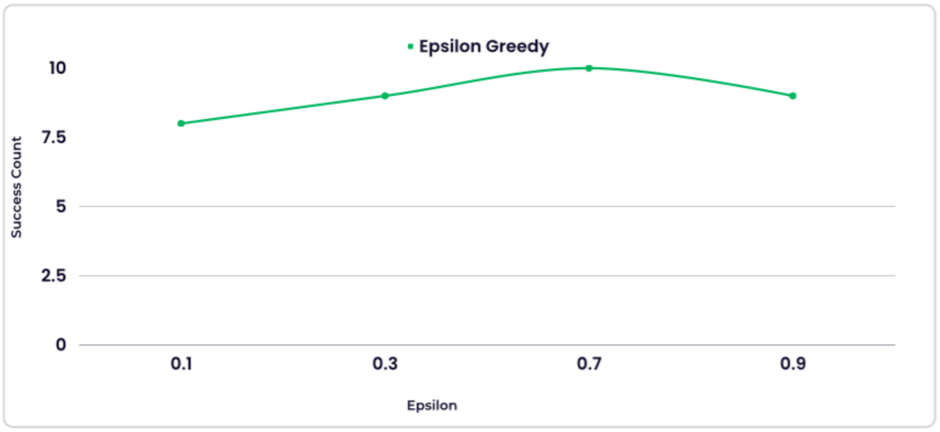
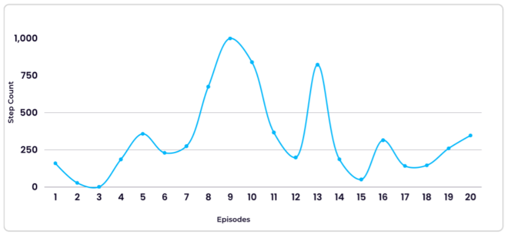
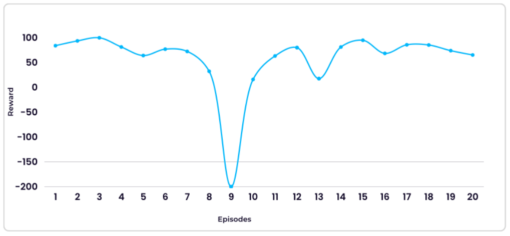


Figure 26: Airport Reached vs Epsilon for 50 Episodes

From the following graph, it can be inferred that the epsilon value that gave the best result was 0.7. The success rate increased from 0.1 to 0.7 and started decreasing after. Based on this, further experimentation was done on more episodes for Epsilon 0.7

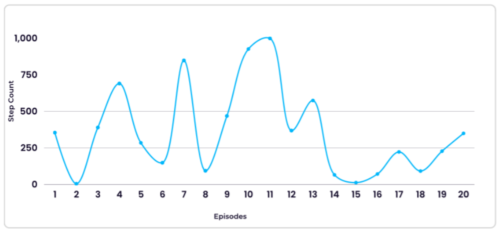


(a) Episode vs Step-Count for 100 Episodes

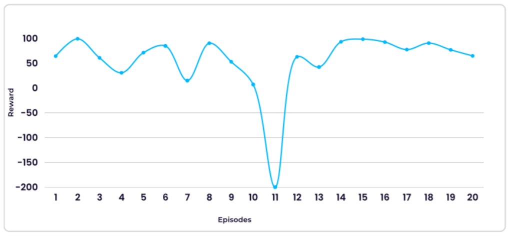


(b) Episode vs Reward for 100 Episodes

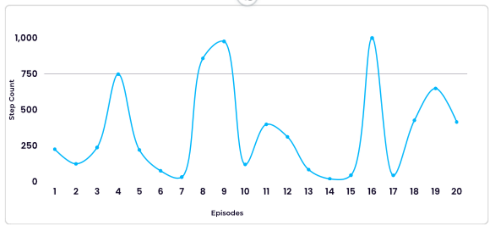
Figure 27: Epsilon - 0.7 for 100 Episodes



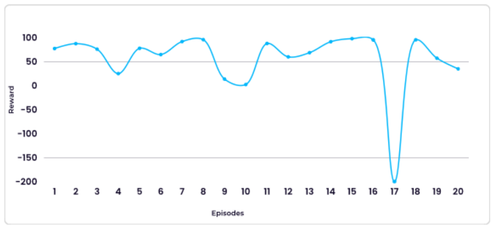
(a) Episode vs Step-Count for 500 Episodes



(b) Episode vs Reward for 500 Episodes



(c) Episode vs Step-Count for 1000 Episodes



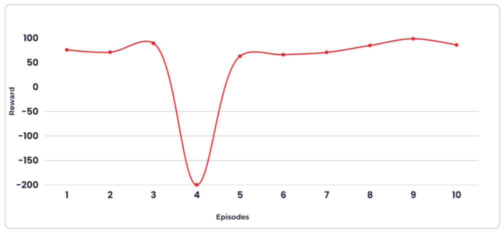
(d) Episode vs Reward for 1000 Episodes

Figure 28: Epsilon - 0.7 for 500 and 1000 Episodes

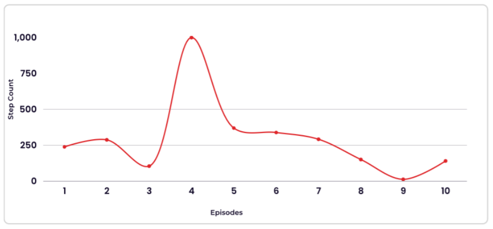
7.2.1.2 Boltzmann Q Policy

For the Boltzmann Q Policy, we can change the value of temperature to see how the performance of the model is varying. For this scenario, I have considered the following Temperature values.

$$\tau = \{0.1, 0.3, 0.7, 0.9\} \tag{19}$$

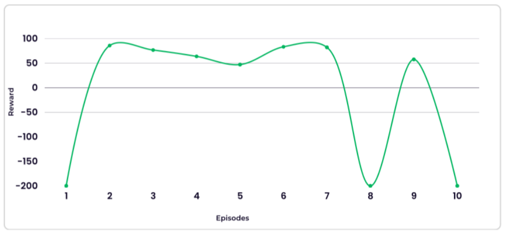


(a) Episodes vs Reward

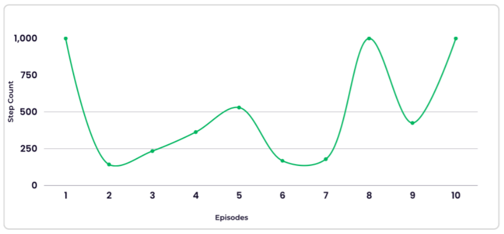


(b) Episodes vs Step Count

Figure 29: Temperature - 0.1 for 50 Episodes

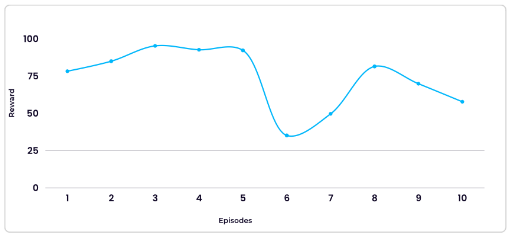


(a) Episodes vs Reward

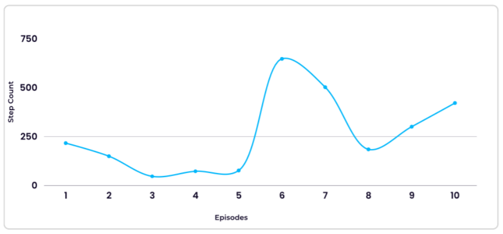


(b) Episodes vs Step Count

Figure 30: Temperature - 0.3 for 50 Episodes

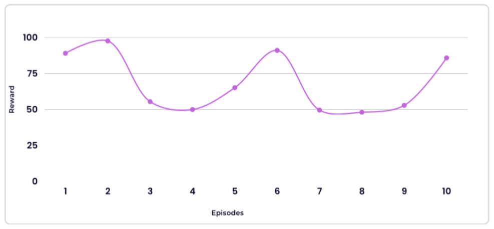


(a) Episodes vs Reward

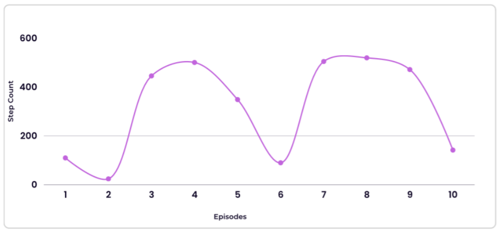


(b) Episodes vs Step Count

Figure 31: Temperature - 0.7 for 50 Episodes



(a) Episodes vs Reward



(b) Episodes vs Step Count

Figure 32: Temperature - 0.9 for 50 Episodes

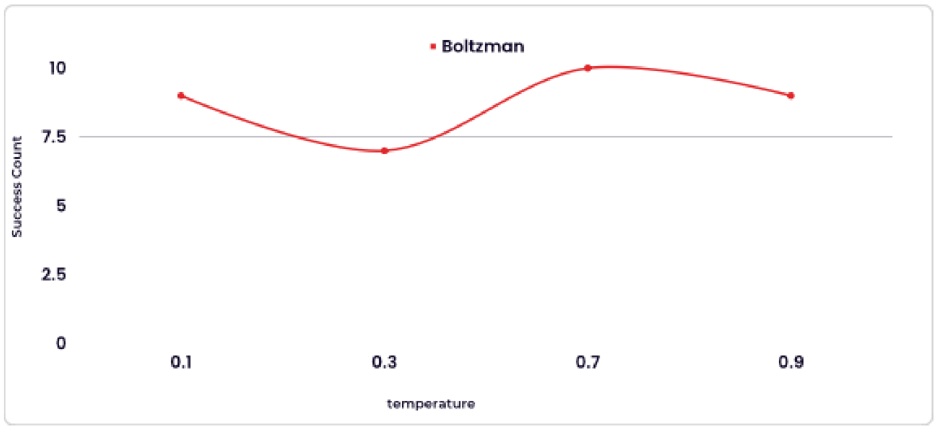
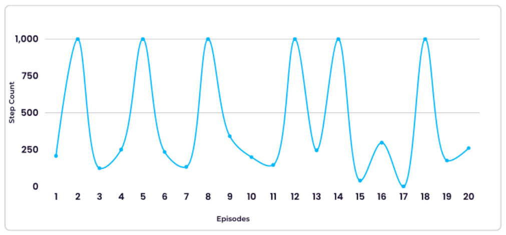
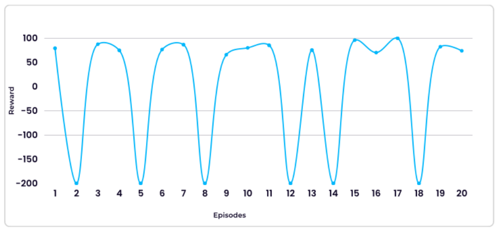


Figure 33: Airport Reached vs Temperature

From the following graph, it can be inferred that the temperature value that gave the best result was 0.7. The success rate increased from 0.1 to 0.7 and started decreasing after. Based on this, further experimentation was done on more episodes for Temperature 0.7

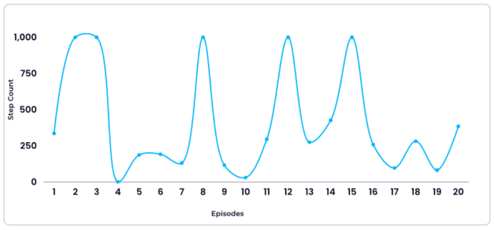


(a) Episode vs Step-Count for 100 Episodes

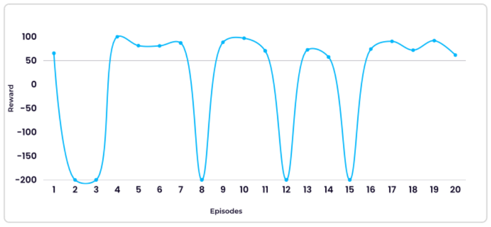


(b) Episode vs Reward for 100 Episodes

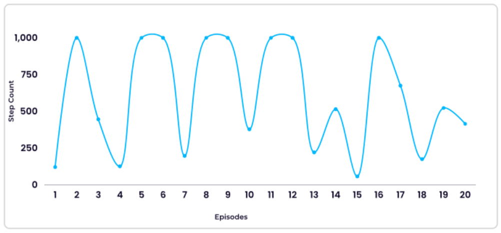
Figure 34: Temperature - 0.7 for 100 Episodes



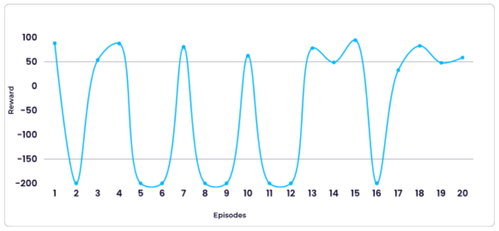
(a) Episode vs Step-Count for 500 Episodes



(b) Episode vs Reward for 500 Episodes



(c) Episode vs Step-Count for 1000 Episodes



(d) Episode vs Reward for 1000 Episodes

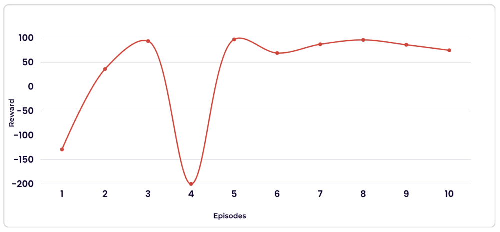
Figure 35: Epsilon - 0.7 for 500 and 1000 Episodes

7.2.2 Emergency Aircraft and Airport with 2 Obstacles

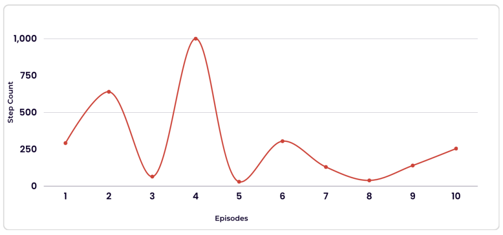
7.2.2.1 Epsilon Greedy Policy

For the Epsilon Greedy Policy, we can change the value of Epsilon to see how the performance of the model is varying. For this scenario, I have considered the following Epsilon values.

$$\epsilon = \{0.1, 0.2, 0.3, 0.5, 0.7, 0.9\} \tag{20}$$

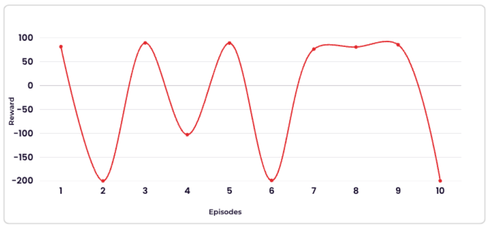


(a) Episodes vs Reward

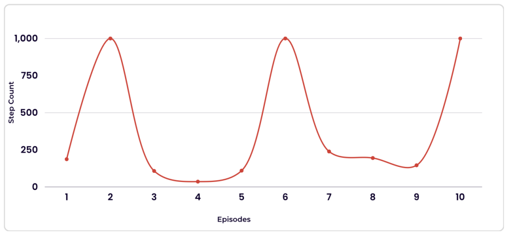


(b) Episodes vs Step Count

Figure 36: Epsilon - 0.1 for 50 Episodes 2 Obstacles

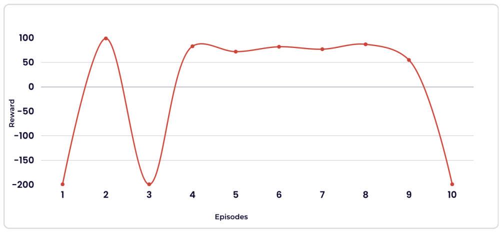


(a) Episodes vs Reward

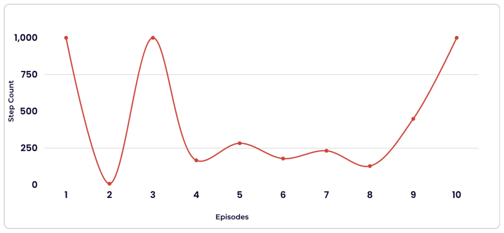


(b) Episodes vs Step Count

Figure 37: Epsilon - 0.2 for 50 Episodes 2 Obstacles

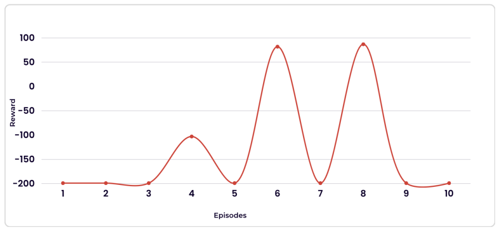


(a) Episodes vs Reward

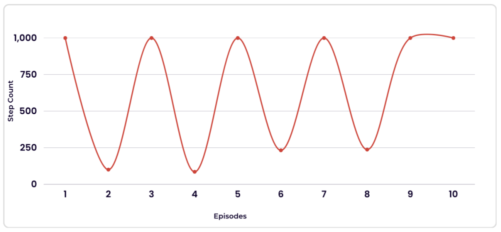


(b) Episodes vs Step Count

Figure 38: Epsilon - 0.3 for 50 Episodes 2 Obstacles

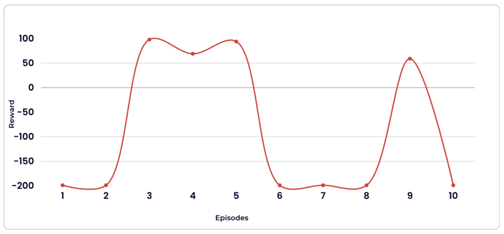


(a) Episodes vs Reward

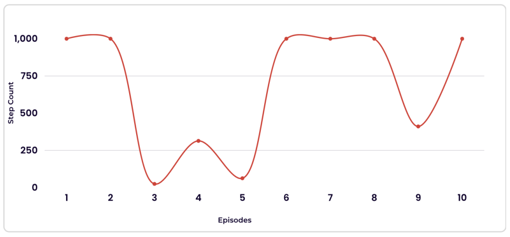


(b) Episodes vs Step Count

Figure 39: Epsilon - 0.5 for 50 Episodes 2 Obstacles

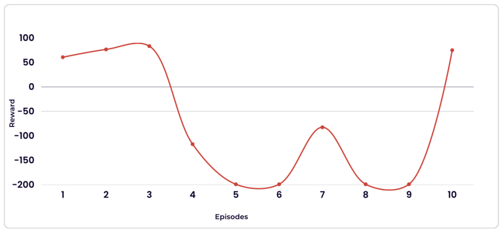


(a) Episodes vs Reward

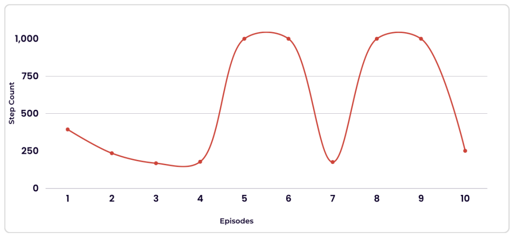


(b) Episodes vs Step Count

Figure 40: Epsilon - 0.7 for 50 Episodes 2 Obstacles



(a) Episodes vs Reward



(b) Episodes vs Step Count

Figure 41: Epsilon - 0.9 for 50 Episodes 2 Obstacles

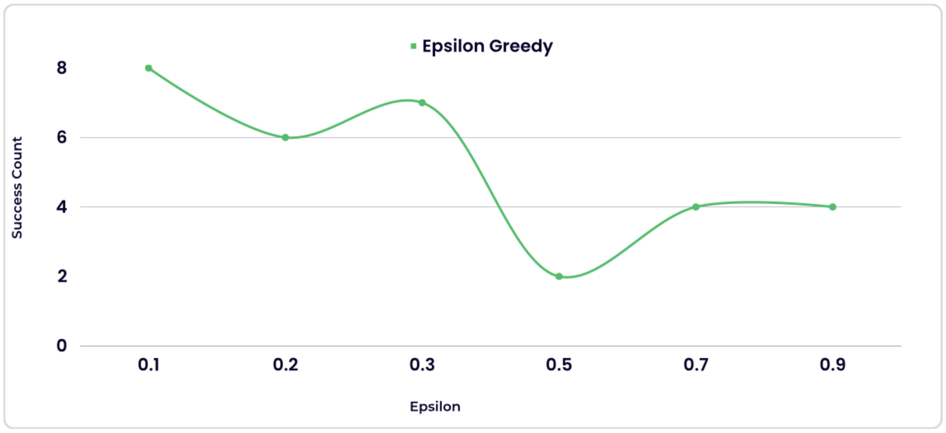
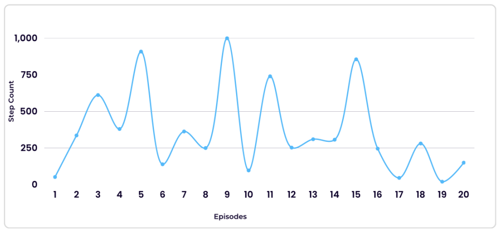
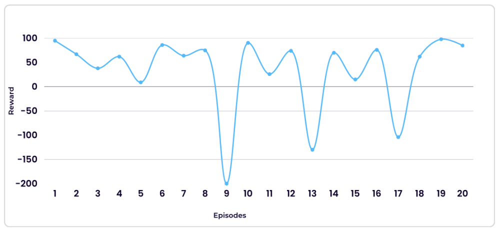


Figure 42: Airport Reached vs Epsilon for 50 Episodes

From the following graph, it can be inferred that the epsilon value that gave the best result was 0.1. The success rate has an overall decreasing trend from 0.1 to 0.9. Based on this, further experimentation was done on more episodes for Epsilon 0.1

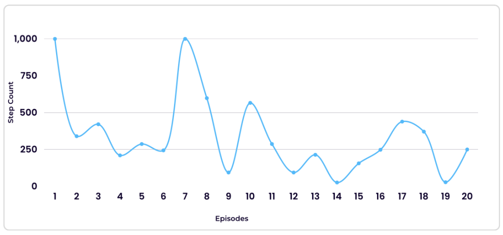


(a) Episode vs Step-Count for 100 Episodes

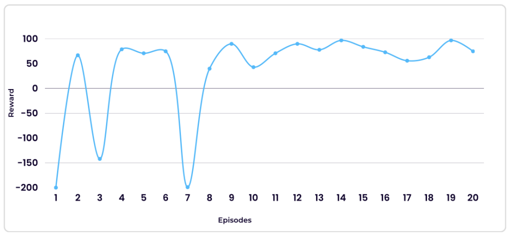


(b) Episode vs Reward for 100 Episodes

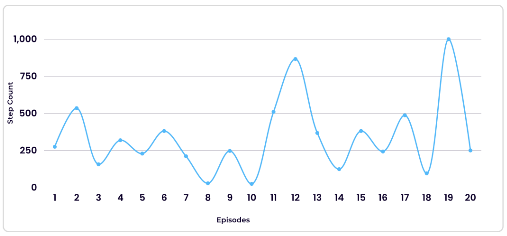
Figure 43: Epsilon - 0.1 for 100 Episodes and 2 Obstacles



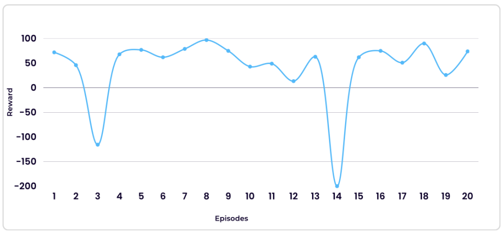
(a) Episode vs Step-Count for 500 Episodes



(b) Episode vs Reward for 500 Episodes



(c) Episode vs Step-Count for 1000 Episodes



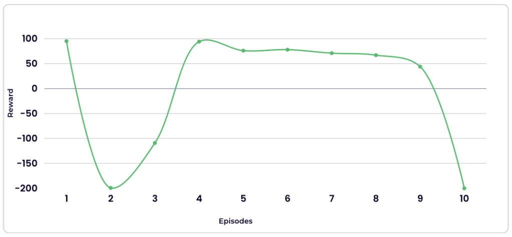
(d) Episode vs Reward for 1000 Episodes

Figure 44: Epsilon - 0.1 for 500 and 1000 Episodes with 2 obstacles

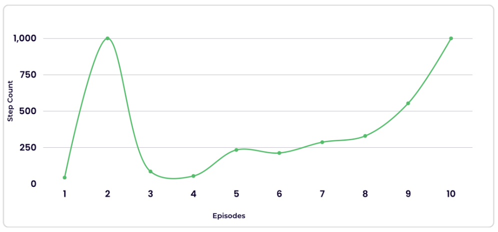
7.2.2.2 Boltzmann Q Policy

For the Boltzmann Q Policy, we can change the value of temperature to see how the performance of the model is varying. For this project, I have considered the following Temperature values.

$$\tau = \{0.1, 0.2, 0.3, 0.5, 0.7, 0.9\} \tag{21}$$

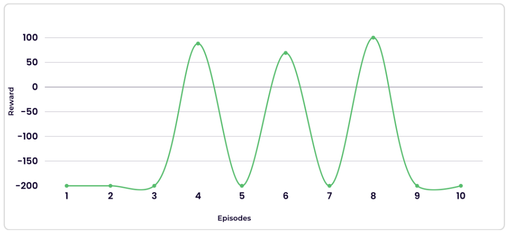


(a) Episodes vs Reward

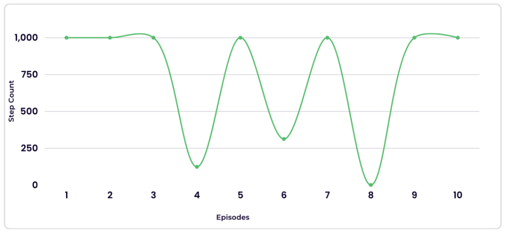


(b) Episodes vs Step Count

Figure 45: Temperature - 0.1 for 50 Episodes with 2 obstacles

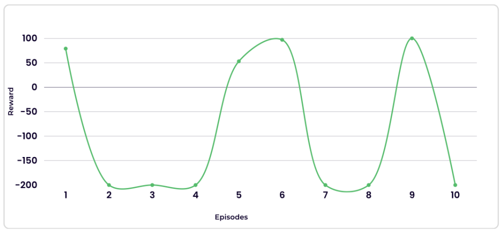


(a) Episodes vs Reward

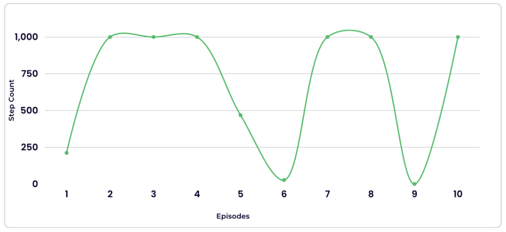


(b) Episodes vs Step Count

Figure 46: Temperature - 0.2 for 50 Episodes with 2 obstacles

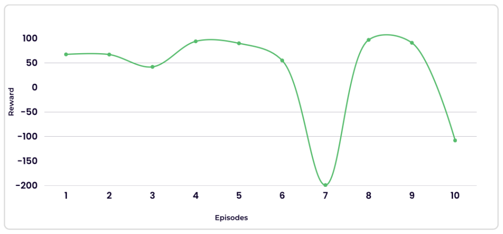


(a) Episodes vs Reward

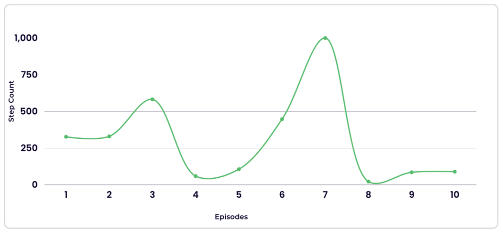


(b) Episodes vs Step Count

Figure 47: Temperature - 0.3 for 50 Episodes with 2 obstacles

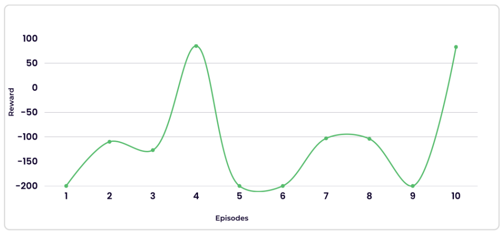


(a) Episodes vs Reward

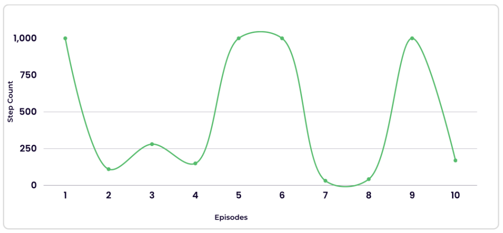


(b) Episodes vs Step Count

Figure 48: Temperature - 0.5 for 50 Episodes with 2 obstacles

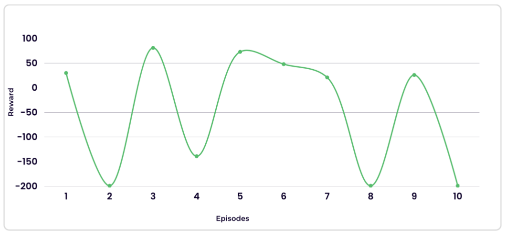


(a) Episodes vs Reward

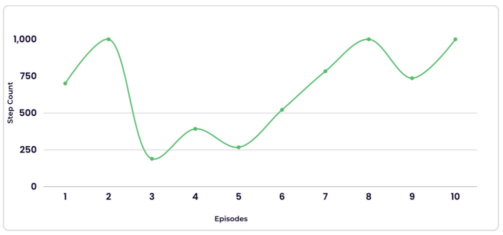


(b) Episodes vs Step Count

Figure 49: Temperature - 0.7 for 50 Episodes with 2 obstacles



(a) Episodes vs Reward



(b) Episodes vs Step Count

Figure 50: Temperature - 0.9 for 50 Episodes with 2 obstacles

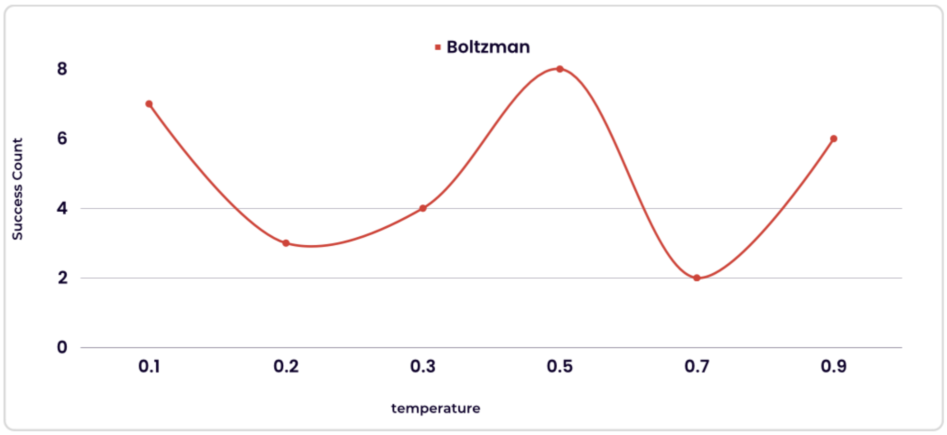
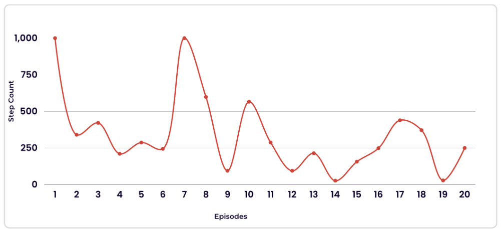
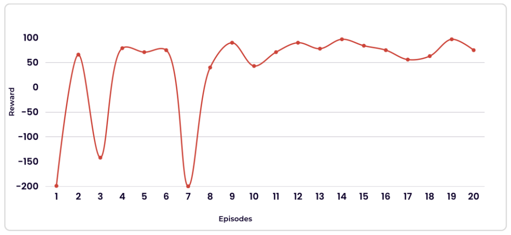


Figure 51: Airport Reached vs Temperature

From the following graph, it can be inferred that the temperature value that gave the best result was 0.5. The success rate has an overall wavy trend from 0.1 to 0.9, with peak performance at 0.5. further experimentation was done on more episodes for Temperature 0.5

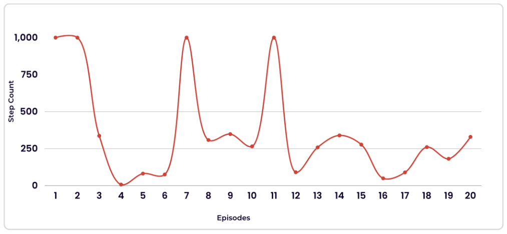


Episode vs Step-Count for 100 Episodes

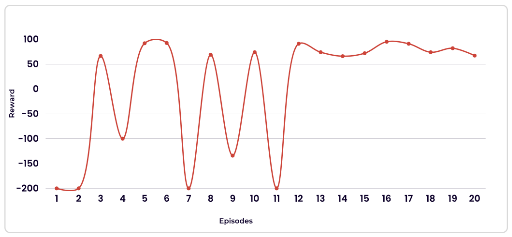


(a) Episode vs Reward for 100 Episodes

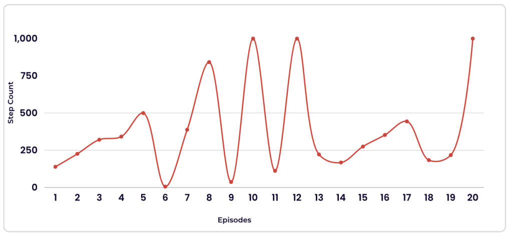
Figure 52: Temperature - 0.5 for 100 Episodes with 2 Obstacles



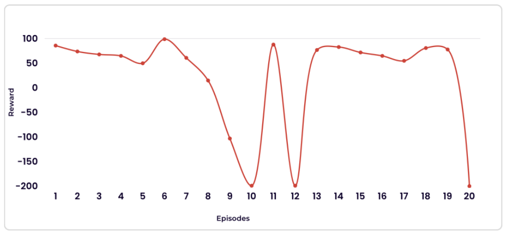
(a) Episode vs Step-Count for 500 Episodes



(b) Episode vs Reward for 500 Episodes



(c) Episode vs Step-Count for 1000 Episodes



(d) Episode vs Reward for 1000 Episodes

Figure 53: Epsilon - 0.5 for 500 and 1000 Episodes with 2 obstacles

7.3 Performance on Reaching an Airport

7.3.1 Emergency Aircraft and Airport Scenario

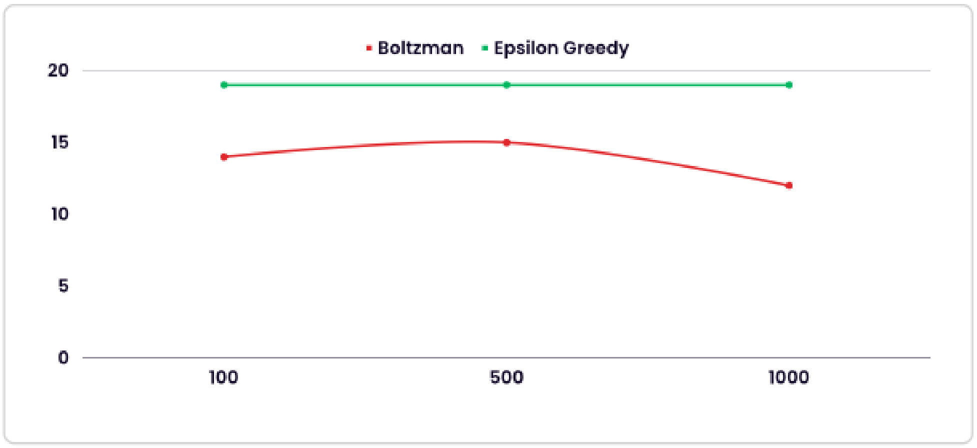


Figure 54: Airport Reached vs Episodes for Airport and Aircraft Scenario

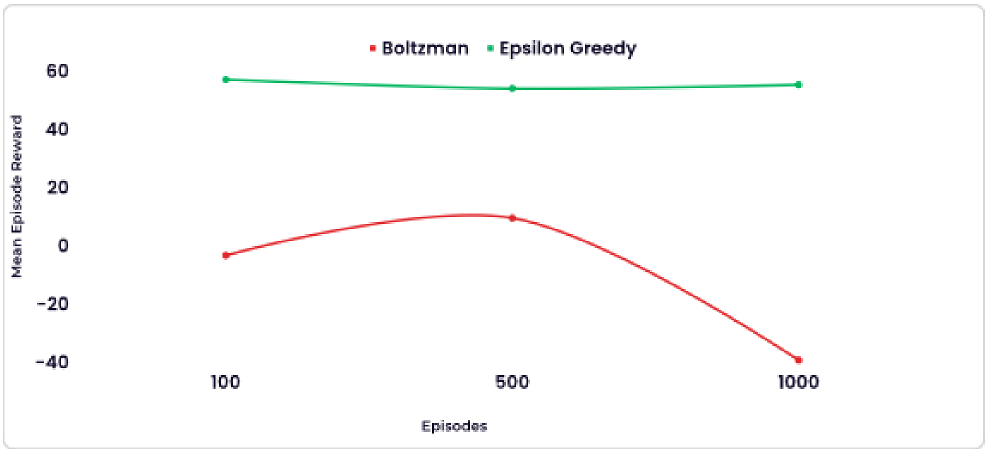


Figure 55: Mean Episode Reward vs Episodes for Airport and Aircraft Scenario

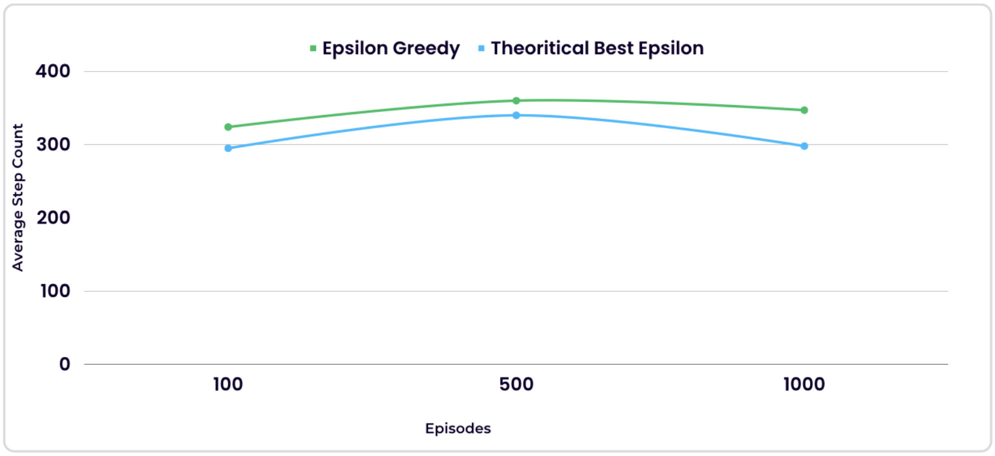


Figure 56: Average Step Count vs Theoretical Best Step Count for Airport and Aircraft Scenario for Epsilon Greedy

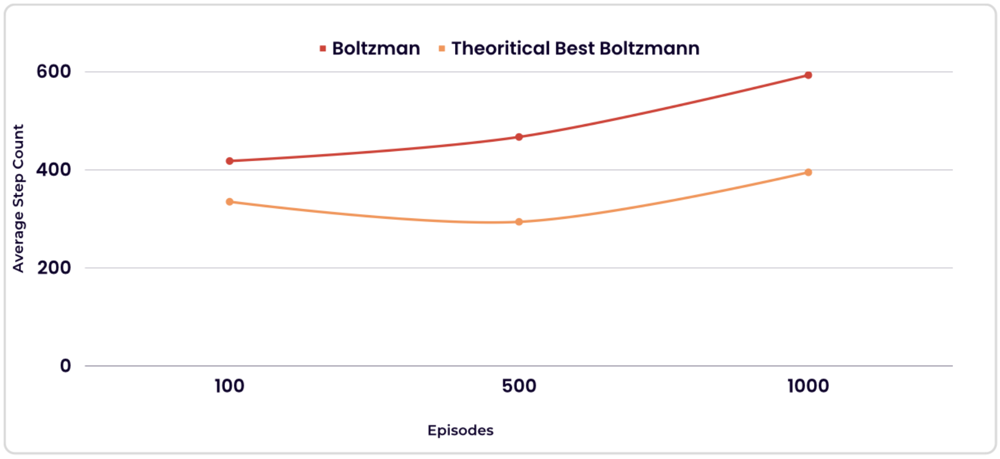


Figure 57: Average Step Count vs Theoretical Best Step Count for Airport and Aircraft Scenario for Boltzmann Q Policy

7.3.2 Emergency Aircraft and Airport with 2 Obstacles

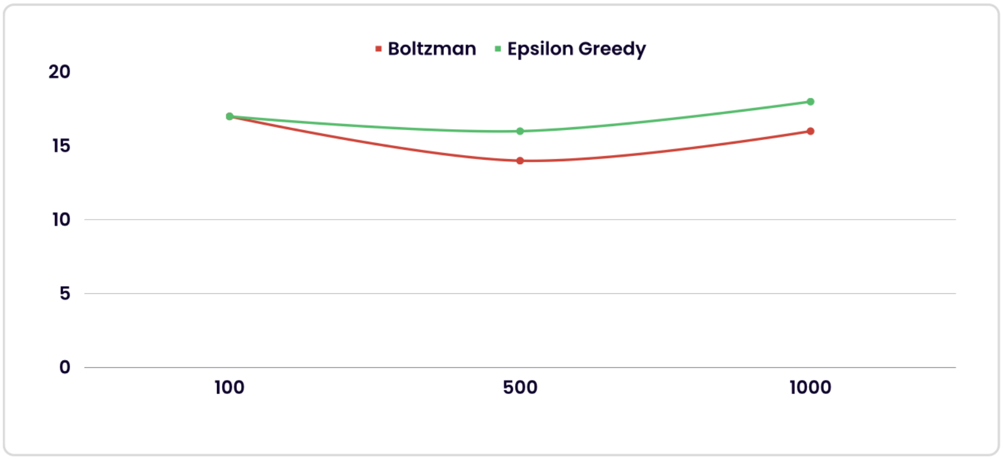


Figure 58: Airport Reached vs Episodes for 2 obstacles Scenario

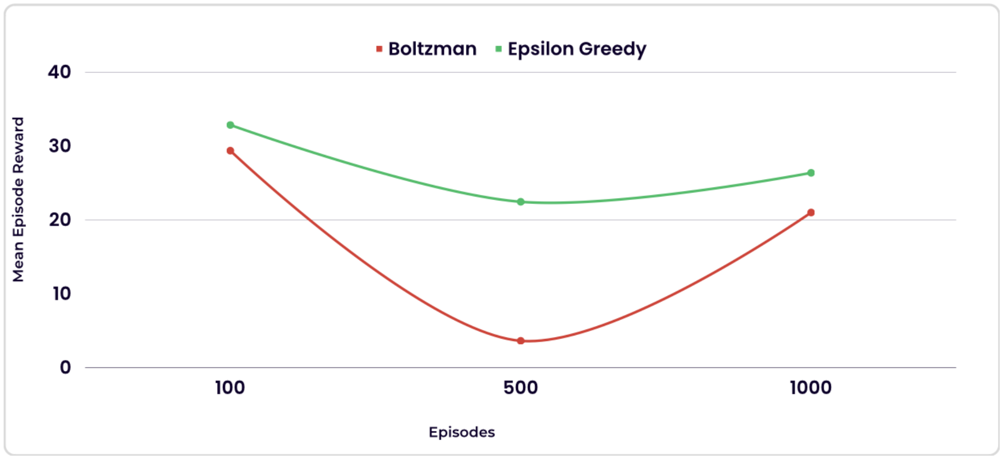


Figure 59: Mean Episode Reward vs Episodes for 2 obstacles Scenario

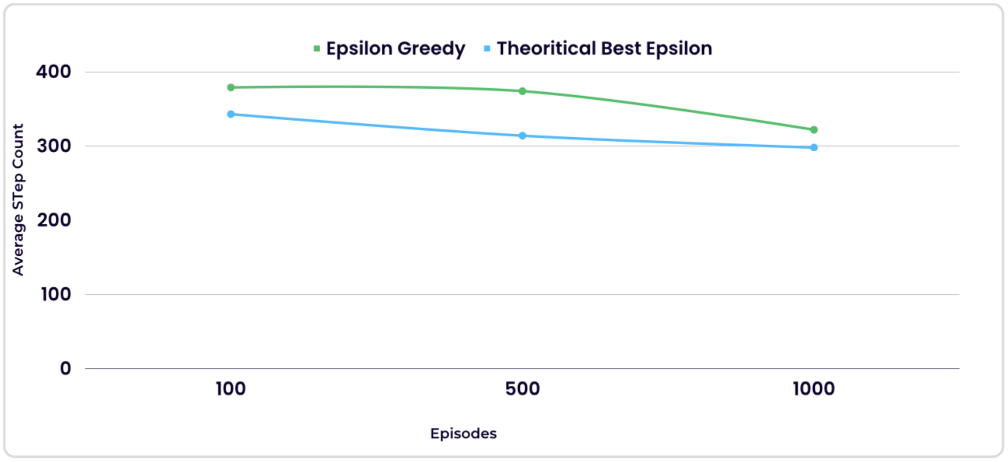


Figure 60: Average Step Count vs Theoretical Best Step Count for 2 obstacles Scenario for Epsilon Greedy

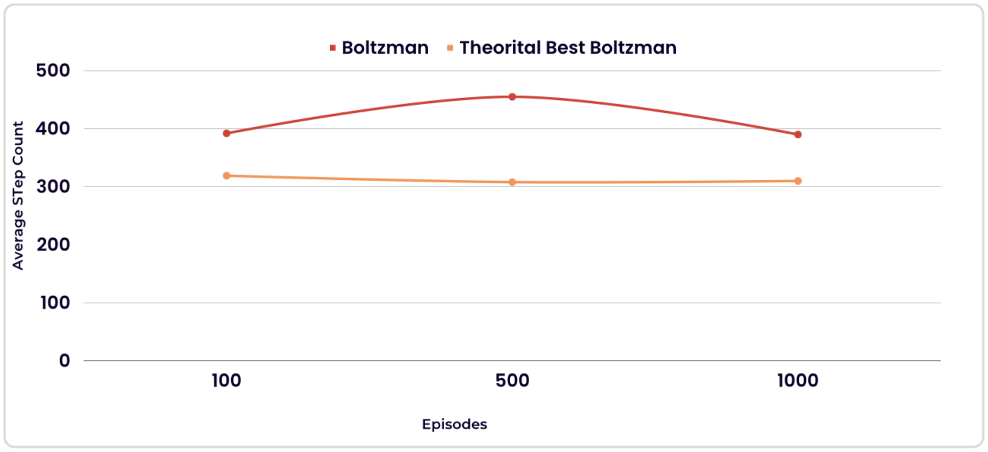


Figure 61: Average Step Count vs Theoretical Best Step Count for 2 obstacles Scenario for Boltzmann Q Policy

Figures 54 and 58 illustrate that the Epsilon Greedy policy performed better in learning an optimal ATC navigation behavior to direct the aircraft to the airport. Accordingly, Figures 55 and 59 indicate that the Epsilon Greedy Policy's Mean episode reward is higher than that of the Boltzmann Q Policy, meaning that under the Epsilon Greedy exploration, the learned agent directed the aircraft to the airport in less time.

Comparing the Average step count vs the Theoretical best step count, Figure 56 and Figure 60 show that the average step count for Epsilon Greedy is closer to the theoretical best compared to Boltzmann Q policy shown in Figure 57 and Figure 61.

When comparing the performance of Boltzmann and Epsilon Greedy policy, it can be inferred that the Epsilon Greedy policy performed better both in success rate, mean episode reward, and average step count. One reason why the Epsilon-Greedy policy performed better might be because of its simplicity. Epsilon Greedy policy is also more robust to any changes in the Q-Values. Unlike Boltzmann's Q-Policy, Epsilon Greedy Policy is less sensitive to small changes in Q value. Epsilon greedy policy is also quicker to converge to an optimal policy because of its exploration-exploitation trade-off. Due to these reasons, I believe Epsilon Greedy Policy is more suitable to this scenario considering the large state space and the reward signals used.

CHAPTER 8

Conclusion

Routing of emergency aircraft is one of the most stressful situations that the ATC faces and it needs the utmost attention and priority. If the ATC is not able to provide the required details to the airplane in time, it might result in catastrophic results. Reinforcement learning (RL) is one approach that can be used to create an ATC agent to help route the emergency aircrafts to the airport as quickly as possible. In this project, I have implemented a Deep Q Network (DQN) agent to successfully navigate an emergency aircraft to the airport in the airspace as fast as possible. The effectiveness of the DQN algorithm is demonstrated by the trend of increasing mean episode reward and decreasing trend of a number of steps, which is a glaring indication of the potential advantages of RL in this area.

Additionally, I have used two commonly used reinforcement learning exploration strategies. The strategies are the Epsilon-greedy policy and the Boltzmann Q policy. After testing on various episodes, it was observed that the Boltzmann Q policy was outperformed by the Epsilon greedy policy both in time and reward. It can be inferred that choosing a correct exploration strategy can have a significant impact on the outcome of the model. This research can be taken further by incorporating obstacles that are moving in the airspace to navigate the emergency aircraft to the airport. We can also add multiple emergency aircraft and multiple airports to make the scene close to the real world. We can also consider introducing the altitude of the aircraft and model in 3-D state space.

LIST OF REFERENCES

[1] M. A. Nielsen, "Neural networks and deep learning," http://neuralnetworksanddeeplearning.com/index.html, 2015.

[2] F. A. Administration and U. S. F. A. Administration, "Faa order jo 7110.65z - air traffic control."

[3] M. Hawley and R. Bharadwaj, "Application of reinforcement learning to detect and mitigate airspace loss of separation events," in *2018 Integrated Communications, Navigation, Surveillance Conference (ICNS)*, 2018, pp. 4G1--1--4G1--10.

[4] Y. Nakamura, R. Mori, H. Aoyama, and H. Jung, "Modeling of runway assignment strategy by human controllers using machine learning," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, 2017, pp. 1--7.

[5] K. Ng and C. Lee, "A modified variable neighborhood search for aircraft landing problem," in *2016 IEEE International Conference on Management of Innovation and Technology (ICMIT)*, 2016, pp. 127--132.

[6] M. Xiangwei, Z. Ping, and L. Chunjin, "Sliding window algorithm for aircraft landing problem," in *2011 Chinese Control and Decision Conference (CCDC)*, 2011, pp. 874--879.

[7] F. A. Administration and U. S. F. A. Administration, *Pilot's handbook of aeronautical knowledge.* Skyhorse Publishing Inc., 2009.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[9] Amber, "(deep) q-learning, part1: basic introduction and implementation," https://medium.com/@qempsil0914/zero-to-one-deep-q-learning-part1-basic-introduction-and-implementation-bb7602b55a2c, 2019.

[10] Z. Wang, H. Li, H. Wu, F. Shen, and R. Lu, "Design of agent training environment for aircraft landing guidance based on deep reinforcement learning," in *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 02, 2018, pp. 76--79.

[11] L. Lyu, Y. Shen, and S. Zhang, "The advance of reinforcement learning and deep reinforcement learning," in *2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, 2022, pp. 644--648.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[13] "Exploration strategies," https://analyticsindiamag.com/exploration-reinforcement-learning/.

[14] "Gym documentation," https://www.gymlibrary.dev/content/basic_usage/.