UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Data Science

A turn-based strategy game to evaluate Multi-Armed Bandit policies in an adversarial setting

Relatore: *Prof.* Antonio Candelieri
Co-relatore: *Dott.* Riccardo Perego

Tesi di Laurea Magistrale di:

Davide Lagano
Matricola 838358

Anno Accademico 2019-2020

"Be realistic, demand the impossible!"
Che Guevara

To my parents and Dora

Acknowledgements - Ringraziamenti

A tesi finita, mi ritrovo a pensare a questi due anni di magistrale pieni di emozioni e di eventi. Prima di passare ai ringraziamenti, vorrei ripercorrere questi anni di magistrale. Il tutto è iniziato con un sentimento di spaesamento, passando da una facoltà completamente diversa, a quella di Data Science. Dopo un anno ecco l'esperienza più bella della magistrale, in cui ho conosciuto gente fantastica: l'Erasmus. Successivamente, la pandemia, durante la quale il mio percorso universitario giungerà a compimento. Periodo in cui in molti hanno perso i propri cari e ci si è effettivamente resi conto di quanto le relazioni umane abbiano, o dovrebbero avere, un peso importante nelle nostre vite, ormai immerse in una società capitalistica basata sul consumismo. Citando uno degli economisti più importanti di sempre: "I vizi specifici dell'economia che viviamo sono due: il lavoro non è assicurato a tutti e i profitti sono divisi in modo arbitrario e iniquo." (John Maynard Keynes)

A me la speranza di poter contribuire nell'evoluzione della società verso principi di eguaglianza, speranza, inclusione.
A me la speranza lavorativa di poter lasciare il segno, in positivo, in un settore controverso e attualmente non del tutto compreso.

A me l'augurio di credere sempre in me stesso, di aver coraggio nel prendere decisioni difficili che mi si presenteranno in futuro.
A me l'augurio di mettere le relazioni al primo posto.

Infine, un ringraziamento speciale ai miei genitori, sempre presenti nella mia vita e che hanno permesso di farmi arrivare fin qui.
Ringrazio inoltre: il mio relatore, Candelieri; Giorgos, disponibile ad aiutarmi durante i problemi tecnici; Chiunque io abbia incontrato durante questi due anni, che consapevolmente o meno, hanno permesso di farmi diventare chi sono.

A Dora, invece, un grazie esclusivo.

Table of Contents

List of Figures

Chapter 1

Introduction

This chapter will be divided into different sections. In the first part, the reasons that led me to undertake this study will be explained, and its goals will be clarified. After that, a section will follow where the origins of the problem will be briefly described. Then possible applications of the Multi-Armed Bandit problem will be listed and explained. Finally, an overview of the work and the output will be concisely explicated.

1.1 Motivations and goals

In a world that year after year is increasingly representing the definition of Infosphere, which produces huge amounts of every type of data, the Data Science discipline has become of great importance. Companies are increasingly basing their activities on valuable data collection. Since modern technologies have enabled the creation and storage of increasing amounts of information, data volumes have increased rapidly. The role of Data Science is to interpret these data and find patterns inside the data in order to make the best decisions possible and to create new attractive products.
This study will be based on a classic reinforcement learning (RL) problem, namely the Multi-Armed Bandit (MAB) problem. MAB concerns taking decisions under uncertainty and under a limited set of resources, by allocating resources while balancing between

exploration (i.e., making a decision to reduce uncertainty) and exploitation (i.e., making the best decision concerning the knowledge collected so far). RL is a branch of Machine Learning, and the primary purpose is to create and use algorithms that can learn by themselves. The purpose will be to apply and compare different types of algorithms in two different settings: one decision-maker and two (adversarial) decision-makers, respectively.

1.2 What is Machine learning and Reinforcement Learning?

One of the multiple branches of Data Science is Machine Learning. There is a variety of definitions of Machine Learning. The most representative are:

- Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed. - (Arthur Samuel, 1959)
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. - (Tom Mitchell, 1997)

Furthermore, according to Aurélien Géron, there are multiple different Machine Learning systems, that are possible to be classified in categories, considering the following criteria:

- The human supervision is present or not (supervised, semi-supervised, unsupervised, Reinforcement Learning)
- They can learn incrementally on the fly or not (batch versus online learning)
- They are based on comparing new data values to known data values (instance-based learning) or on training the model with a training set due to finding patterns in the data points and then building a predictive model (model-based learning).

Out of the categories mentioned above, this study will focus on Reinforcement Learning (RL). RL's main characteristic is called agent; it observes the environment and makes choices based on rewards and penalties. Then it is able to learn by itself what is the best choice. Based on these choices, the agent builds up a strategy called policy. The goal of the agent is to gain the best reward or the smallest penalty.
A good example of Reinforcement learning is DeepMind's AlphaGo: a computer program able to beat the world champion of the game Go. It learned the winning policy by analyzing millions of matches and playing against itself. [2]

1.3 The Multi-Armed Bandit problem

In this study, an RL topic, the Multi-Armed Bandit (MAB) problem, will be analysed.
The Multi-Armed Bandit problem is an optimal learning topic principally used to make decisions over time in an uncertain environment.

The original story to define the Multi-Armed Bandit problem is also the easiest one: it bases its concepts on slot machines. Firstly, the 'One-armed bandit' relates to one slot machine and its mechanism for playing. This means pulling a lever (the arm) with a related unknown reward represented by the payoffs for hitting the jackpot. The Multi-Armed Bandit considers multiple slot machines with different probabilities of winning.
The MAB problem has a limited set of resources (bags of money) that have to be allocated in different alternatives (slot machines) with partial information available.
The mechanic of the problem is simple: suppose that there are a defined number of slot machines K available and there are enough coins to be able to play T times. The set of all arms is A. Playing x slot machines at time n makes it possible to get a random reward W_x^{n+1}.
In the formula 1.1 the elements not explained so far are: the value of an arbitrary action a and the action selected on a certain time step A_t.
Then $q_x(a)$ is the expected reward, by selecting the action a.

$$q_*(a) = \mathbb{E}[W_x^{n+1} | A_t = a] \tag{1.1}$$

[7]

The goal is to predict the expected value of each arm's reward μ_x as close as possible to the real reward value. This is possible by finding the best trade-off between exploration and exploitation. In the exploration, the surrounding is explored, therefore less known slot machines are played. This increases the risk of gaining a lower reward, but at the same time increases the possibilities to find a slot machine that gives a higher reward. In the exploitation, the arm that gives the best reward is chosen. This could maximize the reward, but it eliminates the chance to explore better slot machines. The only way to estimate the mean of every slot machine μ_x is to insert coins in it and memorize the observations.
In the Multi-Armed bandit, the process is online. Every slot machine has an unexplored value called "average winning" that is possible to estimate and update at each time step by observing the winning W_x^{n+1}. Then it will update the thoughts about that arm in real-time. For this reason, at every iteration, the arm that can be pulled can be different.

It is possible to represent the object function of this problem in this way:

$$\max_{\pi \in \Pi} \mathbb{E}^{\pi} \sum_{n=0}^{N} \gamma^n \mu_x^n \tag{1.2}$$

[7] x^n is the decision made at time n and $\gamma \in (0, 1]$ is a discount factor.
In this explanation the most simple reward is considered. The reward of each arm is based on Bernoulli distribution and can be either 1 or 0. The slot machines example is not itself a practical application.
Some practical applications follow in the next sub-chapter.

1.4 Possible applications of the Multi-Armed Bandit problem

There are multiple applications of the Multi-Armed Bandit problem that have been studied, some of them are:

- Advertising:

Every time a user visits a website, the owner has to show which one of N advertisements is much more profitable to display. There is no information about the user, the content of the advertising and the web page content required. A binary reward is gained, being 1 if the user clicks on the advertisement, and 0 if it does not.
In this context, the contextual bandits could be useful because it is possible to categorize the user's query as context and every time an advertisement fails in one web page, it can be shown in another context.

- Clinical trials:

Before to make a treatment for a disease public, three phases need to be passed. In this context, the second phase is essential. It consists of a test in which the treatment is administered on two large groups of people: at one group, a real treatment dose and at the other group a placebo treatment. To gain data about the dose-toxicity relationship, it is impossible to include a big group of people and randomize them at all the possible dose level considered in the trial. Some part would be exposed to a high level of toxicity and some part to an ineffective dose levels. This would be unethical. To make it ethically responsible, it is possible to adapt the Multi-Armed Bandit problem to it. The MAB approach would choose the dose sequentially, at every iteration, considering the previous results. This problem can be identified as a multi-armed bandit problem considering a trade-off between rewards and error probability. [3]

- Network server selection:

Suppose that several servers have a different processing speed and a job has to be processed to one of it. By using the MAB approach, it is possible to consider every single server as an arm. The goal is to learn at every iteration which server is the best to work with, and then choose which one fits best.

1.5 Outline and overview of the Master Thesis

In this study, a MAB game will be created. The protagonists of this game are one or two spaceships, guided by different MAB strategies. Their goal is to travel to other planets to mine and collect as many crystals as possible. The study's mission is to analyze which algorithm suites best in certain situations, considering different contexts. The MAB versions that will be analyzed are the contextual multi-armed bandit problem and the adversarial bandit.
In the contextual MAB problem, one spaceship is playing. The focus is on finding the best MAB algorithm that can find the best strategy, considering the context's information. In the adversarial bandit, two spaceships are playing. The focus is on finding the best MAB algorithm that can find the best strategy considering that an adversary modifies the context.
The algorithms used are:

- Mean Algorithm
- UCB – The Upper Confidence Bound Algorithm
- Time elapsed based MAB Algorithm
- Thompson Sampling Algorithm
- Thompson Sampling with Probability Matching Algorithm

All these algorithms will be tested in two different versions: with and without the implementation of the Epsilon-Greedy strategy. Moreover, all the algorithms will be tested on planets with two versions of crystals production:

- For each planet, the amount of crystals produced per unit of time is described by a Gaussian distribution (with a planet specific mean and variance values).
- For each planet, the amount of crystals produced per unit of time is described by a planet specific probability distribution (which could be different from the Gaussian one).

The effectiveness of the different MAB strategies, in the different situations, are analysed by considering the number of crystals collected by each player along the game. Every decision requires a different cost in terms of crystals: the closer is the planet chosen, the lower the amount of crystals to be used to reach it.
As it is possible to see in figure 1.1, not only the game logic but also a graphical interface has been developed to visualize how a game play evolves over time. All the graphic parts of the game will be explained in chapter 3.
To briefly explain the results of this Master Thesis, there is no evidence of a difference in terms of performances between being in the context in which the production of the planets' crystals are Gaussian or if the distributions are different. Moreover, the Thompson Sampling Algorithms and, in general, all the algorithms that implement the Epsilon-Greedy strategy, offer the best performances.

Figure 1.1: Results of two spaceships

Chapter 2

Literature review

Now that the MAB problem is defined, chapter 2 aims to analyze the state of the art.
The main paper that inspired this study's idea was: *Exploration-Exploitation in a Contextual Multi-Armed Bandit Task* of *Eric Schulz, Emmanouil Konstantinidis and Maarten Speekenbrink*. [1]
This article comes from the scientific community of neuroscience. In the article, the ways humans balance between exploration and exploitation have been explored in turn-based gaming contexts. It focused on an experimental paradigm named Mining in Space; this means that multiple planets have to be mined from one or several participants. The goal of the [1] paper is to "use the Contextual MAB task as a method to assess decision making in uncertain environments and test how participants behave in this task" [1].
Participants had to decide which of the four planets they wanted to mine, described by 3 different binary elements, being the contexts. The paper's result was that participants were able to adapt their decision to the context well, even in potentially non-stationary environments through probability matching. A Contextual Gaussian Process algorithm can best describe participants' choices that probability matches according to expected outcomes.

Chapter 3

Definition and framework

In this chapter, the development of the turn-based strategy game will be explained. Furthermore, the visualization of the game, the data structure and the analytical framework will be defined.

3.1 Design of the game

The application of the MAB problem that will be discussed in this study regards a turn-based strategy game.
The environment of the game is:

- Ship 1: first spaceship.
- Ship 2: second spaceship.
- One *Home* planet.
- N: planets for crystal's extraction (*Home* excluded). The planets are Mars, Venus, Uranus.
- $M_i^{(t)}$: quantity of crystals on the planet i at time t. Each planet will start with 0 crystals on it.

- P_i: probability distribution of the planet's production of crystals of planets i per time unit.
- τ_i: travel time (per time unit) between *Home* and the planet i. The τ_i for Mars is 1, for Venus is 2, for Uranus is 3.
- c: costs (in crystals) per travel or upload or download per unit time. The c for each iteration is 1.
- K^t: crystals owned at time t.
- *State*: state of the spaceship. Which can be on ready, travelling or operating state.
- q: quantity of crystals on the spaceship.
- Q_{max}: maximum capacity on the spaceship. The Q_{max} for both spaceships is 2000.
- T: duration of the match (per time unit). The T is 10000.

The game logic consists of one or more spaceships guided by RL algorithms that start their trip from 'home' planet, bound for the planet that believes can gain most crystals possible. The goal is to build up the best strategy in order to own as many crystals as possible at the end of the game K^T.
There are 3 planets considered in the game: 'Mars', 'Venus', 'Uranus'. Every planet updates its production of crystals at every iteration based on the probability distribution in use on that planet. As will be seen in this chapter, the distribution can be the same for every planet, but with different means and standard deviations, or different for every planet. There are some conditions of the spaceship:

- Every time the spaceship is in *ready* state, it has to choose the planet i* where to go to collect the crystals

$$(i* \in 1, ..., N) \tag{3.1}$$

- The spaceship does not know the distribution of probabilities $P_i, i = 1, ...N$ because it is a 'learning and optimization' task. Therefore multiple strategies based on finding the best trade-off between exploration and exploitation will be used.
- Uploading the crystals from the planet i* and downloading them in the *home* planet needs one time unit, regardless of the number of crystals.
- Downloading and uploading are both associated to the state' operating'.
- Once the spaceship finishes the uploading it immediately leaves, direction *home*.
- Whenever the ship is in the state *ready* it immediately chooses the destination planet.

The RL algorithms are based upon the following assumptions:

- The only rewards the spacecraft considers are the uploaded crystals. The possibility of obtaining rewards if other choices have been made is not considered. This is called *bandit feedback*.
- The *reward distribution* is IID: every planet bases his production of crystals on a distribution. Every time an action a is chosen, the chosen reward is based independently from the distribution, that is initially unexplored by the algorithms.
- In the previous chapter, a classical Multi-Armed Bandit problem was introduced. The discount factor of it was $\gamma \in (0, 1]$. In this study's application, the discount factor is not Boolean but it is the number of crystals that the spacecraft uploads $\gamma \in q$.

Moreover, as explained in the chapter before, there are different MAB versions that will be implemented:

- Firstly, the focus is on the contextual multi-armed bandit. In this MAB version, each round's rewards depend on a context observed by the algorithm before making a decision. Only one spacecraft is involved. In this case, the spacecraft will work in an 'optimal' environment.

- In the second scenario, the focus will be on the adversarial bandit. In this case, two spaceships are involved in the game; this means that the environment is not optimal. In this case, the expectation is that the algorithms will work worse because every spacecraft is independent, and they don't know anything about the rival's strategy.

Finally, let's talk about the crystal production of the planets. In both versions previously explained, firstly, the probability distributions Pi of all the planets will be based on Gaussian distribution; afterwards, all the planets' production will be based on different distributions. The production of crystals of the planets is not correlated to each other. The four different situations will be listed below:

- In the contextual multi-armed bandit scenario with productions of crystals based on Gaussian distribution for all the planets, the distribution for the production of crystals of Mars has a mean of 5 and a standard deviation of 1; the distribution for the production of crystals of Venus has a mean of 6 and a standard deviation of 1.5; the distribution for the production of crystals of Uranus has a mean of 6 and a standard deviation of 2.

- In the adversarial bandit scenario based on Gaussian distribution for all the planets, the planets' production gets double values because there are 2 spaceships instead of 1. This way, it is possible to compare the two scenarios. The distribution for the production of crystals of Mars has a mean of 9 and a standard deviation of 2; the distribution for the production of crystals of Venus has a mean of 11 and a standard deviation of 3; the distribution for the production of crystals of Uranus has a mean of 12 and a standard deviation of 4.

- In the contextual multi-armed bandit scenario based on different distributions for each planet, the production of Mars is based on a Gaussian distribution with a mean of 6 and a standard deviation of 1.5; The production of Venus is based on a Poisson distribution with a Lambda value of 5; The production of Uranus is based on a negative binomial distribution with 18 trials and a 0.75 probability of success.

- In the adversarial bandit scenario based on different distributions for each planet, the production of Mars is based on a Gaussian distribution with a mean of 11 and a standard deviation of 3; The production of Venus is based on a Poisson distribution with a Lambda value of 11; The production of Uranus is based on a Negative binomial distribution with 35 trials and a 0.75 probability of success. As defined before, the goal of this study is to compare the algorithms and check which ones work better under certain circumstances. In order to do it, the algorithm's cumulative reward of each spacecraft will be compared in the charts.

3.2 Development of the game

In figure 3.1 it is possible to see one framework of the study, represented by a block diagram. It considers the case where there is only one spaceship.
The input that is needed in order to start the framework is:

$$\begin{aligned} N,\ T,\ c,\ Qmax,\ seed \\ set\tau_i \ \ i = 1, ..., N \\ setP_i \ \ i = 1, ..., N \\ M_i^{(0)} = 0 \ \ i = 1, ..., N \\ K^{(0)} = k \end{aligned}$$

According to figure 3.1, the game would start at time 0. The state of the spaceship is initialized as ready, the quantity of crystals on the spaceship is 0, and all the position variables are set on none.
As a first step, the spaceship checks if the actual time of the game is minor then the final game time. If it is not, the game is finished; otherwise, it checks the state of the spaceship, where it found the three main branches:

- State = 'ready': the spaceship chooses the destination based on the strategy in use

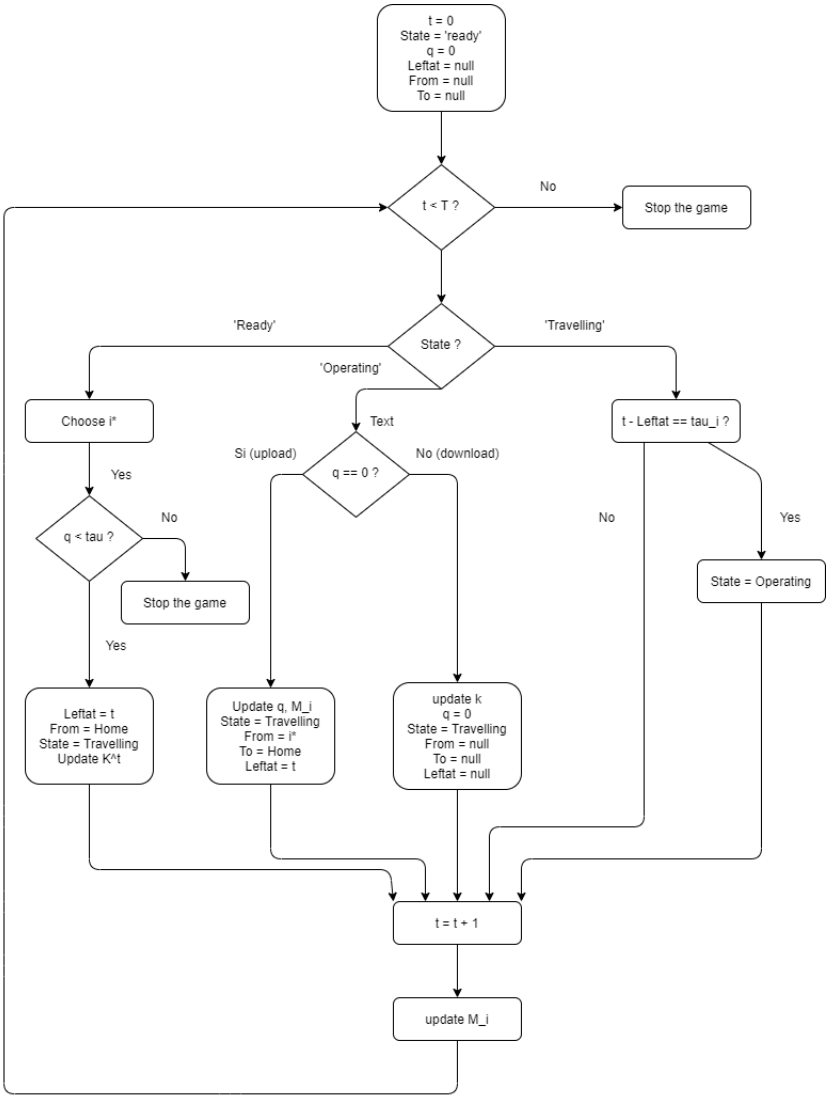

Figure 3.1: framework of the turn-based strategy game with one spaceships

and checks if there are enough crystals to pay the fuel to go to the destination planet. If it does not, there is not enough fuel to travel; consequently, the game is finished. If it does, the position variables, the state and the crystals are updated straight away. The crystals are updated because they are converted into fuel. Hence this formula is

applied:

$$K^t = K^t - (2 * \tau_{i*} + 2) * c \tag{3.2}$$

- State = 'operating': in this branch, the game checks if the quantity of crystals in the spaceship is equal to 0. If it is, this means that it is on the destination planet and needs to upload crystals from it. In this case, the position variables, the quantity of crystals on the destination planet and the quantity of crystals in the spaceship are updated:

$$q = min(M_{i*}, Q_{max}) \tag{3.3}$$

$$M_i = M_i^t - q \tag{3.4}$$

  Formula 3.3 uploads the minimum value between the number of crystals on the planet and the maximum capacity on the spaceship.
  Formula 3.4 updates the crystals on the destination planet.
  If $q \neq 0$ the spaceship is at planet home and it has to download the crystals previously updated. This means update K^t according to the formula 3.5 and initialise $q = 0$. All the position variables are updated.

$$K^t = K^t + q \tag{3.5}$$

- State = 'travelling': in this branch, the game checks if the destination planet's time distance is equal to the difference between the actual time and the time when the spaceship left the planet. If this condition is true it means that the spaceship arrived at the destination planet and the state is updated in *operating*; otherwise, it repeats this block until the condition becomes true.

Finally, the last two blocks: the first updates the time, the second updates the crystals owned by the planets.

In figure 3.2 the framework of the turn-based strategy game with two spaceships is

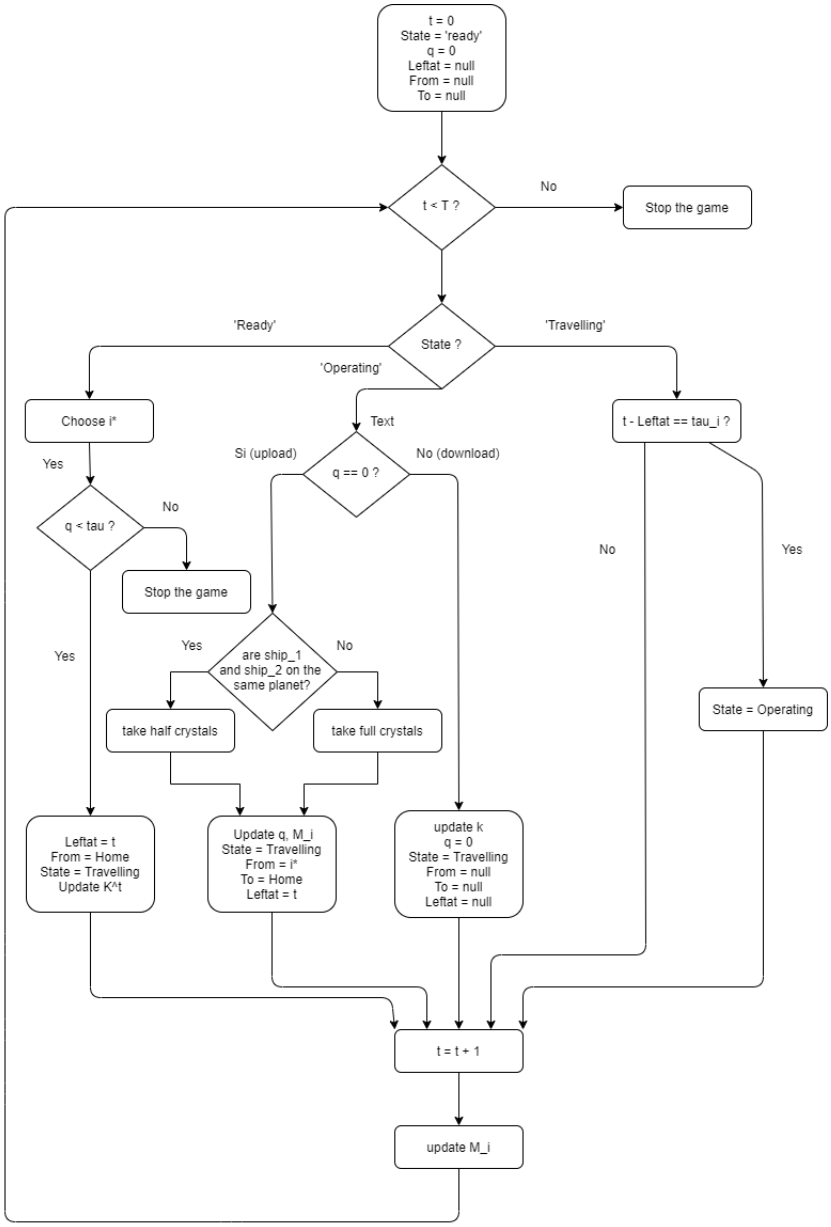

Figure 3.2: framework of the turn-based strategy game with two spaceships

represented. As it is possible to see, the only difference between the framework 3.1 and the framework 3.2 is that in the framework with two spaceships there is an extra control in the uploading root. This check is implemented in case the following three conditions occur:

- Both spaceships are on the same destination planet.
- Both spaceships arrive at the same time.
- There are not enough crystals to fill the maximum capacity of both.

It is possible to see the formula 3.6 used in this case:

$$q = \frac{min(M_{i*}, Q_{max})}{2} \tag{3.6}$$

3.3 Visualization

For this game, a visualization has been developed. In the images shown below, it is possible to see some frames of the visualization.
Starting with the game's variation with one player: figure 3.3 represents the game's initial phase. At the beginning of the game, as it is possible to see from figure 3.3, the spaceship's position is 'home'. At the bottom left of the image, the score of the spaceship is shown. As mentioned earlier, the spaceship starts with 200 crystals in the warehouse. In figure 3.3 there are 190 crystals available because the spaceship already chooses the destination planet and based on the τ it converted the crystals in fuel.
In figure 3.4, 3.5 and 3.6 it is possible to see the spaceship in the 'travelling' phase, directed to Mars, Venus and Uranus. Random frames were taken. For this reason the score of the spaceship doesn't follow the order of figure 3.3. Finally, figures 3.7, 3.8 and 3.9 show the spaceships on the destination planets.
In the variant with 2 spaceships the only differences with the previous game setting are that two spaceships and scores are visualized, competing with each other. Figures 3.10,

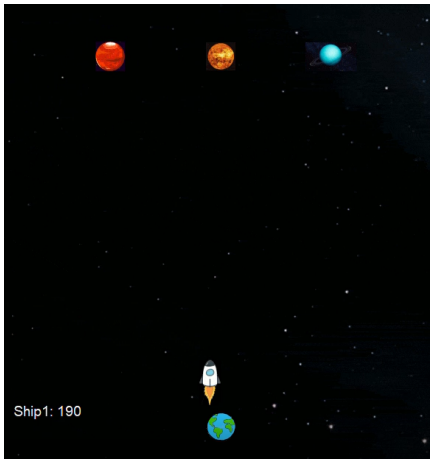

Figure 3.3: Spaceship at home

3.11, 3.12, 3.13, 3.14 and 3.15 show some frames regarding different phases of the ship game.

3.4 Data Structure

In this sub-chapter, the data structure is analyzed. In this data structure, the data used in order to analyze the results are stored.
Once the game starts, a dataframe is created and at each iteration updated. The dataframe is saved as .csv file and contains the following variables:

- t: current time
- *Ship name*: name of the spaceship.
- *State*: state of the spaceship.
- *Destination*: destination of the spaceship.
- *AvailableCrystals*: available crystals on all the planets.
- *AvailableCrystals Marte*: available crystals on Mars.

- *AvailableCrystals Venere*: available crystals on Venus.
- *AvailableCrystals Urano*: available crystals on Uranus.
- *crystals loaded*: quantity of crystals loaded on the spaceship.
- *crystals owned*: quantity of crystals stored at home. It is also called 'score'.

Figure 3.16 is an example of the daretaframe just explained. It represents the first 10 iterations of the game, in the adversarial strategy variation.

3.5 Analytical framework

In this subchapter, the analytical framework is summarised.
Considering the spaceships: travelling to each of the planets has a cost in terms of crystals, known to the player, that depends on the planet on which it is travelling; The uncertainty of the context concerns the number of crystals present on each planet, unknown value for the spaceships:

- In the contextual setting, the only player is able to estimate the probability distribution of the planet.
- In the adversarial setting, it is impossible to estimate the probability distribution of the planet, because the other player creates an "interference".

The only context information the spaceships know is only and exclusively the number of crystals he owns. Considering the table 3.16, only the variables 'crystals loaded' and 'crystals owned' are known.

Finally, the lasts relevant aspects to underline concerns:

- The spaceships' performance: they are evaluated based on the number of crystals possessed at each iteration.

- The game takes place over a time horizon of 10,000 iterations. The trends of the number of crystals possessed throughout the game will be compared to the various experimental strategies and settings.

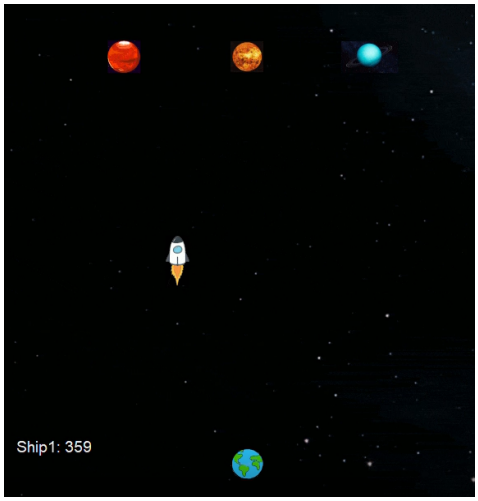

Figure 3.4: Spaceship travelling to Mars

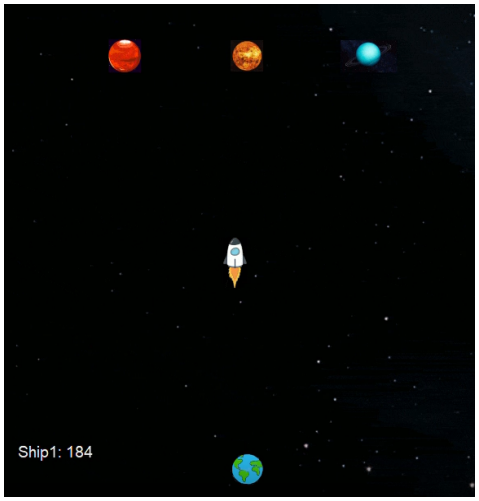

Figure 3.5: Spaceship travelling to Venus

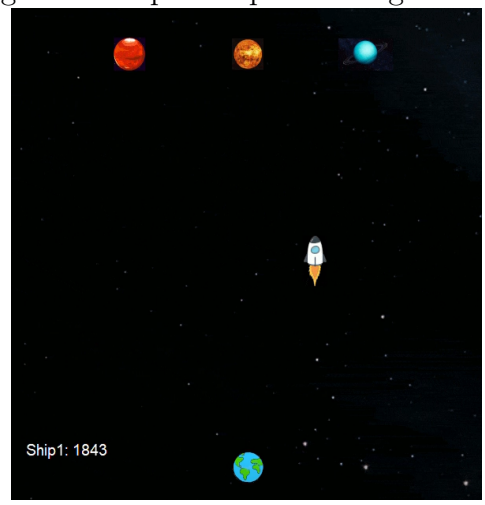

Figure 3.6: Spaceship travelling to Uranus

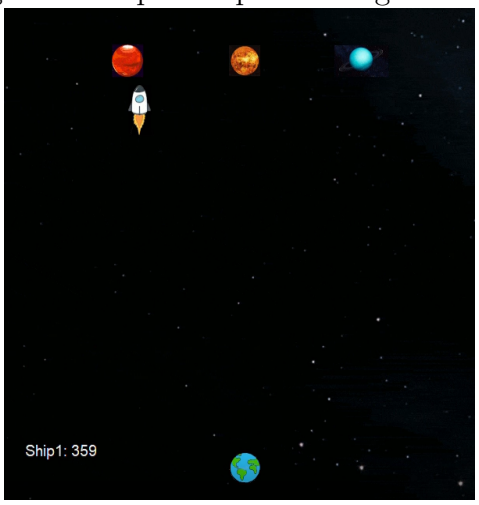

Figure 3.7: Spaceship on Mars

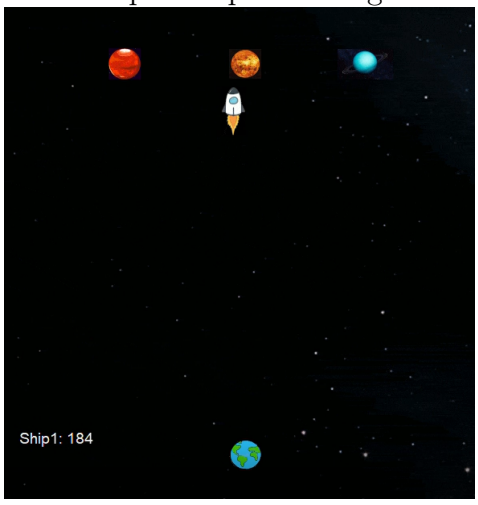

Figure 3.8: Spaceship on Venus

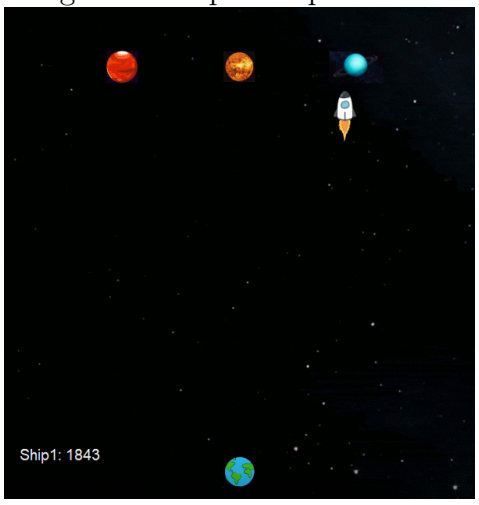

Figure 3.9: Spaceship on Uranus

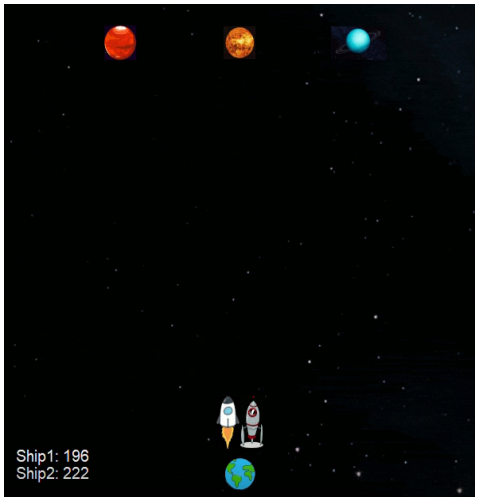

Figure 3.10: Spaceships at home

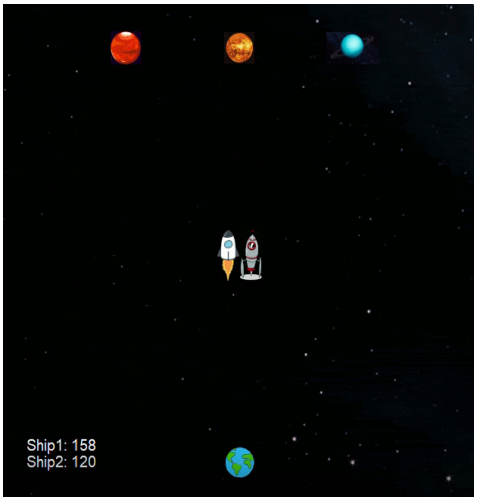

Figure 3.11: Spaceships travelling

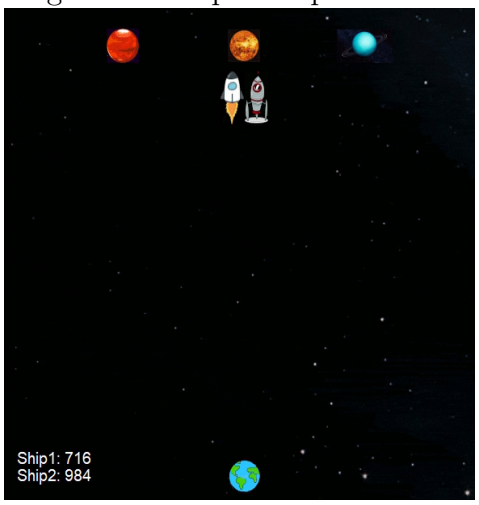

Figure 3.12: Spaceships on destination planet

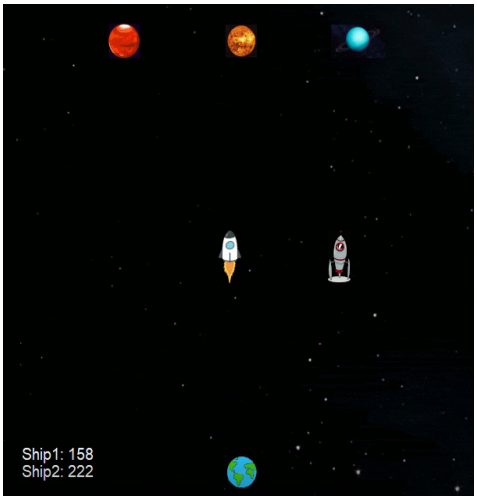

Figure 3.13: Spaceships travelling

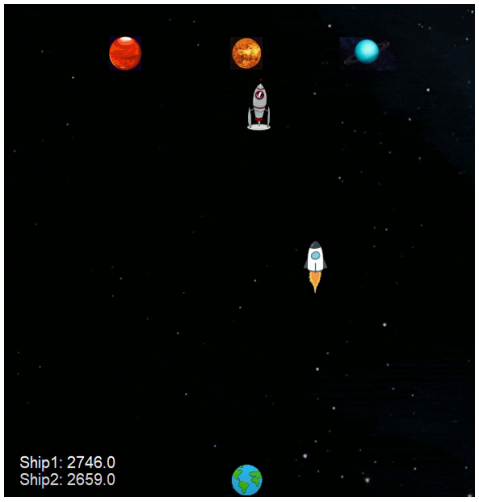

Figure 3.14: One spaceship travelling and the other on destination planet

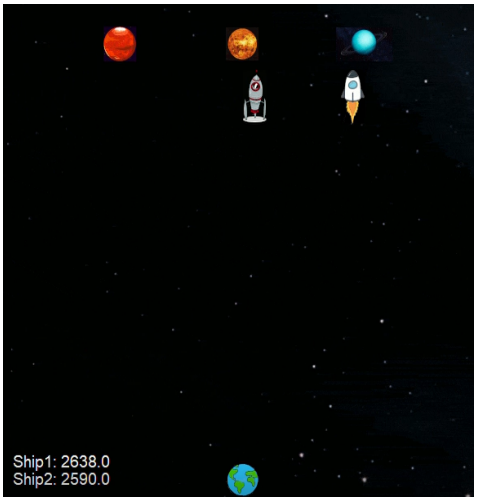

Figure 3.15: Spaceships on the destination planets

t	Ship_name	State	destination	AvailableCrystals	AvailableCrystals Marte	AvailableCrystals Venere	AvailableCrystals Urano	crystal loaded	crystals owned
1	Enterprise	ready	Home	[10, 10, 10]	10	10	10	0	100
1	Millennium Falcon	ready	Home	[10, 10, 10]	10	10	10	0	100
2	Enterprise	travelling	Urano	[21, 18, 18]	21	18	18	0	92
2	Millennium Falcon	travelling	Marte	[21, 18, 18]	21	18	18	0	96
3	Enterprise	travelling	Urano	[31, 29, 29]	31	29	29	0	92
3	Millennium Falcon	operating	Marte	[31, 29, 29]	31	29	29	31	96
4	Enterprise	travelling	Urano	[12, 44, 47]	12	44	47	0	92
4	Millennium Falcon	travelling	Home	[12, 44, 47]	12	44	47	31	96
5	Enterprise	operating	Urano	[21, 56, 59]	21	56	59	59	92
5	Millennium Falcon	operating	Home	[21, 56, 59]	21	56	59	0	96
6	Enterprise	travelling	Home	[30, 69, 12]	30	69	12	59	92
6	Millennium Falcon	ready	Home	[30, 69, 12]	30	69	12	0	127
7	Enterprise	travelling	Home	[40, 82, 28]	40	82	28	59	92
7	Millennium Falcon	travelling	Venere	[40, 82, 28]	40	82	28	0	121
8	Enterprise	travelling	Home	[46, 95, 42]	46	95	42	59	92
8	Millennium Falcon	travelling	Venere	[46, 95, 42]	46	95	42	0	121
9	Enterprise	operating	Home	[56, 104, 57]	56	104	57	0	92
9	Millennium Falcon	operating	Venere	[56, 104, 57]	56	104	57	104	121
10	Enterprise	ready	Home	[65, 7, 66]	65	7	66	0	151
10	Millennium Falcon	travelling	Home	[65, 7, 66]	65	7	66	104	121

Figure 3.16: Example of the dataframe

Chapter 4

Algorithms used

In this chapter, all the Algorithms used in the study will be analyzed, and the generic formulas are presented. Below a list of the used algorithms will follow:

- Epsilon-Greedy Algorithm
- Mean Algorithm
- UCB – The Upper Confidence Bound Algorithm
- Time elapsed based MAB Algorithm
- Thompson Sampling Algorithm
- Thompson Sampling with Probability Matching Algorithm

4.1 Epsilon-Greedy Algorithm

The Epsilon-Greedy Algorithm is the most straightforward Algorithm used in the multi-armed bandit problem environment. The policy of the $\epsilon - greedy$ Algorithm is to pick the greedy action. It means that it chooses the arm that gave the best reward until that time (exploitation) with probability $1 - \epsilon$. It explores the environment (exploration) with

probability ϵ, choosing the arm to pull with probability $1/n$. In this case, the Algorithm explores unknown or less profitable arms. In Figure 4.1 the workflow of the $\epsilon - greedy$ algorithm is shown.

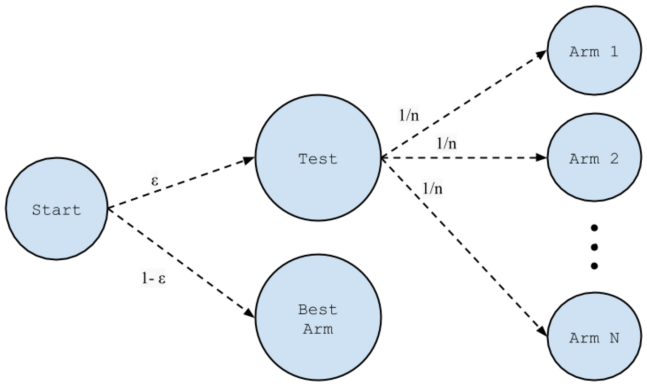

Figure 4.1: E-greedy algorithm workflow. In this game, the planet is represented by the 'arm'

In the current study, the $\epsilon - greedy$ Algorithm is not used as a standalone algorithm, but as an implementation in each algorithm. This means that each algorithm has two versions: the version with and without $\epsilon - greedy$. In the firstly mentioned version, the ϵ is set equal to 0.15.

4.2 Mean Algorithm

The first Algorithm used in this study is the Mean Algorithm. The concept behind this Algorithm is straightforward: it bases its principle on the planet's choice that, on average, has granted the most significant number of crystals to the spaceship. The initial phase of the game focuses on exploration because it must visit all the planets before starting the exploitation phase. The mean of the chosen planet is at each round of the game updated. This allows the Algorithm to change the destination planet whenever the usual destination planet's mean becomes lower than the other planets' means. This will be explained by an

example: crystal production for the game planets is updated at each iteration, it is not updated at every round. In this example, crystal production for the planets is updated at each round for simplification purposes. The crystal production of the planets Mars, Uranus and Venus are respectively 5, 6 and 7. Let us say that during the exploration phase, the spaceship mined 10, 12 and 21 crystals. At this point, the exploitation phase starts. According to the Mean Algorithm, the spaceship chooses Venus as destination planet, because it is the planet where the spaceship on average extracted the most crystals. At the second round, it can mine 7 crystals. According to the mean formula, the mean of crystals mined is 14. The destination planet is still Venus. At the third round, it can mine 7 crystals again; now the mean is 11,7. Now it is possible to see that the highest mean is of Uranus: with an average of 12 crystals. At this point, the destination planet changes.
This strategy is purely "exploitative": it always chooses the planet for which the average of the crystals collected is high. The epsilon-greedy strategy allows to add an exploration component, because with an epsilon probability a random planet is chosen, allowing to "learn" something more for the other planets as well.

Below the generic Mean Algorithm formula 4.1 is shown. μ_G^{MA} is the mean of crystals collected so far on G (planets).

$$P^* = argmax_{P \in p}(\mu_P^{MA}) \tag{4.1}$$

4.3 UCB – The Upper Confidence Bound Algorithm

The second algorithm used in this study is The Upper Confidence Bound (UCB). This algorithm is one of the most used in the Multi-Armed Bandit problem. This because of its conceptual simplicity and strong theoretical guarantees. UCB algorithms base their strong guarantees on their upper confidence bound, which is the expected reward amplified based on the level of uncertainty of the estimation. An optimistic behaviour drives UCB algorithms because they choose the arm to pull with the highest upper confidence bound.

The upper confidence bound represents uncertainty. This is also called the *Optimism in Face of Uncertainty* principle.

In the UCB algorithm used in this study, the upper bound is represented by the std dev. This is because the planets that have been chosen often have a 'more consolidated' mean and std dev. The std dev of the planets chosen few times is much higher because they are not as well known as the other planets. As mentioned before, the UCB algorithm is *Optimistic in Face of Uncertainty*; for this reason, it will tend to favour planets with higher uncertainty (std dev). Based on Figure 4.2, the UCB algorithm will be explained with an example. Firstly, some assumptions need to be considered: the orange lines are the means of the crystals carried by the spaceship from the planets and the upper bounds as the std dev of the same variable. The planets Mars, Uranus and Venus are respectively represented by A, B, and C.

Figure 4.2 shows the planets' situation when the game has already started. It is possible to see that Uranus has been chosen more times than the other planets because its confidence interval is really small. If the Mean algorithm was applied, which is a more conservative algorithm, Uranus would have been chosen as the destination planet because it has a higher mean. Nevertheless, considering the UCB algorithm that considers uncertainty as an opportunity, it chooses Mars as a destination planet. With a risk of adverse thinking, the spaceship can gain much fewer crystals, but it can also gain the highest number of crystals.

Also in this Algorithm, the mean and standard deviation values are updated after each decision and subsequent observation of the number of crystals collected. The average represents the exploitation component, the dev std represents the exploration component and β serves to balance the trade-off between the two components

The generic UCB formula 4.2 is shown. μ_G^{UCB} and σ_G^{UCB} are the mean and std dev of crystals collected so far on G (planets).

$$P^* = argmax_{P \in p}(\mu_P^{UCB} + \beta\sigma_P^{UCB}) \tag{4.2}$$

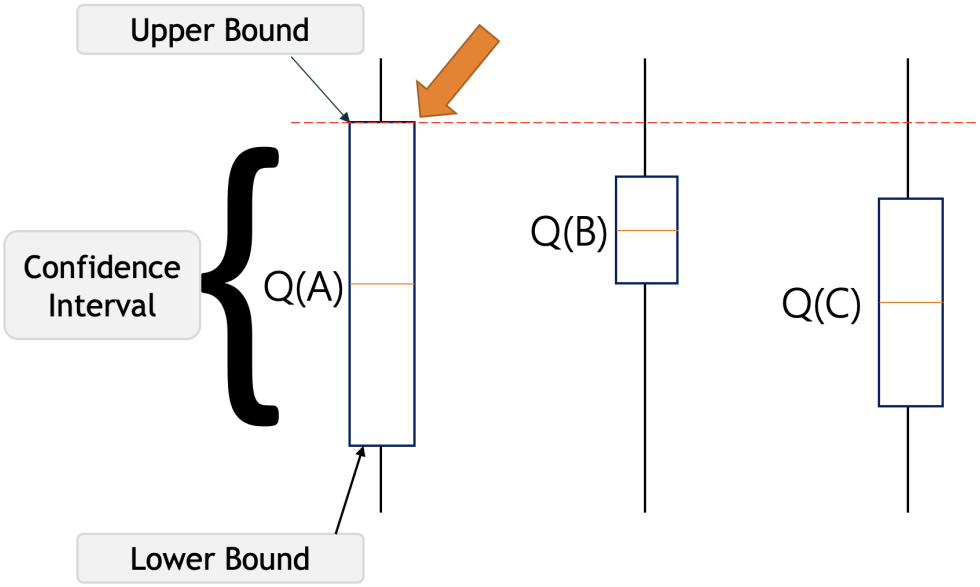

Figure 4.2: UCB Upper Confidence Bound

4.4 Time elapsed based MAB Algorithm

The Time elapsed based MAB Algorithm also belongs to the UCB Algorithms family. This variation was designed for the recommendation systems. Being of the same family as the UCB Algorithm previously explained, its characteristics are the same. The Time elapsed based MAB Algorithm formula is the 4.3. It is similar to the formula 4.2, the only difference is the replacement of the std dev with the $\sqrt{\frac{2ln(t)}{T_P}}$. It represents the square root of the current iteration divided by the number of times the spaceship has been on that planet. The more times the spaceship has been on the planet in question, the less bonus it adds to the mean of crystals mined from it. On the other hand, it has been a long time since the last time the planet was mined, or the spaceship went few times to that planet, the uncertainty bonus added to the mean increases.

For the Time elapsed based MAB Algorithm, the average is the exploitation component (as in UCB) while the exploration component, in this case, is linked to the time elapsed since the last time a specific decision was made. The more time passes the more uncertainty

increases. This is the main difference from UCB.

$$P^* = argmax_{P \in p}(\mu_P^{MAB} + \sqrt{\frac{2ln(t)}{T_P}}) \tag{4.3}$$

4.5 Thompson Sampling Algorithm

The late algorithm used is the Thompson Sampling, also called 'posterior sampling'. The algorithms used until this moment base their choice on the current mean of the rewards received from choosing one planet or another. Thompson Sampling Algorithm has another approach: it creates a probability model from the rewards until that time obtained, and afterwards, it samples from these rewards to choose the destination planet. In this way, not as it were is a progressively exact estimate of the possible reward gotten, but the algorithm also gives a level of confidence in this reward, and this confidence increments proportionally with the samples collected. This mechanism to update the believes as more data are available is called *Bayesian Inference*. For this reason, the Bayesian assumption is added. Considering an example of the game analyzed in this study: A spaceship has to collect as many crystals as possible, considering three planets from which it can mine. The spaceship does not know which planet has the most crystals. A binary output is chosen to simplify the example: 0 (that planet is not the one with the most crystals available) and 1 (that planet is the one with the most crystals available). Because of the binary output, its behaviour can be represented by the Bernoulli distribution. What the spaceship is searching is the planet with the highest probability of containing the most crystals. As explained before, the Thompson Sampling Algorithm creates a model of the reward probabilities. In this case, the Beta distribution is perfect for modelling this probability case. In the Beta distribution, there are two parameters: α and β. The formula used is:

$$\frac{\alpha}{\alpha + \beta} = \frac{number\ \ of\ \ success}{total\ \ number\ \ of\ \ trials} \tag{4.4}$$

In figure 4.3 the Beta distribution considering different α and β is shown. Initially, the

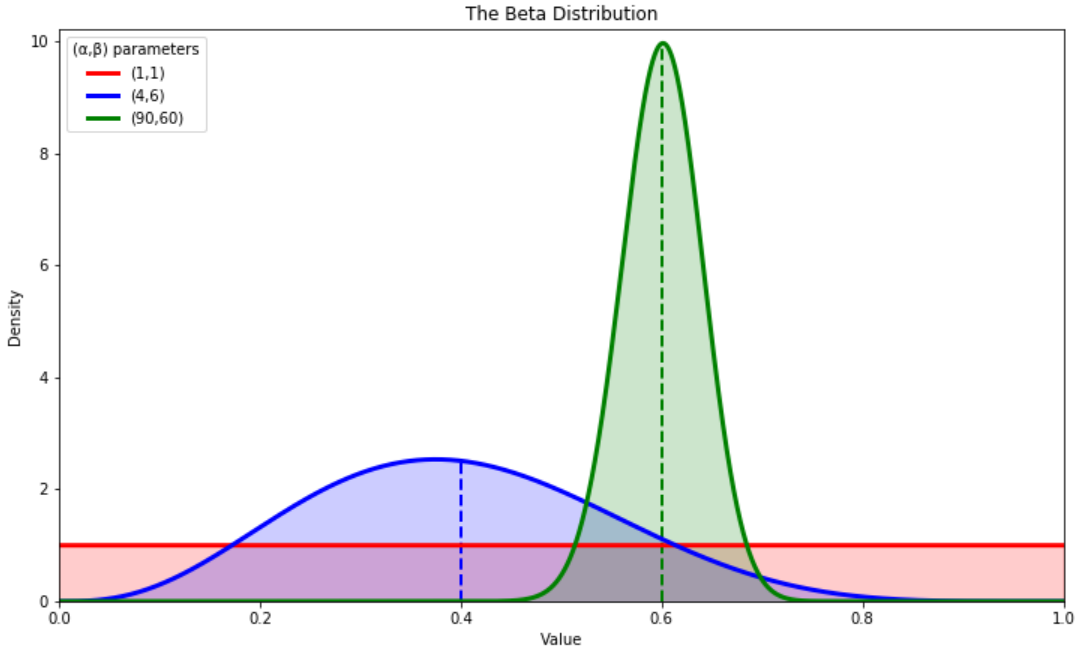

Figure 4.3: Beta distribution for different values of alpha and beta

probablility of finding the planet with the most crystals in it is unknown, so α and β are set to 1, that create the red line (uniform distribution). This initial guess is called *prior probability*, which represents the probability of an event's occurrence before having data about it. It is represented by β. After the prior probability, some evidence is collected because of the first attempt and α and β are updated. At this point, α is updated with a $+1$ if the planet chosen is the best one and it remains unchanged if the planet chosen is not the best one. The number of failures β is updated oppositely: $+1$ if the planet chosen is not the best one and $+0$ if the planet chose is the best one. After the prior probability, the *posterior probability* take place. It represents the new probability, based on new data. After more and more attempts, the curve becomes more accurate, passing through the blue curve ending up at the green curve. At this point, the Thompson Sampling algorithm

finds an estimated mean reward and confidence level in this estimate. As is it possible to see in the blue curve in figure 4.3, the planets that have been chosen a few times have broad distributions. This means that they have a more extensive range of possible values to be chosen than the green curve because the green curve, as mentioned before, is more accurate and has a narrow curve.

Compared to the mean and UCB Algorithm that chooses the destination planet based on the highest estimated reward, the Thompson Sampling algorithm samples from the Beta distribution and chooses the destination planet based on its highest return value. It gives more chance to be also chosen to actions that have been tried infrequently. In this way, the planets with a low estimated mean reward and that have been chosen a few times (blue curve) can return a more extensive sample value and become the selected planet at the actual round.

At this point already, it is possible to understand that Thompson sampling considers the exploration phase more than the mean and UCB algorithms. This because the last algorithms cited tend to choose and continue with the green curve since the first moment. Instead, Thompson sampling considers the full width of the curve; this benefits the blue curve. After more and more rounds, the confidence in the estimated mean increases. This means that all the curves become much broader and become closer to the true mean. In this phase of the Algorithm, the exploitation phase becomes more prominent since the planet with a higher probability of having the most crystals will be the default choice.

The Thompson Sampling is not an optimal algorithm because the planet's curve that becomes the default choice will be close to the true mean, but the true mean of the other planets' curve may never be found. [4]

The "sampling" part represents the exploration component; the exploitation is instead incorporated in the mean and variance values, which are always updated after each observation.

In the current study, number 4.5 shows the formula applied. μ_G^{TS} and σ_G^{TS} are the mean and std dev of crystals collected so far on G (planets).

$$P^* = argmax_{P \in p} N(\mu_P^{TS}, \sigma_P^{2\,TS}) \tag{4.5}$$

4.6 Thompson Sampling Algorithm with Probability Matching

Finally, the Thompson Sampling Algorithm with Probability Matching is used. This is a variation of the previous Algorithm. The Probability Matching part consists of predicting the distribution based upon the rewards gained until that time. It randomly allocates observations to arms according to the Bayesian posterior probability that each arm is optimal. [5] In the current study, number 4.6 shows the formula applied. Where P indicates the probability density function calculated for g and conditioned on all the observations made up in the Thompson Sampling Algorithm and denoted by D_g

$$P^* = argmax_{P \in p} \mathbb{P}(g|D_g) \tag{4.6}$$

Chapter 5

Code explanation and application of the Algorithms

In the previous chapter, all the algorithms used in this study have been analyzed. In this chapter, the algorithms' applications will be presented, and the pseudocode will be discussed. As explained in chapter 3, the algorithms explained below follow the frameworks of figures 3.1 and 3.2. The methods used in this study are:

- Mean Algorithm
- UCB – The Upper Confidence Bound Algorithm
- Time elapsed based MAB Algorithm
- Thompson Sampling Algorithm
- Thompson Sampling with Probability Matching Algorithm

All the algorithms explained below are inserted in the 'Choose i*' blocks of figures 3.1 and 3.2. It depends if there is the 1 spaceship variation or the 2 spaceships variation.

5.0.1 Variables

Some variables are used in different algorithms. All the variables will be explained below in order to avoid the repetition of them for every algorithm. The variables explained below are composed of a list of 3 elements. Each element represents one planet. The planets follow the order: Mars, Venus, Uranus. They are updated at each round.

- Numbers of selection: each element in the list shows the number of times a planet was chosen. In formulas, it will be represented by n
- Crystals mined: each element in the list shows the number of crystals mined from each planet until that moment.
- New list: it is composed of 3 elements: 0, 1 and 2. Each element represents one planet. The order of the planets is the same as the variables before. The logic behind the creation of this variable is that at the beginning of the game, it is important to explore all the planets before starting with the exploitation phase. For this reason, at each round the destination planet is deleted from the list, so that the spaceship can't go on that planet until it explores all the other planets.
- List of crystals carried: each element in the list shows the history of crystals carried from the ship for each planet.
- List of crystals carried: it is composed of a list of 3 elements. Each element represents one planet. The planets follow the order: Mars, Venus, Uranus. Each element in the list shows the history of crystals carried from the ship for each planet. They are updated at each round.

5.1 Mean Algorithm

The first Algorithm used in the study is the easiest one. First, different variables are set. The variables used are:

- Numbers of selection
- Crystals mined
- New list
- Mean available crystals list: each element in the list shows the mean of crystals mined until that moment t from each planet. The calculation of each element in the list is represented in formula 5.1:

$$\mu_t = \frac{\mu_{t-1} * (n-1) + (q_{t-1})}{n} \tag{5.1}$$

Figure 5.1 shows the pseudocode of the algorithm without the Epsilon-Greedy strategy. Figure 5.2 shows the pseudocode of the algorithm with the Epsilon-Greedy strategy.

```
SET:
numbers_of_selection = [0,0,0] (list containing the number of times a planet is chosen)
crystals_mined = [0,0,0] (list containing the quantity of crystals mined from each planet)
mean_available_crystals_list = [0,0,0] (list containing the average of the mined crystals compared to the times the planet was selected)
new_list = [0,1,2] (list of planets)

def mean_algorithm ():
    if the ship didn't go to all the planets:
        if t <= 2:
            mean_available_crystals_list = mean_available_crystals_list
        else:
            update numbers_of_selection, crystals_mined, mean_available_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    else: # go to the planet with the highest mean
        restore new_list
        update numbers_of_selection, crystals_mined, mean_available_crystals_list
        self.destination = pick the planet with the highiest mean
        return destination
```

Figure 5.1: Mean algorithm strategy

In figures 5.1 and 5.2 the pseudocodes are shown. In the initial phase of the Algorithm, the variables are set. Afterwards, the first if statement allows the spaceship to go to all the planets and it updates the variables at each iteration. In the second phase, if the $\epsilon - greedy$ strategy is presented in the Algorithm (figure 5.2), the spaceship visits a random planet with probability ϵ. Otherwise the Algorithm, in both versions, follows the third and last

```
SET:
numbers_of_selection = [0,0,0] (list containing the number of times a planet is chosen)
crystals_mined = [0,0,0] (list containing the quantity of crystals mined from each planet)
mean_available_crystals_list = [0,0,0] (list containing the average of the mined crystals compared to the times the planet was selected)
new_list = [0,1,2] (list of planets)

def mean_algorithm:
    epsilon  = 0.15
    explore = pick 'explore' with 'epsilon'% of probability
    if the ship didn't go to all the planets:
        if t <= 2:
            mean_available_crystals_list = mean_available_crystals_list
        else:
            update numbers_of_selection, crystals_mined, mean_available_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    elif explore == 1:
        restore new_list
        update numbers_of_selection, crystals_mined, mean_available_crystals_list
        destination = pick a random planet
        return destination
    else:
        restore new_list
        update numbers_of_selection, crystals_mined, mean_available_crystals_list
        self.destination = pick the planet with the highiest mean
        return destination
```

Figure 5.2: Mean + Epsilon-Greedy algorithm pseudocode

phase. As explained in chapter 4, it chooses the planet with the highest mean. This Algorithm is repeated until the game finishes.

5.2 UCB – The Upper Confidence Bound Algorithm

The second method used in the study is similar to the Mean algorithm. Also for this algorithm, firstly different variables are set. The variables are:

- Numbers of selection
- Crystals mined
- List of crystals carried: In the formula 5.2 it will be represented by z
- New list
- Mean std available crystals list: each element in the list shows the mean + standard deviation of crystals carried until that moment t from each planet. The calculation

of each element in the list is represented in the formula 5.2.

$$\mu_t \sigma_t = \mu_{t-1} + \sigma(z) \tag{5.2}$$

In the formula 5.2 μ_{t-1} is calculated by the formula 5.1.
Figure 5.3 shows the pseudocode of the algorithm without the Epsilon-Greedy strategy. Figure 5.4 shows the pseudocode of the algorithm with the Epsilon-Greedy strategy.

```
SET:
numbers_of_selection = [0,0,0] (list containing the number of times a planet is chosen)
crystals_mined = [0,0,0] (list containing the quantity of crystals mined from each planet)
list_of_crystals_carried = [0,0,0] (list containing the history of crystals carried from the ship for each planet)
mean_std_available_crystals_list = [0,0,0] (list containing the mean + std of the mined crystals compared to the times the planet was selected)
new_list = [0,1,2] (list of planets)

def UCB:
    if the ship didn't go to all the planets:
        if t <= 2:
            mean_std_available_crystals_list = mean_std_available_crystals_list
        else:
            update numbers_of_selection, crystals_mined, list_of_crystals_carried, mean_std_available_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    else:
        restore new_list
        update numbers_of_selection, crystals_mined, list_of_crystals_carried, mean_std_available_crystals_list
        destination = pick the planet with the highiest mean + std
        return destination
```

Figure 5.3: UCB algorithm pseudocode

In figures 5.3 and 5.4 the pseudocode is shown. In the initial phase of the Algorithm, the variables are set. Afterwards, the first 'if statement' allows the spaceship to go to all the planets and it updates the variables at each iteration. In the second phase, if the $\epsilon - greedy$ strategy is presented in the Algorithm (figure 5.4), the spaceship visits a random planet with probability ϵ. Otherwise, the Algorithm, in both versions, follows the third and last phase. As explained in chapter 4, it chooses the planet with the highest mean + std dev value. This Algorithm is repeated until the game finishes.

```
SET:
numbers_of_selection = [0,0,0] (list containing the number of times a planet is chosen)
crystals_mined = [0,0,0] (list containing the quantity of crystals mined from each planet)
list_of_crystals_carried = [0,0,0] (list containing the history of crystals carried from the ship for each planet)
mean_std_available_crystals_list = [0,0,0] (list containing the mean + std of the mined crystals compared to the times the planet was selected)
new_list = [0,1,2] (list of planets)

def UCB:
    epsilon  = 0.15
    explore = pick 'explore' with 'epsilon'% of probability
    if the ship didn't go to all the planets:
        if t <= 2:
            mean_std_available_crystals_list = mean_std_available_crystals_list
        else:
            update numbers_of_selection, crystals_mined, list_of_crystals_carried, mean_std_available_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    elif explore == 1:
        restore new_list
        update numbers_of_selection, crystals_mined, list_of_crystals_carried, mean_std_available_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    else:
        restore new_list
        update numbers_of_selection, crystals_mined, list_of_crystals_carried, mean_std_available_crystals_list
        destination = pick the planet with the highiest mean + std
        return destination
```

Figure 5.4: UCB + Epsilon-Greedy algorithm pseudocode

5.3 Time elapsed based MAB Algorithm

The third method used in the study is the Time elapsed based MAB Algorithm. Also for this algorithm, first different variables are set. The variables are:

- Numbers of selection
- Crystals mined
- New list
- Mab tempo crystals list: each element in the list shows the formula calculation for each planet. The formula applied is 5.3.

$$G^* = argmax_{P \in p}(\mu_P^{MAB} + \sqrt{\frac{2ln(t)}{T_P}}) \tag{5.3}$$

In the formula 5.3 μ_G^{MAB} is calculated by the formula 5.1. t is the current iteration and T_G is the number of times that planet was chosen until that t, represented by *Numbers of selection* variable.

Figure 5.5 shows the pseudocode of the algorithm without the Epsilon-Greedy strategy. Figure 5.6 shows the pseudocode of the algorithm with the Epsilon-Greedy strategy.

```
SET:
numbers_of_selection = [0,0,0] (list containing the number of times a planet is chosen)
crystals_mined = [0,0,0] (list containing the quantity of crystals mined from each planet)
mab_tempo_crystals_list = [0,0,0] (list containing the values calculated by the mab tempo formula)
new_list = [0,1,2] (list of planets)

def mab_tempo(self):
    if the ship didn't go to all the planets:
        if game.t <= 2:
            mab_crystals_list = mab_crystals_list
        else:
            update numbers_of_selection, crystals_mined, list_of_crystals_carried, mab_tempo_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    else:
        restore new_list
        update numbers_of_selection, crystals_mined, list_of_crystals_carried, mab_tempo_crystals_list
        destination = pick the planet with the highiest value of mab_tempo_crystals_list
        return destination
```

Figure 5.5: Time elapsed based MAB Algorithm pseudocode

In Figure 5.5 and 5.6 the pseudocode is shown. In the initial phase of the Algorithm, the variables are set. Afterwards, the first if statement allows the spaceship to go to all the planets and it updates the variables at each iteration. In the second phase, if the $\epsilon - greedy$ strategy is presented in the Algorithm (figure 5.6), the spaceship visits a random planet with probability ϵ. Otherwise, the Algorithm, in both versions, follows the third and last phase. As explained in chapter 4, it chooses the planet with the highest value of the formula 5.3. This Algorithm is repeated until the game finishes.

```
SET:
numbers_of_selection = [0,0,0] (list containing the number of times a planet is chosen)
crystals_mined = [0,0,0] (list containing the quantity of crystals mined from each planet)
mab_tempo_crystals_list = [0,0,0] (list containing the values calculated by the mab tempo formula)
new_list = [0,1,2] (list of planets)

def mab_tempo(self):
    if the ship didn't go to all the planets:
        if game.t <= 2:
            mab_crystals_list = mab_crystals_list
        else:
            update numbers_of_selection, crystals_mined, list_of_crystals_carried, mab_tempo_crystals_list
        destination = pick a random planet
        delete the destination planet from new_list
        return destination
    else:
        restore new_list
        update numbers_of_selection, crystals_mined, list_of_crystals_carried, mab_tempo_crystals_list
        destination = pick the planet with the highiest value of mab_tempo_crystals_list
        return destination
```

Figure 5.6: Time elapsed based MAB + Epsilon-Greedy algorithm pseudocode

5.4 Thompson Sampling Algorithm

For the last two algorithms the process is different: For this algorithm, just one variable is set:

- List of crystals carried

The formula applied at the Thompson Sampling Algorithm is the 5.4.

$$P^* = argmax_{P \in p} N(\mu_P^{TS}, {\sigma_P^2}^{TS}) \tag{5.4}$$

In the formula 5.4: μ_P^{TS} and ${\sigma_P^2}^{TS}$ are the mean and std dev of 30 values sampled by the *List of crystals carried* variable. In this game study, the Thompson Sampling Algorithm is applied in this way: firstly 120 rounds are played by selecting a random planet destination. In the second phase, in the variation with the Epsilon-Greedy strategy, the algorithm chooses a random planet with probability ϵ, otherwise it takes a sample of 30 values of the mean and std dev of every planet, based on the List of crystals carried. Between these 180 values (90 values of mean and 90 of std dev), the destination planet is chosen considering

the planet with the highest value.
Figure 5.7 shows the pseudocode of the algorithm without the Epsilon-Greedy strategy. Figure 5.8 shows the pseudocode of the algorithm with the Epsilon-Greedy strategy.

```
SET:
list_of_crystals_carried = [0,0,0] (list containing the history of crystals carried from the ship for each planet)

def UCB:
    if game round <= 120:
        if t <= 2:
            list_of_crystals_carried = list_of_crystals_carried
        else:
            update list_of_crystals_carried
        destination = pick a random planet
        return destination
    else:
        update list_of_crystals_carried
        destination = pick the planet with the highiest value between mean or std
        return destination
```

Figure 5.7: Thompson Sampling algorithm pseudocode

```
SET:
list_of_crystals_carried = [0,0,0] (list containing the history of crystals carried from the ship for each planet)

def UCB:
    epsilon  = 0.15
    explore = pick 'explore' with 'epsilon'% of probability
    if game round <= 120:
        if t <= 2:
            list_of_crystals_carried = list_of_crystals_carried
        else:
            update list_of_crystals_carried
        destination = pick a random planet
        return destination
    elif explore == 1:
        update list_of_crystals_carried
        destination = pick a random planet
        return destination
    else:
        update list_of_crystals_carried
        destination = pick the planet with the highiest value between mean or std
        return destination
```

Figure 5.8: Thompson Sampling + Epsilon-Greedy algorithm pseudocode

In figures 5.7 and 5.8 the pseudocode is shown. In the initial phase of the Algorithm, the variables are set. Afterwards, the first if statement allows the spaceship to go to all

the planets and it updates the variables at each iteration. In the second phase, if the $\epsilon - greedy$ strategy is presented in the Algorithm (figure 5.8), the spaceship visit a random planet with probability ϵ. Otherwise, the Algorithm, in both versions, follows the third and last phase. As explained in chapter 4, it chooses the planet with the highest value of the formula 5.4. This Algorithm is repeated until the game finishes.

5.5 Thompson Sampling Algorithm with Probability Matching

For the Thompson Sampling Algorithm with Probability Matching, just one variable is set:

- List of crystals carried

The formula applied at the Thompson Sampling Algorithm with Probability Matching is the 5.5.

$$P^* = argmax_{P \in p} \mathbb{P}(g|D_g) \tag{5.5}$$

In the formula 5.5: μ_P^{TS} and $\sigma_P^{2\,TS}$ are the mean and std dev of 30 values sampled by the creation of distributions based on the mean and std dev of *List of crystals carried* variable. In this game study, the Thompson Sampling Algorithm is applied in this way: firstly 120 rounds are played by selecting a random planet destination. In the second phase, in the variation with the Epsilon-Greedy, the algorithm chooses a random planet with probability ϵ. Otherwise, it creates one distribution per planet based on the mean and std dev of the List of crystals carried. Furthermore, it selects 100 values per distribution. Finally, the algorithm takes a sample of 30 values between these values. The destination planet is chosen considering the belonging of the highest value.
Figure 5.9 shows the pseudocode of the algorithm without the Epsilon-Greedy strategy. Figure 5.10 shows the pseudocode of the algorithm with the Epsilon-Greedy strategy.
Figures 5.9 and 5.10 the pseudocode is shown. In the initial phase of the Algorithm, the variables are set. Afterwards, the first if statement allows the spaceship to go to all the

```
SET:
list_of_crystals_carried = [0,0,0] (list containing the history of crystals carried from the ship for each planet)

def UCB:
    if game round <= 120:
        if t <= 2:
            list_of_crystals_carried = list_of_crystals_carried
        else:
            update list_of_crystals_carried
        destination = pick a random planet
        return destination
    else:
        update list_of_crystals_carried
        destination = pick the planet with the highiest value of the distributions
        return destination
```

Figure 5.9: Probability matching algorithm pseudocode

```
SET:
list_of_crystals_carried = [0,0,0] (list containing the history of crystals carried from the ship for each planet)

def UCB:
    epsilon  = 0.15
    explore = pick 'explore' with 'epsilon'% of probability
    if game round <= 120:
        if t <= 2:
            list_of_crystals_carried = list_of_crystals_carried
        else:
            update list_of_crystals_carried
        destination = pick a random planet
        return destination
    elif explore == 1:
        update list_of_crystals_carried
        destination = pick a random planet
        return destination
    else:
        update list_of_crystals_carried
        destination = pick the planet with the highiest value of the distributions
        return destination
```

Figure 5.10: Probability matching + Epsilon-Greedy algorithm pseudocode

planets and it updates the variables at each iteration. In the second phase, if the $\epsilon - greedy$ strategy is presented in the Algorithm (figure 5.10), the spaceship visits a random planet with probability ϵ. Otherwise the Algorithm, in both versions, follows the third and last phase. As explained in chapter 4, it chooses the planet with the highest value of the formula 5.5. This Algorithm is repeated until the game finishes.

Chapter 6

Results

In this chapter, the results of this study are summarized.
In all figures and tables presented in this chapter, the "mab tempo" represents the Time elapsed based MAB Algorithm.
It is important to remark that the measure considered to evaluate the performance of every MAB strategy, in the different settings, is the amount of crystals collected by spaceships along and at the end of the game. There are different interesting points to underline. In Figure 6.1 it is possible to see two charts, both related to the one-player gaming mode. The first chart reports the amount of crystals owned by spaceships over time for the first experimental setting, where the crystal production distribution of each planet is a Gaussian. The second chart reports the same information but for the second experimental setting, where different types of crystal production distributions are used. In both cases, the algorithms performs the same way. There are small variations: it is possible to see that in the second chart the Mean Algorithm with the $\epsilon - greedy$ strategy implementation performs at the same level of the best algorithms; and the UCB Algorithm with the $\epsilon - greedy$ strategy implementation performs a bit closer to the best algorithms. For this reason, I would say that for the algorithms there are no differences in terms of performances between being in the context in which the production of the crystals of the planets are of the same distribution or not. In Table 6.2 it is possible to see the final results of the algorithms. Results

are highlighted based on a color scale: the results of the algorithms that have performed very well are shown in green and then switch to yellow, orange and red for the algorithms that have performed worse.
From figure 6.1 it is possible to see that all the algorithms in the version with the $\epsilon-greedy$ strategy perform better than the algorithms without the $\epsilon-greedy$ strategy. Analysing the chart and the results in the Table 6.2, it is possible to think that this happens because the algorithms that use the $\epsilon-greedy$ strategy implementation have a better exploration phase. This aspect is also evident from which Algorithm perform better between the Algorithms version without the $\epsilon - greedy$ strategy implementation. As was told in chapter 4, the algorithms with the best exploration phase are the Thompson Sampling variations (with and without Probability Matching) firstly and then the Time elapsed based MAB Algorithm. Instead, the Mean and UCB Algorithm are already more focused on the exploitation phase at the beginning of the game. This is evident in Figure 6.1. The algorithms that perform best are the Thompson Sampling variations (with and without Probability Matching and with and without the $\epsilon - greedy$ strategy implementation) and the Time elapsed based MAB Algorithm with the $\epsilon - greedy$ strategy implementation. Then the Time elapsed based MAB Algorithm and at the end Mean and UCB Algorithm.

In figure 6.3 and in table 6.4 the results of the two spacecraft game variation are presented. In this case, figure 6.3 presents the results of the two different spaceships in different situations (Gaussian distribution for the production of crystals for all the three planets or three different distribution for the production of crystals for the three planets) per chart. The most interesting aspects to underline are:

- Also in this comparison, as in the comparison with the related to the two-players gaming mode (i.e., adversarial setting) , it is possible to see that there is no evidence of different results between considering Gaussian distribution for the production of crystals for all the three planets or three different distribution for the production of crystals for the three planets.

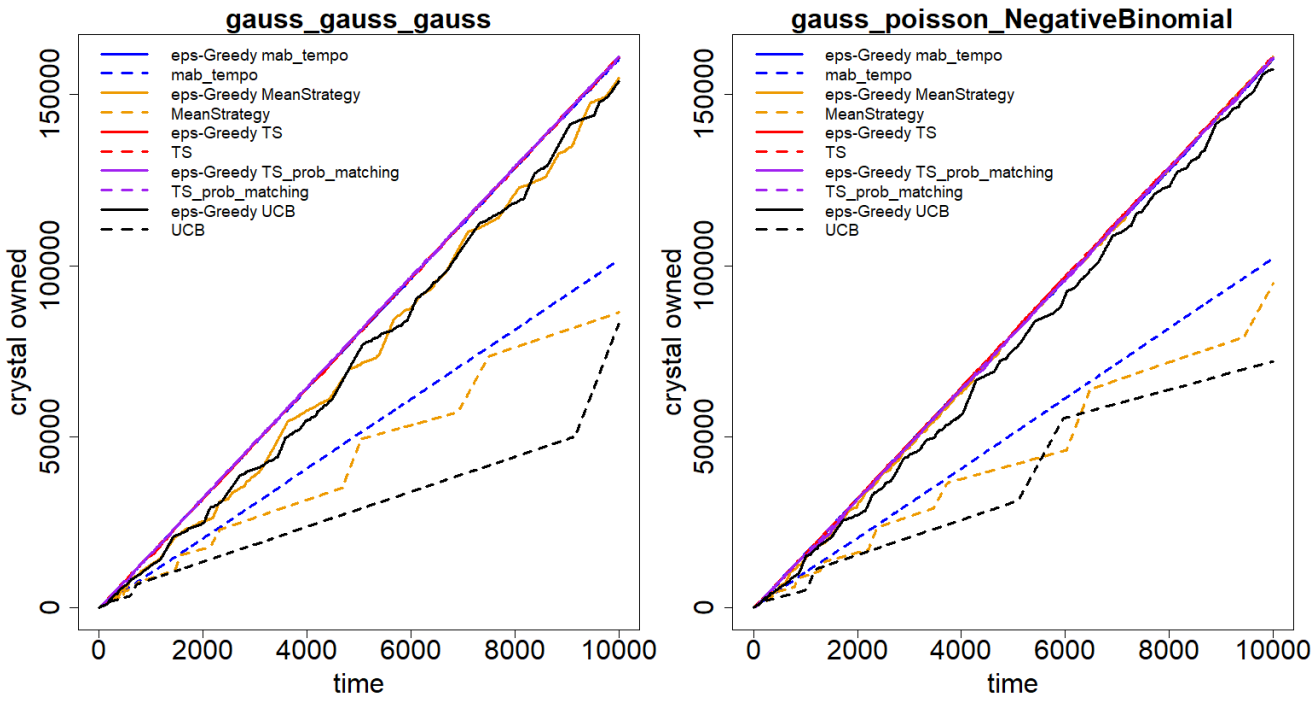

Figure 6.1: Results one spaceship

- According to the table 6.4, in some cases, a significant gap between the two spaceships can be observed although the strategies do not privilege one player over the other, regardless of whether the production of crystals on the planets is based on a Gaussian distribution for all the planets or different distributions. The Algorithms in which a significant gap between the two spaceships is present are:
  - Time elapsed based MAB Algorithm
  - Mean with the $\epsilon - greedy$ strategy implementation Algorithm
  - Thompson Sampling with the $\epsilon - greedy$ strategy implementation Algorithm
  - Thompson Sampling (partially)

gauss_gauss_gauss	eps-Greedy_mab_tempo	160566
gauss_gauss_gauss	mab_tempo	101583
gauss_poisson_NegativeBinomial	eps-Greedy_mab_tempo	160497
gauss_poisson_NegativeBinomial	mab_tempo	102154
gauss_gauss_gauss	eps-Greedy_MeanStrategy	154812
gauss_gauss_gauss	MeanStrategy	86333
gauss_poisson_NegativeBinomial	eps-Greedy_MeanStrategy	161197
gauss_poisson_NegativeBinomial	MeanStrategy	94794
gauss_gauss_gauss	eps-Greedy_TS	161038
gauss_gauss_gauss	TS	160788
gauss_poisson_NegativeBinomial	eps-Greedy_TS	160619
gauss_poisson_NegativeBinomial	TS	161136
gauss_gauss_gauss	eps-Greedy_TS_prob_matching	161186
gauss_gauss_gauss	TS_prob_matching	161018
gauss_poisson_NegativeBinomial	eps-Greedy_TS_prob_matching	160839
gauss_poisson_NegativeBinomial	TS_prob_matching	160760
gauss_gauss_gauss	eps-Greedy_UCB	153857
gauss_gauss_gauss	UCB	83008
gauss_poisson_NegativeBinomial	eps-Greedy_UCB	157476
gauss_poisson_NegativeBinomial	UCB	72019

Figure 6.2: Table one spaceship

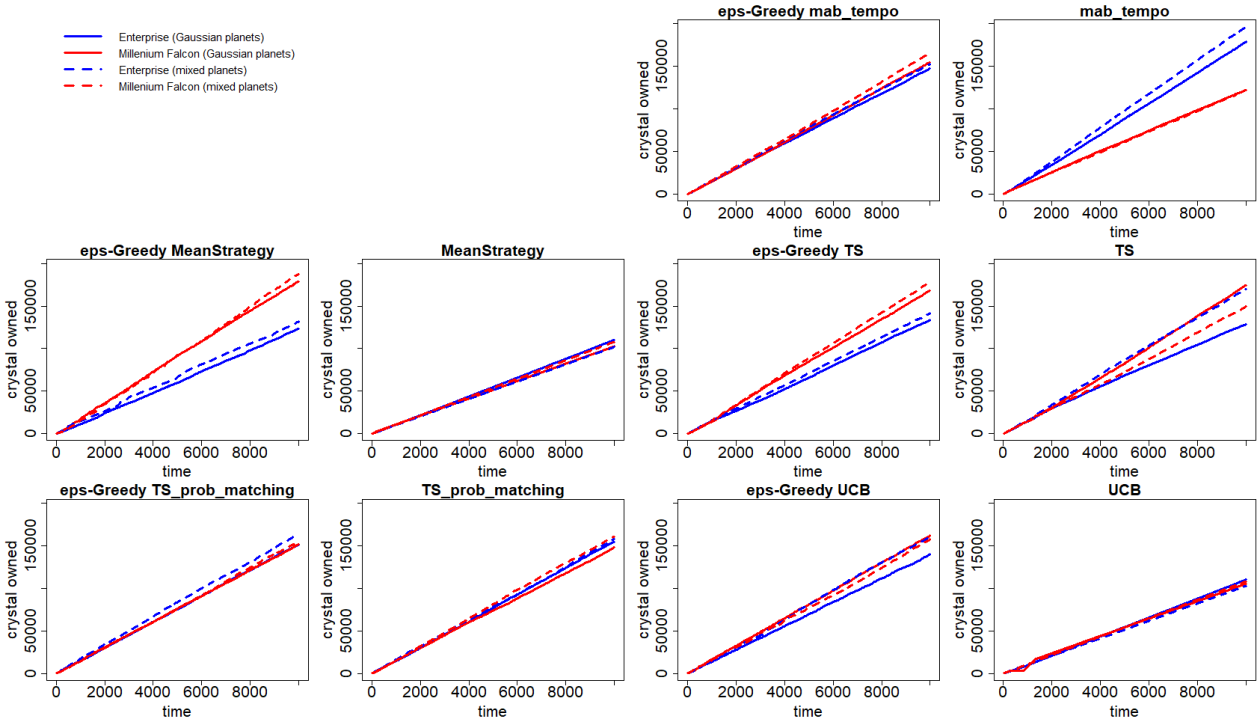

Figure 6.3: Results two spaceship

distributions	strategy	Enterprise's crystals	Millennium Falcon's crystals
gauss_gauss_gauss	eps-Greedy_mab_tempo	147900	154676
gauss_gauss_gauss	mab_tempo	178732	122252
gauss_poisson_NegativeBinomial	eps-Greedy_mab_tempo	152504	165228
gauss_poisson_NegativeBinomial	mab_tempo	196690	121924
gauss_gauss_gauss	eps-Greedy_MeanStrategy	123492	178986
gauss_gauss_gauss	MeanStrategy	110150	102598
gauss_poisson_NegativeBinomial	eps-Greedy_MeanStrategy	131608	187648
gauss_poisson_NegativeBinomial	MeanStrategy	101926	107914
gauss_gauss_gauss	eps-Greedy_TS	133422	168460
gauss_gauss_gauss	TS	128399	174533
gauss_poisson_NegativeBinomial	eps-Greedy_TS	141500	178178
gauss_poisson_NegativeBinomial	TS	169655	149275
gauss_gauss_gauss	eps-Greedy_TS_prob_matching	151264	151488
gauss_gauss_gauss	TS_prob_matching	154568	147944
gauss_poisson_NegativeBinomial	eps-Greedy_TS_prob_matching	163452	155382
gauss_poisson_NegativeBinomial	TS_prob_matching	158110	160746
gauss_gauss_gauss	eps-Greedy_UCB	140157	161745
gauss_gauss_gauss	UCB	110024	105305
gauss_poisson_NegativeBinomial	eps-Greedy_UCB	160704	157354
gauss_poisson_NegativeBinomial	UCB	102785	108049

Figure 6.4: Table two spaceships

Chapter 7

Conclusion

Finally, in this last chapter, the main point of this master thesis will be summarised. In chapter 3, the problem definition has been illustrated. It includes the design of the game, the development of the game, the graphic part and the data structure. In chapter 4 and 5, the theory and the implementation of the algorithms used are explained. Finally. in chapter 6, the results are shown.
The future studies correlate to this master thesis conclude the chapter.

7.1 Summary of the work

Let us go through throw this master thesis:

- In chapter 1, the study is introduced. The motivations and goals are explained. Then, Machine Learning and Reinforcement Learning are explained. This because the MAB problem derives from these two subjects. Afterwards, a MAB problem is explained, for simplification reasons the easiest version of it has been considered, the slot machine example. Finally, possible applications of the MAB problem are discussed.

- In chapter 2, the literature review has been considered, in which the paper that

inspired this master thesis is discussed.

- In chapter 3, firstly, the design of the mining in the space game treated in this study has been analyzed. In this subchapter, the rules and the mechanisms of the game are underlined. Afterwards, the development of the game is considered, focusing on the block diagram, key to how the game works. Finally, the graphic and the data structure of the game is explained.
- In chapter 4, the theory of the algorithms used is explained, focusing on examples. To let the reader understand it better.
- In chapter 5, the application within the game of the algorithms is discussed. Moreover, the pseudocode of the application of the algorithms is presented.
- Finally, in chapter 6, the results are presented, based on the output of the game.

7.2 Future studies

The results of this study is already satisfying. It is clear that in this Mining in Space game, the algorithms that invest more time in the exploration phase are the ones that performs the best. Moreover, there are no differences in considering the different strategies about the crystal's production of the planets regarding the results.

Obviously, however, improvements can be made to this master thesis:

Considering the algorithms: more algorithms and politics could be tested. For example, it is interesting to see that, in some cases, a significant gap between the two spaceships can be observed although the strategies do not privilege one player over the other, regardless of whether the production of crystals on the planets is based on a Gaussian distribution for all the planets or different distributions. For this reason, it would be interesting to integrate a policy whereby the spaceships are aware that they have to compete with an

opponent and can predict the opponent's moves in such a way as to be able to optimize the collection of crystals.

Considering the development of the game: some simplifications could be overcome. The correlation about the crystal's production of the planets could be implemented, and different initial exploration strategies could be tried. Moreover, the graphic of the game could be more dynamic and captivating.

References

[1] Maarten Speekenbrink Eric Schulz, Emmanouil Konstantinidis. *Exploration-Exploitation in a Contextual Multi-Armed Bandit Task.* 2015.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems.* O'REILLY, 2019.

[3] Marie-Karelle Riviere Maryam Aziz, Emilie Kaufmann. *On Multi-Armed Bandit Designs for Phase I Clinical Trials.* 2019.

[4] Steve Roberts. *A BABY ROBOT'S GUIDE TO REINFORCEMENT LEARNING.* 2020.

[5] Steven L. Scott. *A modern Bayesian look at the multi-armed bandit.* 2010.

[6] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits.* Microsoft Research NYC, September 2019.

[7] Ilya O. Ryzhov Warren B. Powell. *Optical Learning.* WILEY.

APPENDIX

Appendix A

Technical implementation

The goal of the Appendix A is show the programming languages and the packages used to be able to work on this study. Firstly, Python is introduced, with the relatives packages used. Finally, the use of R is discussed.

A.1 Python

"Python is powerful... and fast; plays well with others; runs everywhere; is friendly easy to learn; is Open. These are some of the reasons people who use Python would rather not use anything else." [1] This is a citation from the official python website. Python is the most used programming language in the Data Science field and, in general, in many fields, including the scientific one. The most powerful characteristic of it is the simple syntax and the vast amount of external libraries. The strong community is a plus.
The version used is the Python 3.9.0. It was released on the 5th of October 2020.
The Environment used is PyCharm, the Community Edition 2020.2.3. PyCharm is an integrated development environment (IDE), developed by the company JetBrains.

[1] https://www.python.org/about/

A.1.1 Packages used

The packages installed and used in this work out of Python's standard utility modules are reported in the following list. Descriptions of packages are in accord with their official documentation. Installed version is indicated.

1. **Pandas 0.24.2**
   pandas[2] is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools.

2. **Numpy 1.16.5**
   numpy[3] is the fundamental package BSD-licensed for scientific computing with Python. It can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows it to seamlessly and speedily integrate with a wide variety of databases.

3. **Random**
   random [4] is a module that implements pseudo-random number generators for various distributions.

4. **Time**
   time [5] module provides various time-related functions. For this study, the function *sleep()* is used. It suspend execution of the calling thread for the given number of seconds.

5. **Turtle**
   turtle [6] is the module used to do the graphic part of this study. Turtle graphics is a popular module that allows the creations of basic game graphic. It was part of the

[2]Source: https://pandas.pydata.org/
[3]Source: https://numpy.org/
[4]https://docs.python.org/3/library/random.htmlmodule-random
[5]https://docs.python.org/3/library/time.htmlmodule-time
[6]https://docs.python.org/3/library/turtle.html

original Logo programming language developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967.

A.2 R

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. [7].
According to the documentation:
"R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has:

- an effective data handling and storage facility;
- a suite of operators for calculations on arrays, in particular matrices;
- a large, coherent, integrated collection of intermediate tools for data analysis;
- graphical facilities for data analysis and display either directly at the computer or on hardcopy;
- a well developed, simple and effective programming language (called 'S') which includes conditionals, loops, user defined recursive functions and input and output facilities. (Indeed most of the system supplied functions are themselves written in the S language.)" [8]

The R version used is the version 4.0.3, released on 10 October 2020. In this study, it is used to get all the charts and tables results.

[7]Source: https://www.r-project.org/
[8]https://cran.r-project.org/doc/manuals/r-release/R-intro.htmlPreface