UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Air Traffic Control (ATC) Management. 3D Visualization.

Bachelor Thesis

Manish Thani Awtaney
Specialization: Software Engineering
Project Director and Tutor: Lluís Pérez Vidal
02/01/2017

Abstract

In this Bachelor Thesis, a software system has been developed for computers, which aims to renew current air traffic control systems by presenting a better user interface and show new ways of interaction and flights visualization features.

The application will let users build flights scenarios and also apply advanced settings such as the speed of each airplane. Once set the scenario, air traffic controllers can visualize and control the simulation of these flights in 2D and 3D, as well as receive feedback from the system about future collisions in the next minutes. Ultimately, they will be able to watch the results of the simulation.

Resumen

En este trabajo final de grado, se ha desarrollado un sistema software para ordenadores cuyo objetivo es renovar los actuales sistemas de control de tránsito aéreo mediante la presentación de una mejor interfaz de usuario y mostrar nuevas maneras de interacción y visualización de los vuelos.

La aplicación permitirá a los usuarios construir escenarios de vuelos y también aplicar ajustes avanzados como la velocidad de cada avión. Una vez establecido dicho escenario, los controladores de tráfico aéreo pueden visualizar y controlar la simulación de estos vuelos en 2D y 3D, así como recibir feedback del sistema sobre futuras colisiones en los próximos minutos. Finalmente, podrán ver los resultados de la simulación.

Resum

En aquest treball final de grau, s'ha desenvolupat un sistema software per a ordinadors amb l'objectiu de renovar els actuals sistemes de control de trànsit aeri mitjançant la presentació d'una millor interfície d'usuari i mostrar noves maneres d'interacció i visualització dels vols.

L'aplicació permetrà als usuaris construir escenaris de vols i també aplicar configuracions avançades com la velocitat de cada avió. Un cop establert l'escenari, els controladors de trànsit aeri poden visualitzar i controlar la simulació d'aquests vols en 2D i 3D, així com rebre feedback del sistema sobre futures col·lisions en els propers minuts. Finalment, podran veure els resultats de la simulació.

Index

Figure Indices

Table Indices

1. Introduction

This thesis forms part of my Final Degree Project from the specialty of Software Engineering at the Faculty of Informatics of Barcelona (UPC - Polytechnic University of Catalonia). It is a project corresponding to the B modality, which means that this project is linked to the UPC departments and it has been proposed as an initiative from my project director. In this case, it is linked to the department of Informatics Languages and Systems (LSI).

Also, mention that in this thesis a software solution will be proposed to solve an important issue related with the air traffic control sector by elaborating project management and development plans as described in the following sections of this document.

2. Context definition

2.1. Problem formulation

Nowadays, there are over a hundred thousand flights per day around the world and these flights should be properly managed in order to avoid conflicts and collisions among airplanes.

Managing all these flights require the help of many air traffic controllers (ATC) and the use of software systems called Traffic Collision and Avoidance Systems (TCAS), which provide additional safety preventing mid-air collisions by warning pilots when other aircraft get too close.

Flight management organizations have been handling this work for a few decades ago, as the number of flights per day was considerable. However, year by year the number of flights is increasing continuously [7], making these systems reach their limits in terms of efficiency.

Furthermore, new types of possible future collisions are appearing and most of these collisions are still not covered by the current TCAS version as they are not updated frequently by the aviation organizations [2]**.** Thus, to be more specific, these systems are getting old, presenting very limited features in terms of visualization and usability.

As a consequence, these aspects are stressing more and more current air traffic controllers, causing both performance and, essentially, safety degradation for future flights and placing an additional burden on the already overloaded human operators. This can lead to the result that air traffic conflicts can be increased gradually and future airplane collisions can possibly happen. Hence, these serious matters are pressuring aviation organizations to improve their systems and solve these problems above all.

After discovering all these problems, one can reasonably expect that optimizing and adding new features to the current air traffic control system will provide benefits in terms of safety and performance and reduce workload pressure to controllers and aviation organizations.

2.2. Stakeholders

In this section, the principal stakeholders of this project are defined. That means, a person, group or organization that has interest or concern in an organization.

Project director

A special mention should be given to the project director, who has put all his trust in the developer for doing this project and also helping him with his wide knowledge about aviation and air traffic control systems.

Developers

Normally, project developers are not included explicitly as they are only in charge of working in the project. But in this case, as this project is a bachelor thesis, the developer also takes part as a stakeholder as it will benefit him for finishing his bachelor and receive the university's bachelor degree.

Air traffic control organizations

This project is aimed for big organizations that daily manage air traffic, which means that they can make use of this software and receive benefits out of it. As the client is not unique and may vary completely depending on the organization, a generic software will be created that can be used for all the institutions. As you may notice, these stakeholders are not participating directly in the project development.

Users

Every user can have access to this software, as it will be open-source, and thus they will have access to use it and also have the possibility to access to its code. That is the reason why they can be considered as stakeholders as they can analyze the software from their personal computers. In this project we will consider users as air traffic controllers.

2.3. Scope

The main scope of the project is to develop a graphical 2D and 3D application that will help the controller to visualize the present and next future positions of each aircraft in the controlled volume and also give facilities to them when taking a decision in the presence of conflicts among airplanes.

Before proceeding to the airplanes visualization, air traffic controllers will have the possibility to customize the flights environment by adding new airplanes and its waypoints, and also update and delete them and see a preview of each airplane trajectory. Furthermore, they will also have the option to modify parameters such as the airplane's speed. Once finished doing all the necessary settings, the visualization will be initiated.

To detect airplanes in a specific air space section during the visualization, airplanes equipped with transponders will be developed to help air traffic controllers identify them so that a 2D map can be generated with the different airplanes positions and, if we need to know more details about a specific airplane, to be able to visualize it with a 3D view.

After implementing this feature, it will be useful to set radars to each plane so they can detect other airplanes and check if they are invading each other's minimum space. To do so, different airplane collision detection methods will be taken into account to compute the airplane's current and future position over time and then compare each of those techniques to know which one is more likely to predict better collisions.

It will be useful as well to develop an alert and messaging system to inform the air traffic controller about different levels of risk that may present two or more airplanes during their flights since it is very important to provide continuous feedback about the current situation of each plane at each moment in order to avoid future collisions.

In cases where future collisions may happen, the air traffic controller will have the possibility to interact with the system to move the airplanes and, therefore, reschedule the flight as much as needed.

Finally, it should be possible as well to see the results of the visualization when it ends, as this could be helpful for analyzing the final status of flights and use it for making further improvements.

3. State of the art

3.1. Contextualization

As the aviation knowledge is complex to understand intuitively, some concepts related with airspace, air traffic control and basic aviation rules need to be introduced to get a better notion about the environment where the system will be implanted.

Going into details, this software is a traffic collision avoidance system designed to reduce the incidence of mid-air collisions between aircraft. It monitors the airspace around an aircraft for other aircrafts equipped with a corresponding active transponder, and warns pilots of the presence of other transponder-equipped aircrafts that may present a threat of collision.

Nevertheless, as the airspace is very large, it can be problematic for avoiding airplane collisions to manage the whole flights around the world by a single international organization using these systems.

As a solution to this issue, currently the entire airspace is divided into small regions called flight information regions (FIR) in which a flight information service and an alerting service are provided. Every portion of the atmosphere belongs to a specific FIR. Smaller countries' airspace is encompassed by a single flight region, while larger countries' airspace is subdivided into a number of regional FIRs, and these countries are responsible for managing airplanes that enter to their region until these leave to another ones [9].

Once every country has their corresponding FIRs, these are subdivided into even more small areas that are assigned to each air traffic controller, who will be in charge of guiding all the flights that pass through that specific area.

Even being a well-organized air space management hierarchy, conflicts and collisions still appear nowadays, and to prevent it, ATCs enforce traffic separation distance rules [1], which insists that each aircraft maintains a minimum amount of empty space around it every time.

Finally, in case that an airplane invades another airplane's minimum space, several measures are taken by current systems and, thus, it is needed to describe them in order to measure and analyze their capacities.

3.2. Currently used systems in ATC management

The current existing systems and most extended in the market are the Traffic Alert and Collision Avoidance System (TCAS) family systems since these are internationally accepted by the Federal Aviation Administration (FAA) to be used in commercial and non-commercial flights. Thus, these are the only family of systems we will focus and deeply analyze as current existing systems.

3.2.1. TCAS family systems

The Traffic Alert and Collision Avoidance System (TCAS) provide a solution to the problem of reducing the risk of mid-air collisions between aircraft. TCAS is a family of airborne systems that function independently of ground-based air traffic control (ATC) to provide collision avoidance protection.

The currently used ones are TCAS I and, most widely accepted, the TCAS II:

- TCAS I provides proximity warning only, to aid the pilot in the visual acquisition of potential threat aircraft.

- TCAS II provides traffic and resolution advisories (explained in section 3.2.2.) in a vertical direction to avoid conflicting traffic. Also, systems coordinate resolution advisories to maximize aircraft separation so that if one aircraft is instructed to climb, the other aircraft will be instructed to descend.

  As a summary, this represents a better option than TCAS I as it more comprehensive by providing additional features and the main reason why it is used in the majority of commercial aviation aircraft [6]

Figure 1: *TCAS II*

According to the FAA manual, the currently used version of TCAS II is version 7.1, which was presented in February 2011 [2], adding more improvements to the previous version in terms of proposing solutions to detect new possible types of future collisions [3].

3.2.2. System components and functionalities

As detailed in the TCAS II version 7.1 Manual [2], installation consists of the following components:

- *TCAS computer units*: Performs airspace surveillance, intruder tracking, its own aircraft altitude tracking, threat detection, resolution advisory (RA) maneuver determination and selection, and generation of advisories.

- *Mode S transponder:* Aircraft have transponders to assist in identifying them on air traffic control radar and this is useful in collision avoidance systems for detecting aircrafts at risk of colliding with each other. A mode S transponder is designed to help avoiding over interrogation of the transponder in busy areas and to allow automatic collision avoidance.

- *Antennas:* These antennas enable the Mode S transponder to receive interrogations and reply to the received interrogations at considerable frequencies.

- *Cockpit presentation:* The TCAS interface with the pilots is provided by two displays: the traffic display (TA) and the resolution advisory (RA) display.

  The TA depicts the position of nearby traffic, relative to own aircraft. The RA display provides the pilot with information on the vertical speed or pitch angle to fly.

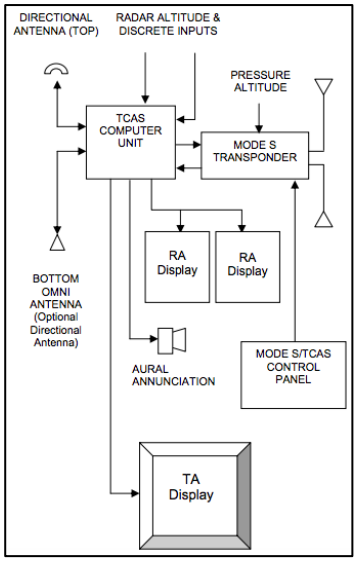

Figure 2: *TCAS II Components Diagram*

- *TCAS Control Panel:* A single control panel is provided to allow the flight crew to select and control all TCAS equipment including the TCAS Processor, the Mode S transponder, and in some cases, the TCAS displays. A typical control panel provides 4 basic control positions:
  - Standby: Power is applied to the TCAS but does not issue any interrogations and the transponder will reply to only discrete interrogations.
  - Transponder: The transponder is fully operational and will reply to all appropriate ground and TCAS interrogations.
  - Traffic advisory (TA): An indication given to the flight crew that a certain intruder is a potential threat. When a TA is issued, pilots are instructed to initiate a visual search for the traffic causing the TA. If the traffic is visually acquired, pilots are instructed to maintain visual separation from the traffic.
  - Resolution advisory (RA): An indication given to the flight crew recommending a maneuver intended to provide separation from all threats or a maneuver restriction intended to maintain existing separation.
    When an RA is issued, pilots are expected to respond immediately to the RA unless they take their own decisions, which could put into risk the safe operation of the flight. In these cases, the controller is no longer responsible for separation of the aircraft involved in the RA until the conflict is terminated.

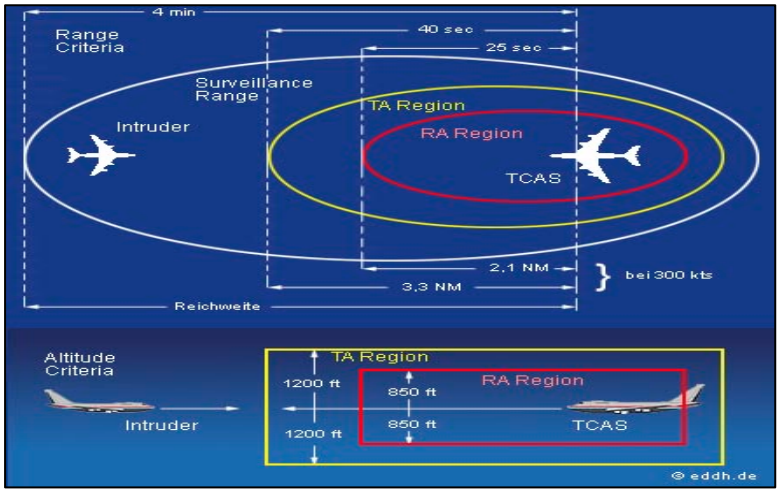

Figure 3: *Traffic and Resolution advisories schema*

3.3. Analysis conclusions

Once researched all the characteristics of these systems, we can observe that even TCAS II has been widely accepted by the aviation administrations, it is not aimed to manage a high number of flights at the same time, provoking many air traffic incidents [5].

Furthermore, TCAS II is designed to prevent mid-air collisions between aircraft, but the current version of this system does not perform a perfect role of aircraft collision detection and avoidance, hence mid-air collisions and near misses do occur occasionally [1]. In Figure 4, some researchers show the worldwide annual flight hours and mid-air collision rate that can be seen during the previous years [7].

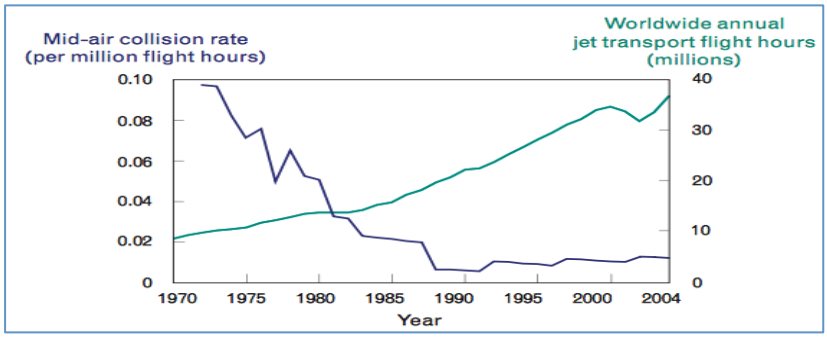

Figure 4: *Worldwide annual flight hours and mid-air collision rate (collision rate based on a ten-year moving average).*

Another important fact is that it does not provide protection against aircrafts that do not have an operating transponder [1]. Moreover, they provide traffic and resolution alerts in the vertical plane, but not in the horizontal plane. Also, their interface is built in 2D with not so many controls and do not provide a very suitable user experience [4].

To sum up, these systems need to be improved due to the limitations described above, requiring them to be updated as soon as possible.

4. Project Methodology

4.1. Agile methodology approach

Due to the tight timetable where the project is being developed in, an agile approach seems to be the best fit. Currently accepted agile methodologies (Extreme programming, SCRUM...) are very team-oriented, and so strictly following any of them would make no real sense.

However, some of these methodologies concepts are still valid for solo projects, like intensive client feedback, which is a concept from agile methodologies even though the project does not have a real client, but by letting the person that is responsible for evaluating the project check the project state as frequently as possible, chances for misunderstandings are greatly reduced and, if they happen, it can be corrected much faster.

4.2. Development Tools

As explained before, the simulation program is in 2D and in 3D. Using the conventional programming languages for developing graphical applications such as C++ with the help of the OpenGL API, are the optimal option in terms of efficiency if an application requires it, but using these technologies can make the development process longer and more complex as we is also needed to build also an engine from scratch, apart from investing time on learning these technologies.

As our application does not need to have very high graphics and also can run at low frame rates, Unity engine [10] is a great option for this graphical application as it helps us to build it faster and, therefore, add more features to it during the given development time.

Furthermore, Git is used for version control of the project and also because it enforces a short cycle approach (by the means of commits) and forces the developer to document any changes made during the development. Github provides a Git repository and it is useful for tracking progress.

4.3. Testing

Any large and complex software requires to be tested in order to check that the functionalities of the system work as expected.

Thus, Unit Testing software testing method is used in our project, as its primary goal is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating the code.

Therefore, this software testing methodology fits with our development process as it reduces the level of bugs in production code, saves development time and automated tests can be run as frequently as required. Furthermore, Unity provides many tools to help integrating this testing method into our application.

4.4. Validation methods

The combination of Unit Testing, Git version control and weekly meetings with the project director are held to ensure the validation of the objectives on their own without need for further measures.

4.5. Possible obstacles and solutions

4.5.1. Timetable

It is expected that the project and its documentation end by the 15th of January 2017, which means that there is a very short period of 5 months to finish it. Due to these conditions in which this project is developed, the timetable is a critical part of the project. To overcome it, a very rigid and realistic timetable will be set with weekly meetings with the project tutor to ensure the project is following the timetable.

4.5.2. Bugs

It is widely accepted that any large complex system can easily introduce bugs during its development. For our system, it is very necessary to ensure the robustness since a lack of this aspect can produce as a consequence many disasters.

Thus, in order to ensure that no bugs are present, a wide robust set of automated tests will be run after completing every task thanks to unit testing.

4.5.3. Usability and UX design

Usability is an important characteristic for this kind of graphical application since air traffic controllers must be able to execute tasks as fast as possible and reduce the learning time to use the application as much as possible and improve the user experience, and this will not be possible if the application interface is not usable.

Thus, we need to perform a UX design already adapted to air traffic controller's domain of knowledge and show meaningful user feedback.

5. Project planification

5.1.Schedule

5.1.1. Estimated project duration

The estimated project duration was approximately 5 months taking into account that it started the 20th of August 2016 and ends the 15th of January 2017.

5.1.2. Considerations

It is important to consider that the initial planning has been revised and updated during the evolution of the project. This happens frequently since the fact of using agile methodologies concepts also implies that new requirements can appear and modify the proposed planning.

5.2. Project Planning

During the project planning, it is important to mention that at every stage, task and subtask needed for developing the system, it is also necessary to invest a small amount of time to write all the progress done in the project report. In the following sections, the main tasks of the project to be developed will be explained.

5.2.1. Preliminary research

Before starting the development of a project, it is important to gain some basic understanding about the field where we want to integrate our system to get a quick orientation on the next development stages. In the context of this project, our main research field is air traffic control management.

Furthermore, this period of time is used to establish a first meeting with the project director, who has a lot of experience in air traffic control, and to learn more about this field and also to clarify the main purpose, objectives and development methodology.

To do so, a MacBook Pro laptop is used as hardware resource, and Google Docs as software resource to elaborate documents.

5.2.2. Project planning and feasibility

This stage consists of different tasks that should be analyzed and discussed in order to elaborate a correct project planning, which can be listed as following:

- Scope
- State of art
- Requirements
- Planning
- Budget
- Contextualization
- Conditions

Moreover, during the planning task, a Gantt diagram is designed to schedule all tasks and subtasks and also to help to guarantee if it is feasible to complete all development stages in the accorded amount of time. This diagram can be seen in Figure 5 in section 5.4.1.

Finally, a MacBook Pro laptop is used as hardware resource, and Google Docs, Google Sheets and GanttPro tools as software resources to write the corresponding documentation and elaborate tables and diagrams to define the different tasks.

5.2.3. Project Implementation

In this phase, the system implementation is done by dividing this process into specific subtasks:

- Environment configuration: It consists in setting up all the software resources needed to be able to start developing the system.

- Build 3D scene: Design a basic 3D landscape with many airplanes and also implement features such as performing each aircraft movement around the scene.

- Build 2D scene: This subtask consists in visualizing all the airplane's position from top and manually controlling each aircraft movement in case of collision.

- Integrate collision detection methods: Each airplane has radars that must be able to detect future collisions with other airplanes and different techniques are held to make it possible.

- Build flight scenario: Consists in scheduling airplane trajectories and also designing Flight Information Regions (FIR) to describe a common flights scenario.

- Design an appropriate User Interface: Several information panels must be implemented in this subtask for managing airplanes, showing messages and alerts about future collisions during the visualization, and also for showing the results of the visualization through more panels.

Note that all these subtasks are developed using the Unity and Blender software resources and a laptop as hardware resource.

5.2.4. Testing

A testing period of time is necessary to check that the functionalities implemented in the previous stage are working as expected. Therefore, a laptop and Unity software testing tools is used to check the collision detection methods, flights scenarios and, finally, verifying that the user interface is appropriate.

5.2.5. Final Document

The final stage consists on definitely closing the project development by handing a user manual and final report revised by the author and the project director and preparing a final presentation. This is done by using softwares such as Google Slides and Google docs, and using a MacBook Pro laptop as hardware resource.

5.3. Estimated time

In Table 1, the estimated amount of hours required to complete the project is shown. Note that each week has 7 working days and that a day is represented as 4 hours of dedications, considering the same as working daily on it part-time. As said before, it must be considered that each day 30 minutes is dedicated to write the final document, as it is also an important part of the project.

	Number of Days	Hours / Day	Number of Hours
Preliminary Field Research	5	3.5	17.5
Project Planification and Feasibility	31	3.5	108.5
Implementation	80	3.5	280
Testing	25	3.5	87.5
Final Document	144	0.5	72
		Total Hours	**565.5 hours**

Table 1: *Total estimated cost hours*

Nevertheless, some potential risks are detected in the elaboration of this planification, which are mainly found in the implementation and testing tasks. To be more specific, collision detection methods and UI subtasks can cause a significant delay since these parts require many discussions with the project director to find the optimal efficient collision methods, and also to find and implement ideas to renew the interface to make it more usable for users. Hence, one can assume that these two risks require being optimized and deeply tested afterwards. The total estimated project duration could be modified if we consider these pessimistic cases by adding an extra amount of 30 hours to it. To handle these possible risks, an action plan is used, which can be depicted in section 5.5.

To sum up, the total estimated time to develop the project is 565.5 hours in the normal case and 565.5 hours in the pessimistic case.

5.4. Gantt chart

In the following sections, two Gantt charts will be presented. The first one helps to see a generalized perspective of the project development, while the second one shows each task very detailed together with its risk level and role assignment (Project Manager, Software Developer, Analyst or Tester). Just to clarify, in each Gantt chart, the duration is measured in days.

5.4.1. General Chart

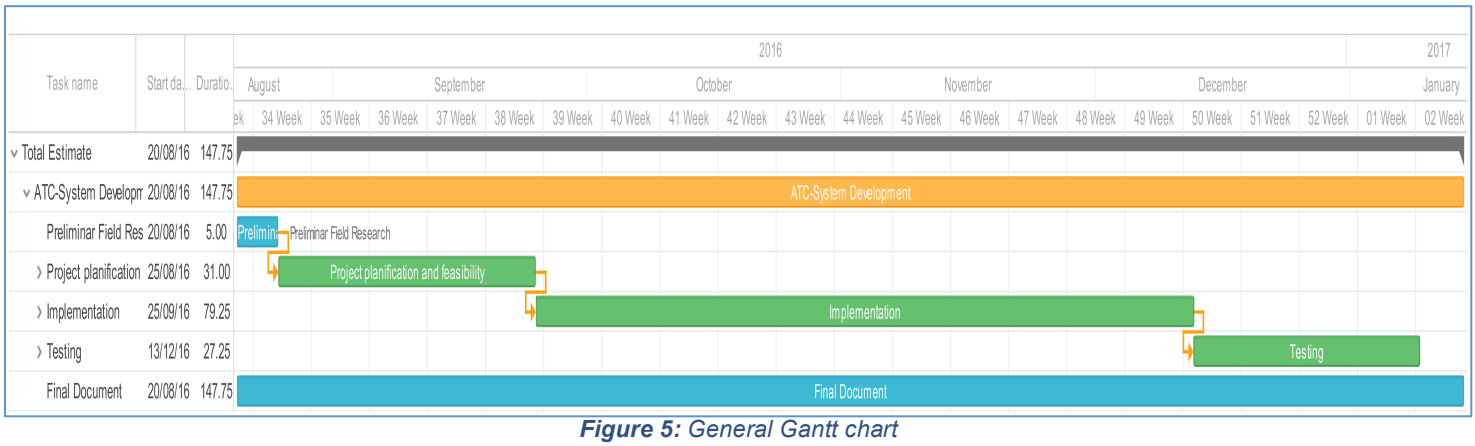

Figure 5: *General Gantt chart*

5.4.2. Detailed Chart

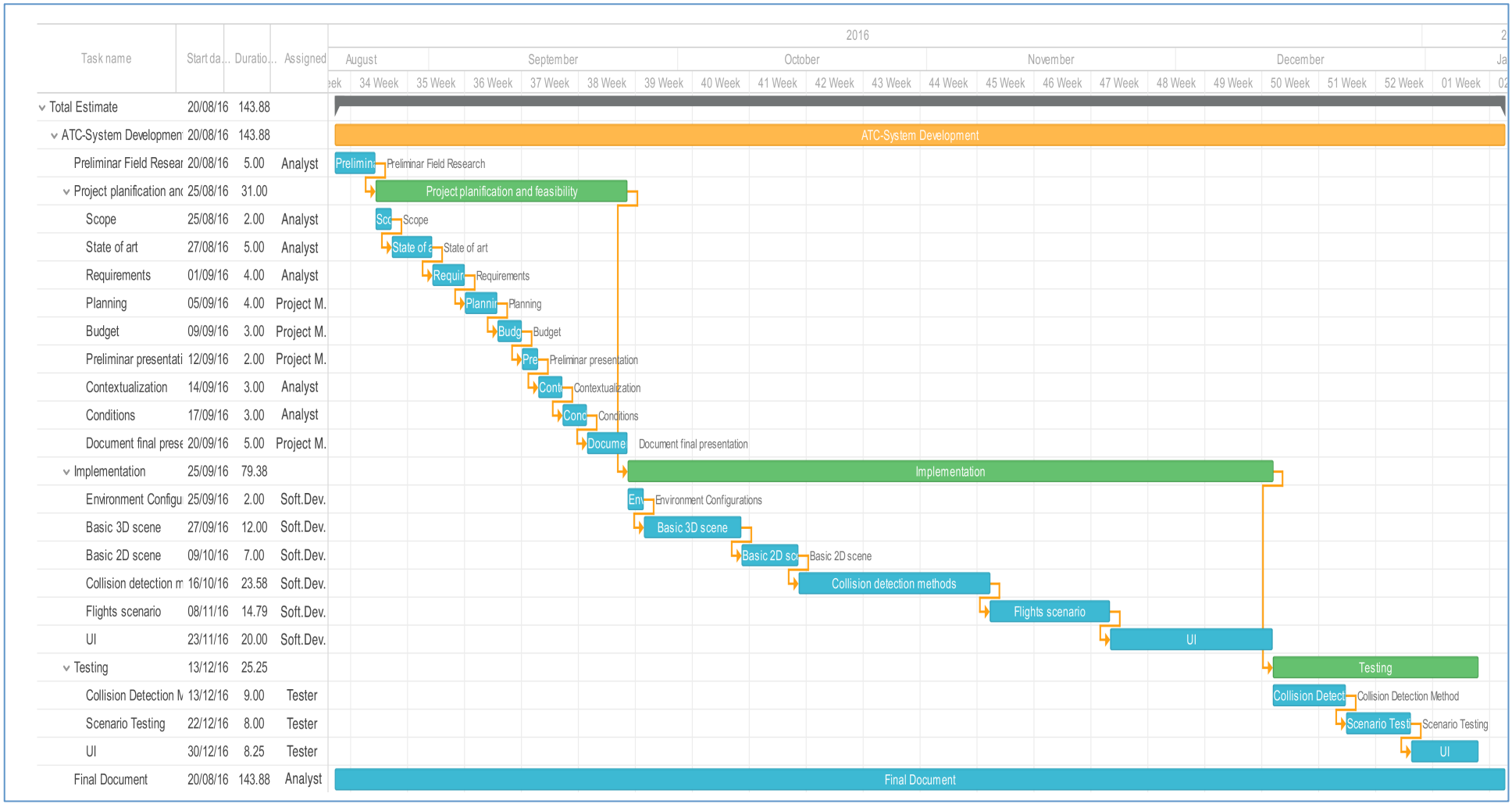

Figure 6: *Detailed Gantt chart*

5.5. Action Plan

As explained previously, the agile methodology concept of intensive client feedback lets us review and adapt dynamically the initial planning. Therefore, if the duration of the stages differs with the expected ones, the planning needs to be modified. For example, if the stage has less duration than expected, the next one will start immediately. However, if the task takes more time to be completed than expected, it will delay the following tasks.

Every week, a meeting with the director of the project is held in order to analyze the project and confirm that it is following a correct process.

In conclusion, there are approximately 15 meetings and the estimation of the dedicated hours per week is 28 hours/week in the normal case. Hence, the project planning is attainable.

5.6. Technologies

5.6.1. Hardware

From the hardware side, we are using a MacBook Pro 2011 Computer as it fulfills the hardware requirements to run the system.

5.6.2. Software

Unity 5

Unity 5 [10] is a multiplatform video game engine created by Unity Technologies. This software has been chosen for the development of this project for several reasons:

- It is especially oriented to the development of video games and other graphical environments, so it matches our needs since one of the main parts of our graphical application is a flight simulator.

- There is a large community of users on the Internet. This helps us considerably when technical doubts occur during development and also reduce an important amount of time in situations where we can reuse other developers extensions instead of creating our owns from scratch.

- The license is completely free unless we generate greater incomes than $100,000 with this project, which is definitely not our case as it is aimed for academic purposes.

- Personally, I am very interested in learning this technology, as it is a very powerful and can be useful for personal professional development. This leads to a small drawback since I am not very experienced using this software, so additional time is required to handle better this technology.

Blender

Blender [11] is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation.

It is used during the implementation of the system since it is needed for creating our custom 2D and 3D models and importing them into the Unity software.

Google Docs, Slides and Sheets

These tools will be used to write documents, elaborate planning and budget sheets and also make presentations online. They bring several advantages as the user can:

- Access, create, and edit documents wherever they go even when there is no connection.
- Everyone can work together in the same document, slide or sheet at the same time.

These features are very useful for the project director and me because the documentation related to the project can be easily evaluated and corrected.

Cacoo

Cacoo is an online tool that allows users create diagrams related to the software. In our project, the Cacoo tools are used basically to generate the class diagrams and show them in this document.

GanttPro

GanttPro is a free online Gantt chart software for project management and it is used for our development planning.

6. Budget

6.1. Considerations

As we know, each needed resources for this project mentioned in previous sections have a price. Therefore, a project cost estimation should be done taking into account hardware, software and human resources.

6.2. Budget control management

In order to control the budget, during the meetings with the project director, the budget is revised and updated with the effective total amount of hours. As a result, the final budget is a completely real budget based on real times and risks, and also unforeseen contingencies can be handled in case they happen.

The total amount of hours can widely vary, especially during the implementation process, making necessary to follow the process above described.

6.3. Project Cost

Even knowing that the project is developed by a student, we must consider that it is developed in a real company as it is essential for estimating the real cost of development. To do so, the total cost can be estimated by the sum of direct costs and indirect costs, that are calculated in sections 6.3.1 and 6.3.2, respectively, and finally adding the cost for unforeseen contingencies, as described in section 6.3.3.

6.3.1.Direct costs

Direct costs are expenses that a company can easily connect to a specific "cost object," which may be a product, department or project. In the field of software projects, this includes items such as labor, software, hardware equipment and software licenses.

Human resources cost

This project is going to be developed by only one person. Hence, this person will need to perform the roles of project manager, analyst and software architect, as well as a software developer engineer in test. Thus, we need to differentiate between each role in the total of 565.5 hours. As shown in Table 2, an estimation of the cost per role is provided, showing the amount of time invested and its respective cost by each role.

Role	Estimated hours	Estimated price per hour	Total estimated cost
Project Manager	49 hours	60.00 € / h	2,940.00€
Analyst	149 hours	50.00 € / h	7,450.00€
Software Developer	280 hours	35.00 € / h	9,800.00€
Tester	87.5 hours	30.00 € / h	2,625.00€
		Total Cost:	**22,815.00€**

Table 2: *Total estimated cost per role*

This calculation can be more detailed by specifying the cost and role assigned to each task based on the project's task Gantt chart, and it can be seen in Table 3.

Task name / Title	Task hours	Assigned to	Role cost per hour	Task cost by role
1. Preliminary Field Research	**17.5**	**Analyst**	**50.00€**	**875.00€**
2. Project planification and feasibility	**108.5**	**Analyst & Project Manager**		**5915.00€**
2.1. Scope	7	Analyst	50.00€	350.00€
2.2. State of art	17.5	Analyst	50.00€	875.00€
2.3. Requirements	14	Analyst	50.00€	700.00€
2.4. Planning	14	Project Manager	60.00€	840.00€
2.5. Budget	10.5	Project Manager	60.00€	630.00€
2.6. Preliminar presentation	7	Project Manager	60.00€	420.00€
2.7. Contextualization	10.5	Analyst	50.00€	525.00€
2.8. Conditions	10.5	Analyst	50.00€	525.00€
2.9. Document final presentation	17.5	Project Manager	60.00€	1,050.00€
3. Implementation	**280.0**	**Software developer**		**9800.00€**
3.1. Environment Configurations	7	Software Developer	35.00€	245.00€
3.2. Basic 3D scene	42	Software Developer	35.00€	1,470.00€
3.3. Basic 2D scene	24.5	Software Developer	35.00€	857.50€
3.4. Collision detection methods	86	Software Developer	35.00€	3,010.00€
3.5. Flights scenario	52	Software Developer	35.00€	1,820.00€
3.6. UI	68.5	Software Developer	35.00€	2,397.50€
4. Testing	**87.5**	**Tester**		**2625.00€**
4.1. Collision Detection Method	31.5	Tester	30.00€	945.00€
4.2. Scenario Testing	28	Tester	30.00€	840.00€
4.3. UI	28	Tester	30.00€	840.00€
5. Final document	**72**	**Analyst**	**50.00€**	**3600.00€**
			Total	**22,815.00€**

Table 3: *Costs per task*

Software cost

Additionally, some software products are needed to carry out the project. Although some of them are available for free, the real cost must be considered since this is an academic project. In Table 4 the software budget is shown.

Product	Price	Units	Useful Life	Total Estimate
Unity 5 Pro	111.58 € (125$) / month	1	5 months	557.90€
Blender	0.00€	1	N/A	0.00€
Google Docs, Slides and Sheets	0.00€	1	N/A	0.00€
GanttPro	0.00€	1	N/A	0.00€
Adobe Reader	18.14€ /month	1	5 months	90.70€
			Total	**648.60€**

Table 4: *Software costs*

Hardware cost

In order to be able to design, implement and test all the application's functionalities, a set of hardware is needed for doing it. In Table 5, an estimation of the cost of hardware resources is provided taking into account their useful life and considering the depreciation aspect, as well as their amortizations.

Product	Price	Units	Useful Life	Total Estimate
MacBook Pro 15"	1,649.00€	1	5 years	164.90€
			Total	**164.90€**

Table 5: *Hardware costs*

Other licenses cost

Finally, other licenses are needed in order to carry out the project. In our case, a repository is necessary for saving the code securely and saving its different versions, as shown in Table 6.

Product	Price	Units	Useful Life	Total estimate
Github (premium account)	6,10 € (7,99$)/month	1	6 months	36,60 €
			Total	**36,60€**

Table 6: *Table of license costs*

Total direct cost

By adding all the budgets provided above, we can get the direct estimated budget for this project, as shown in Table 7.

Concept	Estimated cost
Human resources	22,815.00€
Hardware resources	164.90€
Software resources	648.60€
Other licenses resources	30,50 €
Direct cost total	**23,659.00€**

Table 7: *Table of direct cost*

6.3.2. Indirect costs

These include items such as office equipment rental, electricity and water consumption and some other office utilities. While these items contribute to the company as a whole, they are not assigned to the creation of any one service. Thus, these cost estimations can be shown in Table 8.

Product	Price	Units	Useful Life	Total estimate
Office rent	300.00€ / month	1	5 months	1,500.00€
Electricity and water consumption	100€ / month	1	5 months	500.00€
Utilities (desk, chair, etc.)	200.00€	1	N/A	200.00€
			Total	**2,200.00€**

Table 8: *Table of indirect cost*

6.3.3. Unforeseen contingencies

As a contingency measure, a margin of 10% is established over the total estimated cost of the project.

6.3.4. Total cost

The total cost is given by the sum of the direct, indirect costs and unforeseen contingencies, and it can be seen in Table 9.

Concept	Estimated cost
Direct costs	23,659.00€
Indirect costs	2,200.00€
Subtotal	25,859.00€
Unforeseen contingencies (10%)	2585.90€
Total	**28,444.90€**

Table 9: *Project total cost*

7. Sustainability

In this section, several dimensions of sustainability such as economic, social and environmental of the project will be analyzed and described in order to better quantify its impact.

7.1. Economic dimensions

As seen in the project cost section, both human and material resources are considered, so measures can be taken in case of unforeseen contingencies that may happen during the development of the system.

Furthermore, this project cost is viable and competitive since the minimum resources are consumed. Apart from that, the project is on schedule and an effective methodology is used for achieving an optimal product quality.

In case that at some point of the development process, the project director requires to increase or reduce the features that should be added in the system, the schedule could be easily adapted, specially during the implementation phase.

Also mention that this system does not emerge from any other big project or any collaboration agreement.

7.2. Social dimensions

Currently, the ATC sector needs to implant better systems in order to handle the increasing number of flights internationally. Therefore, this system can make an important business by improving this situation and by adding more facilities to air traffic controllers while they are doing their work. Also, more safety can be achieved by implanting this system in the current society. This means that the consumers' quality of life will definitely improve as less air traffic collisions may happen.

Finally, the only factor that this system can possibly harm the air traffic sector is the learning time for using new systems, since it always requires an important adapting time to pilots and air traffic controllers in order to use it properly. Apart from that, no harm to other collectives is predicted.

7.3. Environmental dimensions

As our project is a software system, there will be no environmental impact when implanting it into the society. However, some material resources specified in the above sections such as electricity and water consumption are used responsibly during the development.

Finally, this software system can be reused by other organizations as this project is intended to be open source.

7.4. Sustainability ratings

Once explained all the different dimensions, we should analyze and self-assess each of them in terms of resources used for the project development, how much exploited they have been and rating possible sustainable risks that this project can produce.

In Table 10, these ratings are shown. Note that project development values range from 0 to 10, exploitation values from 0 to 20 and risks values from -0 to -20.

	Project development	Exploitation	Risks
Environmental	8.2	18	-3
Economic	8.7	18	-5
Social	7.6	14	-8
		Sustainability	**58.5**

Table 10: *Sustainability Ratings*

As a conclusion, by summing all the ratings of each cell, the overall sustainability ratings is 58.5, which demonstrates that the project is highly sustainable considering that the maximum sustainability score achieved can be 90.

8. Specification and requirements

This chapter describes the functional and non-functional requirements that have been taken into account for the quality of the system. Given that the system has not been developed for a defined client, the requirements have been generated by the author of the project from the market study explained at the beginning of the project and consulted and reviewed by the project director.

8.1. Functional requirements

This section defines the use cases of the system through general use case diagrams, and relating them to the actors involved, which in our case, it is the air traffic controller. We also describe the use cases of the system using a table that describes:

- Main Actor
- Precondition
- Trigger
- Main success scenario
- Extensions

8.1.1. General use cases diagram

General Use Case Diagram

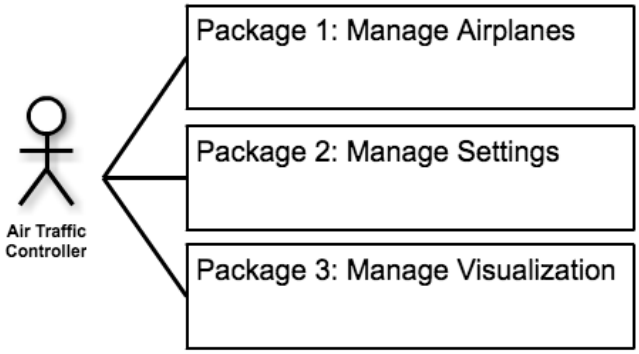

Figure 7: *General use case diagram*

Package 1: Manage Airplanes

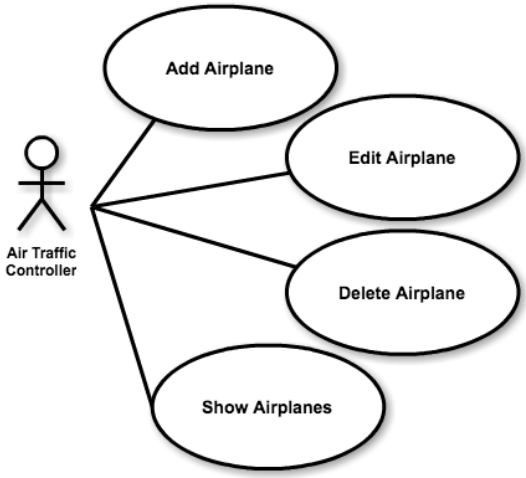

Figure 8: *Manage airplanes use case*

Package2: Manage Settings

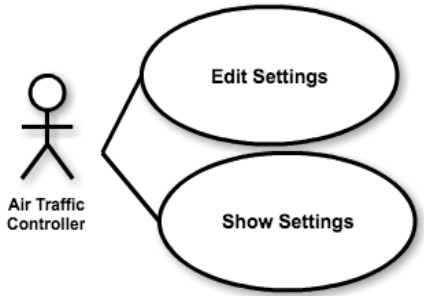

Figure 9: *Manage settings use cases*

Package 3: Manage Visualization

Figure 10: *Manage Visualization use cases*

8.1.2. Use cases description

Use case 1: Show airplanes

Main Actor: Air Traffic Controller
Precondition: None
Trigger: The air traffic controller wants to see all the airplanes.
Success scenario:

1. The air traffic controller wants to see all the airplanes by indicating it to the system.
2. The system shows all the existing airplanes and their trajectories.

Extensions:

2a. There are no existing airplanes.

2a1. The system inform that there are no existing airplanes.

2a2. The use case finishes.

Use case 2: Add airplane

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show airplanes".
Trigger: The air traffic controller wants to add an airplane.
Success scenario:

1. The air traffic controller wants to add an airplane by indicating it to the system.
2. The system shows the fields to be filled with the airplane's information by the user.
3. The air traffic controller fills the corresponding fields with the airplane's information.
4. The air traffic controller chooses to add a waypoint.
5. The system adds the waypoint and shows an updated trajectory of the airplane.

The air traffic controller repeats steps 3 to 5 as needed for completing the content of the airplane.

6. The air traffic controller confirms to add the airplane.
7. The system confirms the operation and notifies it to the air traffic controller.

Extensions:

4a. The air traffic controller decides to delete one or more airplane waypoints.
    4a1. The air traffic controller deletes one or more airplane's waypoints by indicating them to the system.
    4a2. The system deletes the selected waypoints and updates the airplane trajectory.
    4a3. Continues to step 6.

Use case 3: Edit airplane

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show airplanes".
Trigger: The air traffic controller wants to edit an airplane.
Success scenario:

1. The air traffic controller wants to edit a specific airplane by indicating it to the system.
2. The system shows the information related to the airplane selected.
3. The air traffic controller edits a field of the airplane.
4. The air traffic controller chooses to add a waypoint.
5. The system adds the waypoint and shows an updated airplane's trajectory.

The air traffic controller repeats steps 3 to 5 as needed for completing the content of the airplane.

6. The air traffic controller confirms the changes in the airplane.
7. The system confirms the operation and notifies it to the air traffic controller.

Extensions:

4a. The air traffic controller decides to delete one or more airplane waypoints.
    4a1. The air traffic controller deletes one or more airplane's waypoint by indicating them to the system.
    4a2. The system deletes the selected waypoints and updates the airplane's trajectory.
    4a3. Continues to step 6.

Use case 4: Delete airplanes

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show airplanes".
Trigger: The air traffic controller wants to delete an airplane.
Success scenario:

1. The air traffic controller wants to delete airplanes by indicating it to the system.
2. The air traffic controller selects the airplanes to be deleted.
3. The air traffic controller confirms to delete the selected airplanes.
4. The system deletes the selected airplanes and updates the airplanes trajectories.

Extensions:

4a. The air traffic controller decides to cancel the process of deleting an airplane.
4a1. The air traffic controller cancels the process of deleting an airplane.
4a2. The use case finishes.

Use case 5: Show settings

Main Actor: Air Traffic Controller
Precondition: None.
Trigger: The air traffic controller wants to see the settings.
Success scenario:

1. The air traffic controller wants to see the settings by indicating it to the system.
2. The system shows the settings to the air traffic controller.

Use case 6: Edit settings

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show settings".
Trigger: The air traffic controller wants to edit the settings.
Success scenario:

1. The air traffic controller wants to edit the settings by indicating it to the system.
2. The system shows the fields related to the settings to the air traffic controller.
3. The air traffic controller edits a field of the airplane.
The air traffic controller repeats steps 3 as needed for updating the settings.
4. The air traffic controller confirms the changes in the airplane.
5. The system confirms the operation and notifies it to the air traffic controller.

Use case 7: Show visualization

Main Actor: Air Traffic Controller
Precondition: None.
Trigger: The air traffic controller wants to start the airplanes visualization.
Success scenario:

1. The air traffic controller wants to start the airplanes visualization by indicating it to the system.
2. The system shows all the airplanes in the visualization to the air traffic controller.

Use case 8: Stop visualization

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show visualization".
Trigger: The air traffic controller wants to stop the visualization.
Success scenario:

1. The air traffic controller wants to stop the visualization by indicating it to the system.
2. The air traffic controller confirms to stop the visualization by indicating it to the system.
3. The system stops the visualization.
4. The system shows the results of the visualization.

Extensions:

2a. The air traffic controller decides to cancel the process of stopping the visualization.

2a1. The air traffic controller cancels the process of stopping the visualization.
2a2. The use case finishes.

Use case 9: Show 2D mode

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show 3D mode" and the use case "Show visualization".
Trigger: The air traffic controller wants to see the visualization in 2D camera mode.
Success scenario:

1. The air traffic controller wants to see the visualization in 2D camera mode by indicating it to the system.
2. The system shows the 2D view and notifies it to the air traffic controller.

Use case 10: Show 3D mode

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show 2D mode" and the use case "Show visualization".
Trigger: The air traffic controller wants to see the 3D visualization of an airplane.
Success scenario:

1. The air traffic controller wants to see the 3D visualization of an airplane.
2. The system shows the 3D view of the selected airplane and notifies it to the air traffic controller.

Use case 11: Set Visualization Speed

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show visualization".
Trigger: The air traffic controller wants to set the speed of the visualization.

1. The air traffic controller wants to see the visualization in a different speed by indicating it to the system.
2. The system sets the speed of the visualization and notifies it to the air traffic controller.

Use case 12: Edit 2D camera position

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show airplanes" and the use case "Show 2D mode".
Trigger: The air traffic controller wants to edit 2D camera position during the visualization.
Success scenario:

1. The air traffic controller wants to edit 2D camera position during the visualization by indicating it to the system.
2. The air traffic controller edits the position of the camera.

   The air traffic controller repeats step 3 as much as needed to set the position of the camera.

3. The system confirms the operation.

Use case 13: Edit airplane's trajectory

Main Actor: Air Traffic Controller
Precondition: The air traffic controller has selected the use case "Show airplanes" and the use case "Show 3D mode".
Trigger: The air traffic controller wants to edit an airplane's trajectory during the visualization.

1. The air traffic controller wants to edit an airplane's trajectory during the visualization by indicating it to the system.
2. The air traffic controller edits the position of the airplane and its waypoints

   The air traffic controller repeats step 2 as much as needed to set the position of the camera.

3. The system confirms the operation.

8.2. Non functional requirements

All non-functional requirements follow the Volére table format [19].

8.2.1. Perception requirements

Look and feel requirements

Requirement #	#01		Type requirement:		-
Description	The design of the interface is attractive to the user				
Justification	The interface will be pleasant to the view of the users through a color scheme that will invite users to easily interact with our system and, in this way, feel comfortable.				
Satisfaction criteria	A survey will be conducted for users who have used the system, and 90% or more will confirm the design's appeal.				
User Satisfaction	4	User dissatisfaction	1	Priority:	High

Table 11: *Table of requirement #01*

Style requirements

Requirement #	#02		Type requirement:		-
Description	The design is simple and orderly				
Justification	Most users do not have high knowledge in these types of applications. Therefore, the interface should be simple and elements that may provoke confusion will not be shown.				
Satisfaction criteria	A survey will be conducted to different users, where at least the 80% of them should identify and perform correct actions easily.				
User Satisfaction	4	User dissatisfaction	2	Priority	High

Table 12: *Table of requirement #02*

8.2.2. Capacity of use and humanity requirements

Easy to use requirements

Requirement #	#03		Type requirement:		-
Description	The system is easy to use for users.				
Justification	Since users will have different ages, the system will be useful and simple at the time of making use of it.				
Satisfaction criteria	The users' interactions to certain tasks will be measured during the testing phase. The 95% of users should not exceed the expected time to perform a task.				
User Satisfaction	5	User dissatisfaction	3	Priority	High

Table 13: *Table of requirement #03*

Learning requirements

Requirement #	#04		Type requirement:		-
Description	The system is used without a deep prior training.				
Justification	Users will not need deep training in order to use the system.				
Satisfaction criteria	At least the 85% of users that have not worked with similar systems should not have problems by using the system by doing a basic training.				
User Satisfaction	5	User dissatisfaction	2	Priority	High

Table 14: *Table of requirement #04*

Comprehension and courtesy requirements

Requirement #	#05		Type requirement:	-
Description	The system has a formal linguistic register, understandable by any user.			
Justification	Users should understand the system language without any kind of complications.			
Satisfaction criteria	Understandable language is used, so no complications are expected. During the testing phase, it will be observed if users carrying out the usability tests do not have any problems with the linguistic register used.			
User Satisfaction	3	User dissatisfaction	2	Priority: Medium

Table 15: *Table of requirement #05*

Requirement #	#06		Type requirement:	-
Description	Texts displayed by the system are completely readable			
Justification	The texts will be readable for any type of user, independently of any vision problems they may have, and text sources that are known to the user will be used.			
Satisfaction criteria	Users will be tested to see their reading ability. At least the 99% of the users should be satisfied with the reading of the texts.			
User Satisfaction	3	User dissatisfaction	2	Priority: Medium

Table 16: *Table of requirement #06*

8.2.3. Performance requirements

Accessibility requirements

Requirement #	#07		Type requirement:		-
Description	The system will operate with a maximum delay of 3 seconds to run any action.				
Justification	The system has to satisfy its accessibility. One of them is ensuring a correct response time during interactions with users.				
Satisfaction criteria	At the testing stage, the time that each task can take will be measured and evaluated.				
User Satisfaction	5	User dissatisfaction	4	Priority	Medium

Table 17: *Table of requirement #07*

Availability requirements

Requirement #	#08		Type requirement:		-
Description	The system must be available for at least 99% of the time.				
Justification	The user must be able to access the system at any time. This will increase their reliability and trust in the service.				
Satisfaction criteria	Several tests will be run to make sure that the system does not crash or behaves unexpectedly by making visualizations of hundreds of flights.				
User Satisfaction	5	User dissatisfaction	4	Priority	High

Table 18: *Table of requirement #08*

Error tolerance requirements

Requirement #	#09		Type requirement:		-
Description	The system is robust in front of invalid inputs.				
Justification	The system must be able to continue normally when an input is invalid and generate the necessary actions so that this entry does not cause problems.				
Satisfaction criteria	The system will check all inputs. If it is not valid, it will generate a warning informing that the entry entered is invalid and that it must be reentered correctly				
User Satisfaction	4	User dissatisfaction	2	Priority	High

Table 19: *Table of requirement #09*

8.2.4. Maintenance and support requirements

Launching requirements

Requirement #	#10		Type requirement:		-
Description	The system will be updated without problems.				
Justification	The system will be updated without causing inconvenience to users and without causing failures to the system itself.				
Satisfaction criteria	Since our system runs offline, we will make changes to the system and test them to ensure it works as expected. Once done that, the updated source code will be uploaded to the repository.				
User Satisfaction	2	User dissatisfaction	1	Priority	Medium

Table 20: *Table of requirement #10*

Support requirements

Requirement #	#11		Type requirement:		-
Description	The system has a system usage document.				
Justification	Documentation on how to use the system should be provided to users so they can understand how the system works and how to interact with it.				
Satisfaction criteria	A usage document will be handed with all the user controls and system details.				
User Satisfaction	4	User dissatisfaction	2	Priority	Medium

Table 21: *Table of requirement #11*

8.2.3. Other requirements

Open Source Requirements

Requirement #	#12		Type requirement:		-
Description	The software is Open Source				
Justification	The user has the rights to study, collaborate, and distribute the software to anyone and for any purpose.				
Satisfaction criteria	The software source code will be uploaded to a public repository so it will be accessible to everybody.				
User Satisfaction	2	User dissatisfaction	1	Priority	Medium

Table 22: *Table of requirement #12*

9. System Design

9.1. Architecture design

The platform is structured into two different components that interact with each other. These components are the following:

- The local flat-file database [20] is where we store all the information registered in the system. Whenever a user wants to add or access information in the application, it will interact with the database to store or query the data.

- The graphical application that users will interact with in their computers. When the user interacts with the application, the application will be in charge of handling those events and querying the database when necessary.

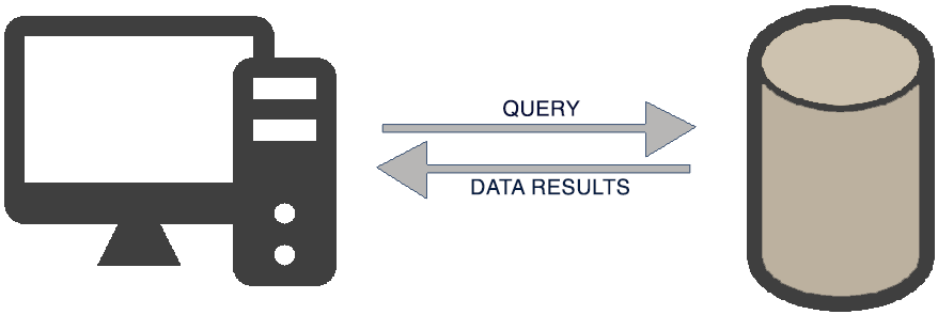

Figure 11: *Architecture and interaction between components*

9.2. Unity software concerns

The architecture in projects with Unity can be very varied, but the vast majority share several features in common.

Firstly, we must explain that Unity divides a project into scenes. Each scene for us will represent many screens that the user can interact with and navigate through them. In our system we have a total of 3 scenes:

- **Menu scene**: In this scene users can see, create, update and delete flights and apply a range of settings to them. After setting up all the flights, this information will be fed to the Simulation scene.

- **Simulation scene**: In this scene, we can visualize in 2D and 3D the flights that users created. Furthermore, we will be able to interact with these views and airplanes and also be notified of future flight conflicts. Once every airplane has arrived to its destiny or if the user wants to stop the simulation before that, the results scene will be shown.

- **Results scene**: Once ended the simulation, valuable information related to it will be shown to users. Finally, when the user has read and accepted the simulation's results, the system will arrive to the initial Menu scene.

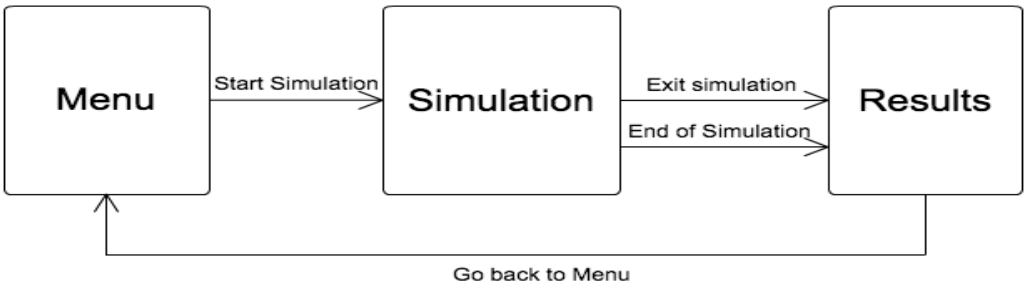

Figure 12: General interaction between scenes

Secondly, in a scene we find the so-called GameObjects, which are only empty containers, but different components can be added to convert them into elements such as user interface elements or audio.

For example, we have created an ATC GameObject, which contains other GameObjects that represent an airplane, waypoints and future positions and each of them has a wide variety of components attached such as UI elements, scripts, renderers, cameras, audios and collision detectors.

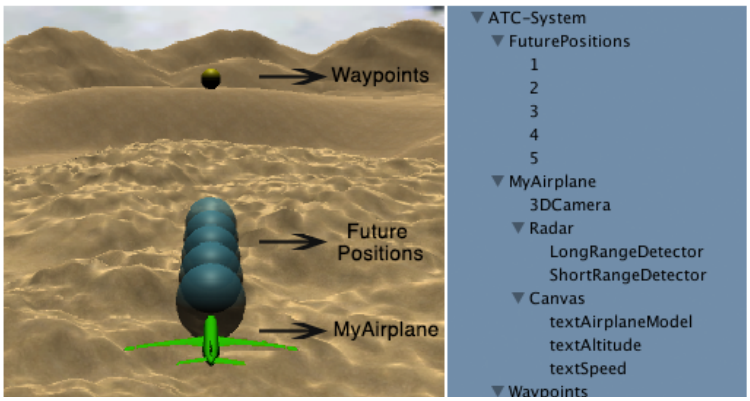

Figure 13: ATC GameObject hierarchy

In third and last place, we have the scripts, which are obligatorily paired with a GameObject of the scene, so any unmatched script will not be executed by the Unity engine.

The scripts are used to control the behavior of the GameObjects during the execution of the scene, which means that we can detect the moments when events occur on that GameObject and, consequently, make it act accordingly. In Unity, each script consists in a class that can be related with other classes that belong to other GameObjects.

As a conclusion, due to these characteristics of the Unity environment, the software patterns and project design have been detailed by strongly taking them into account.

9.3. Software patterns used

In software engineering, a software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design [13]. For this reason, several patterns have been applied to this project in order to achieve these advantages.

9.3.1. MVC pattern

The Model-View-Controller (MVC) pattern [12] separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes:

- The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

- The view manages the display of information.

- Controller: The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

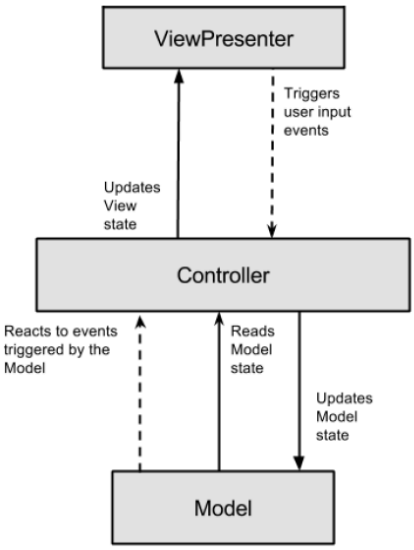

Figure 14: *MVC pattern*

9.3.2. Singleton Pattern

Singleton Pattern [17] involves a single class that is responsible to create an object while making sure that only one single object gets created. For example, it is used to create controllers that manage the system data and control the airplane's behavior during the visualization.

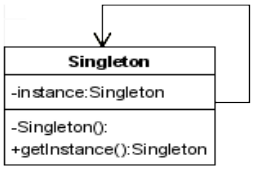

Figure 15: *Singleton pattern*

9.3.3. Prototype Pattern

Prototype pattern [15] refers to creating duplicate objects while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object in terms of cost.

In our system, we generate many objects such as airplanes and waypoints using this pattern because it would be inefficient to initialize all the parameters every time we create an object.

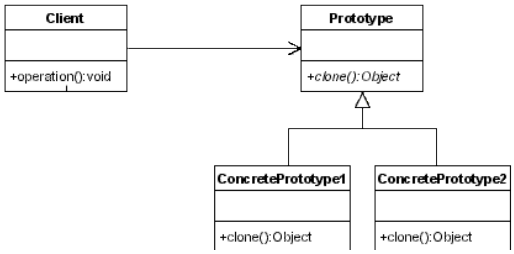

Figure 16: *Prototype Pattern*

9.4. Class diagram

As explained before, the application is divided into three scenes. Consequently, three different diagrams have been designed for each of them.

9.4.1. Menu class diagram

In Figure 17, a class diagram of the menu scene has been provided and some patterns have being applied to it to achieve reusability.

To begin with, the menu class diagram follows the MVC pattern, where each class can represent a model, view or controller as it can be observed by their class name. The process is simple for this scene, the user interface is composed of a panel for showing all the airplanes, another one for inserting airplanes and, finally, the settings interface. All these views are handled by the *AirplaneViewController* and the *SettingsViewController* classes, respectively, and on top of them, a *MainViewController* class will be in charge of enabling these two view controllers depending on the user interaction.

The airplanes and its settings represent the models, and together they form the data of our application. This data will be saved in files following a CSV format. To do so, the settings and airplanes view controller will interact with its data controller (*AirplanesDataController and SettingsDataController* classes) to update their respective model and save it to the corresponding file. In order that data controllers save this information on files, an AirplaneStorage and SettingsStorage classes have been created to do it, and both extend from the *FileStorage* abstract class since they share same path attribute.

The *AirplaneModel* model is composed of 3 attributes:

- **id**: The identification number of the airplane represented as an integer.
- **modelname**: The model name of the airplane represented as a string
- **waypoints**: All the waypoints that the airplane will follow represented as a string composed of a list of coordinates in CSV format.

The *SettingsModel* model is composed of 4 attributes:

- **maxFutureDistance**: The maximum distance that airplanes can predict about future conflicts with other airplanes represented as an integer.
- **shortRadius**: All the airplanes' short-range detector radius represented as an integer.
- **longRadius**: All the airplanes' long-range detector radius represented as an integer.
- **speed**: All the airplanes' speed represented as an integer.

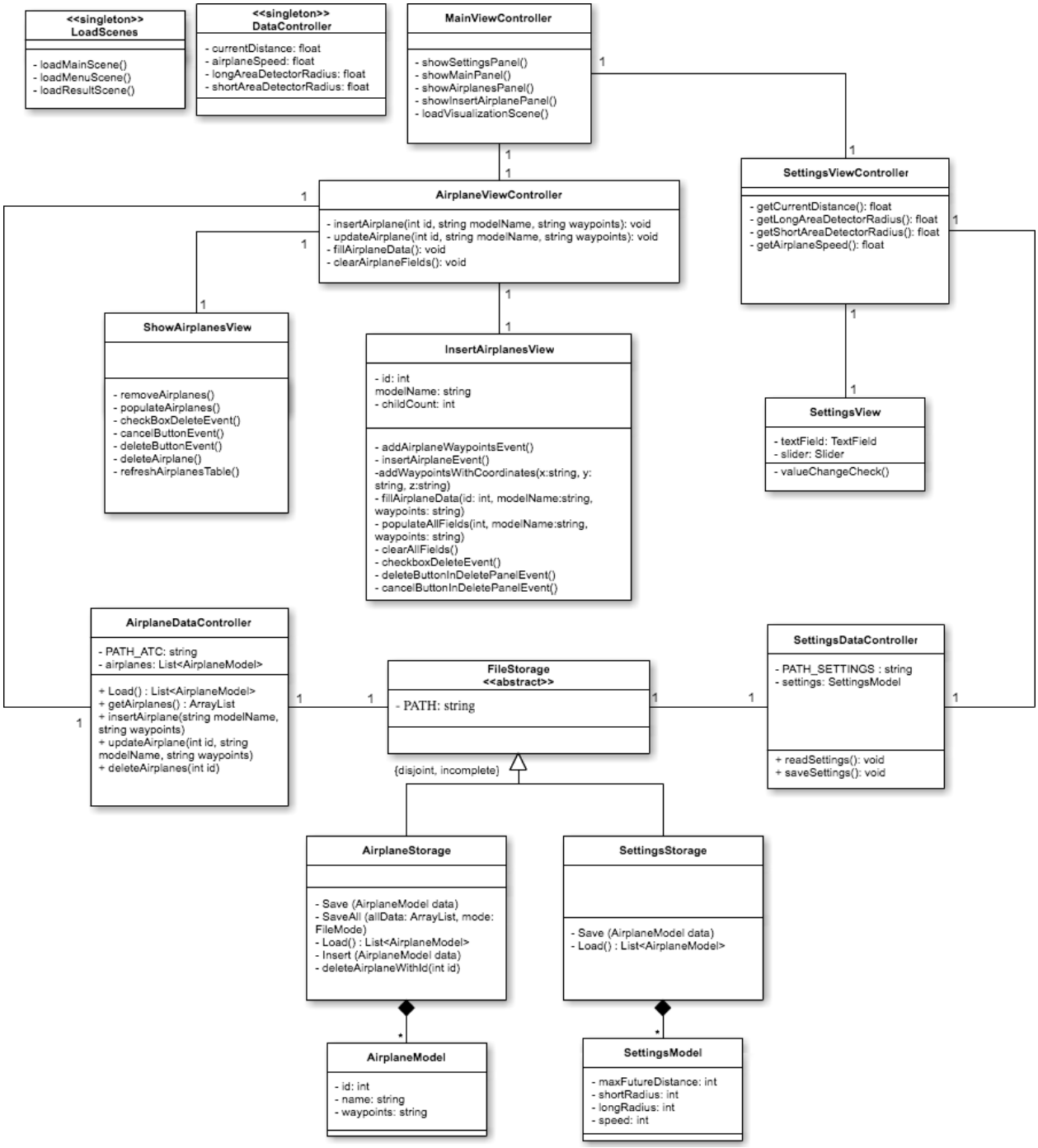

Figure 17: *Menu Class diagram*

When we want to move on to the Simulation Scene, all the airplanes and settings are saved to the *DataController* singleton class, since it will be the only class accessible from the simulation. Once done that, the main view controller will call the method to load the simulation scene by using the *LoadScenes* singleton class, which will be accessible throughout the entire system.

9.4.2. Simulation class diagram

The simulation scene starts with the *RenderController class*, which will draw all the airplanes, its future positions, waypoints and settings provided by the data controller.

Then Unity attaches several scripts to these objects in order to add them some behavior. Thus, an airplane needs to move towards its waypoints using the *AircraftMovement script*, and we also need to relate this behavior with other behaviors, that are:

- Computing the future positions of the aircraft using *FuturePositionCollection* and *FuturePositions* behaviors.
- Showing the airplane details in a panel using the *UIAirplaneDetails* class.
- Make a 3D camera that follows the airplane through the *CameraMovement* script.
- Add collision detectors and control them using *AircraftDetector* scripts.

Alternatively, the *GameMaster* script is used to control all the airplane's behavior and provide general simulation feedbacks to users. It is invoked in cases where:

- A conflict between two airplanes need to be notified through the simulation console by using *UIController* script or through an alarm system using the *AudioController* script.
- The simulation has to be accelerated or stopped by using the *UIController* script.
- The system has to update the 2D map of airplanes position using the *UIController* script.
- Switching to a 2D or 3D view is asked by using the *CameraMovement2D* script.

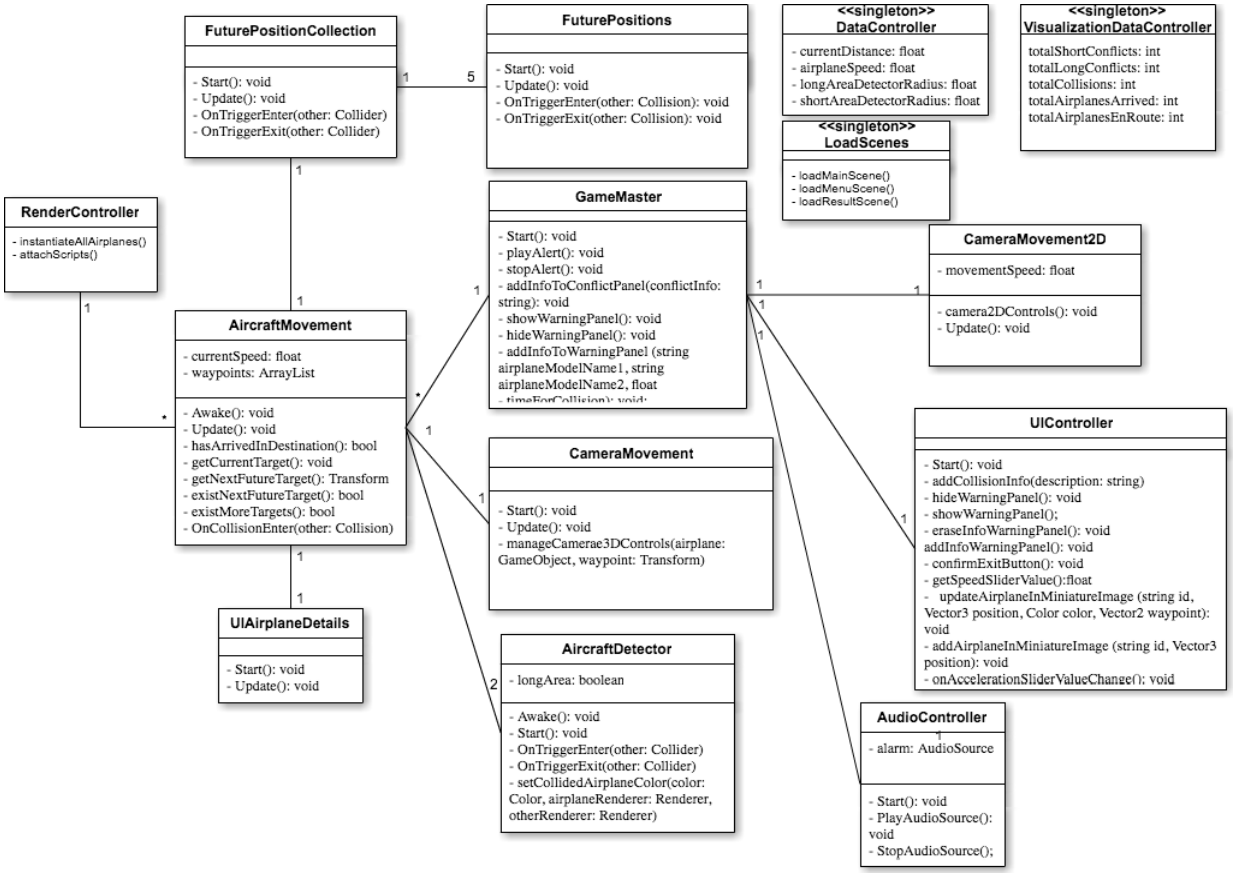

Figure 18: *Simulation Class diagram*

Finally, when the simulation finishes or the user stops it, the simulation's results information is generated and fed to the *VisualizationDataController* singleton class since it will be the only one accessible from the Results Scene. After that, the Results scene is loaded thanks to the *LoadScenes* singleton class.

9.4.3. Results class diagram

Once finished or stopped the simulation, a panel is shown with the information with the simulation, these results are obtained from the *VisualizationDataController* class and shown to the user through the *ResultsView*.

When the user analyzes the data and confirms, the system goes back to the Menu scene. In Figure 19, we can see the results class diagram.

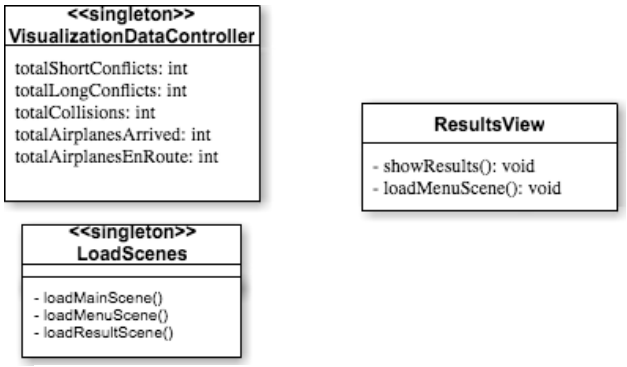

Figure 19: *Results Class diagram*

10. Testing

10.1. Manual tests

Manual testing has been done mostly on the menu interface and in the visualization part since it is the logical part of the product that made sense to try out its features by hand.

Furthermore, the design and user experience cannot be tested automatically and has to be tested by a human in order to obtain a valuable review.

10.2. Automated tests

For the automated testing part, Unity Technologies provides in its assets store the Unit Testing Tools, which is a free extension and the recommended way to test code, as it gives the option to build and execute customized tests in C#, JavaScript or Boo within the Unity environment.

The following tests have being built and checked during the development of this system:

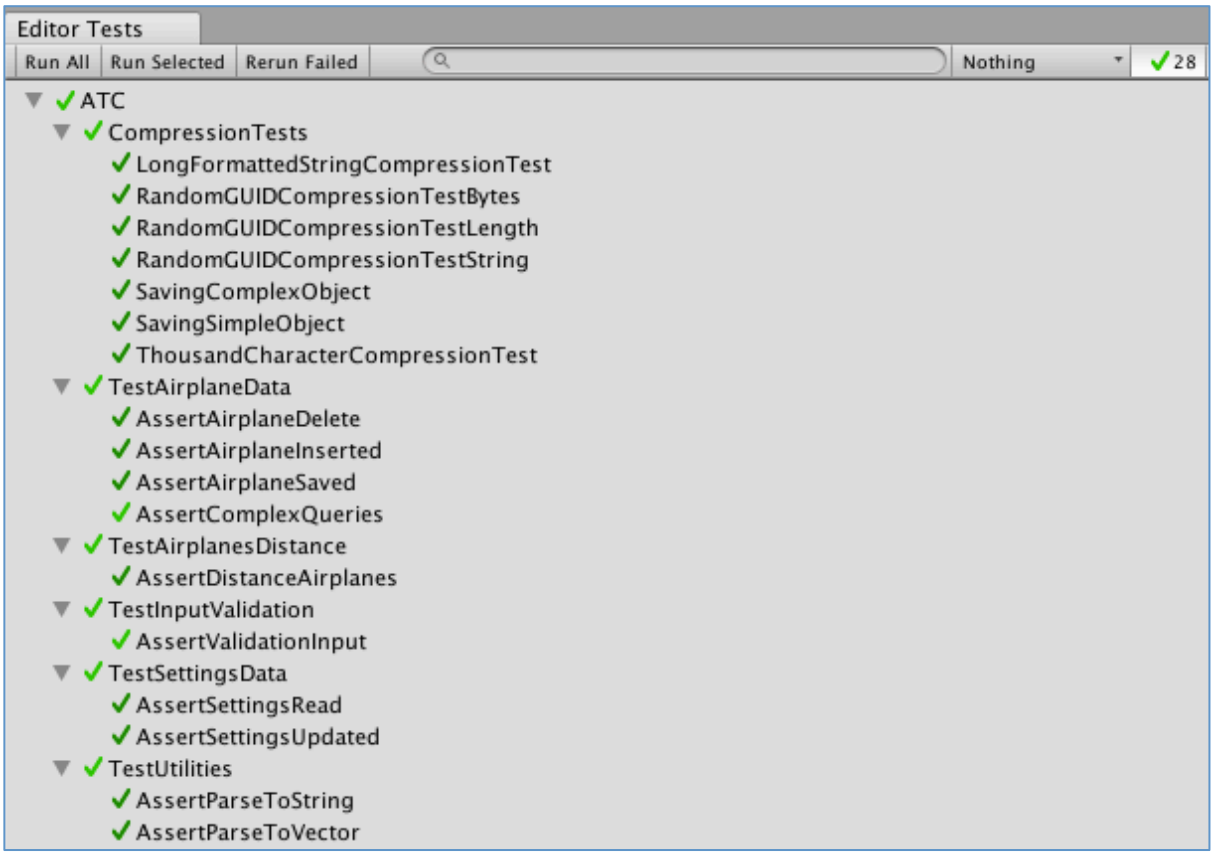

Figure 20: *Automated tests*

These tests have been done to validate that the code works as expected. When refactoring or modifying code after a significant period of time, using tests has ensured that the changes have not affected the application's behavior unexpectedly.

With the Unity testing tools and the list of tests made above, the code verification has been done effectively.

11. FAA Laws and regulations

This project is strongly related with the aviation world and, for obvious reasons, it is expected that it fulfill the rules and regulations stated by this field. All the aviation rules are firstly created and imposed by the FAA in the United States, and later the same rules are applied and imposed by other agencies internationally.

An example of these agencies is EUROCONTROL [3], who is in charge of the aviation administration in Europe. Therefore, the system will have to be implemented by respecting strictly these rules.

However, there exist thousands of laws, but only a few ones are truly applied to our system. In order to simplify that, we can state brief summaries of the applied rules:

1) For a commercial airliner, separation will usually be at least 3 miles laterally, or 1,000 feet vertically. In the en route environment, at higher operating speeds above 10,000 feet and based on the type of Radar and distance from the antenna, a 5 mile rule is applied laterally.

2) In the United States in particular, the Federal Aviation Administration imposes the concept of lowest safe altitude (LSA) [17], and specifically defines it as follows in §119 of Part 91 of the Federal Aviation Regulations (FAR) [18]:

a) Anywhere: An altitude allowing a safe emergency landing without undue hazard to person or property on the ground.
b) Over *Congested Areas*: An altitude of 1,000 feet above the highest obstacle within a horizontal distance of less than 2,000 feet.
c) Over *Populated Areas*: An altitude of 500 feet above ground level.

3) Unless otherwise authorized by the Administrator, no person may operate an aircraft below 10,000 feet MSL [21] at an indicated airspeed of more than 250 knots.

12. Conclusions

Developing the ATC Management and 3D Visualization project has been a great way to put into practice all the knowledge acquired during degree studies. All the functionalities defined at the beginning and in accordance with the scope of the project have been implemented. Even though, there have been deviations that have affected the project planning and budget as detailed previously.

Personally, I have expanded my knowledge of software development and have been able to put into practice the knowledge learned during my degree studies and also learn and apply new skills.

12.1. Difficulties and Limitations

During the elaboration of this project, several difficulties happened, especially during the beginning, where the main issue was the lack of knowledge about the aviation sector. But after a period of time, the project director helped me by sharing his wide knowledge about this field and managed to continue with the development.

Apart from that, as I was not extremely experienced with Unity engine, some programming problems also appeared while designing the user interface, but by asking experienced people about these punctual problems helped me to solve them.

12.2. Future Works

This Final Bachelor project has lots of features that can produce a change in today's aviation industry as explained in this document. Nevertheless, there are still more functionalities that can implemented.

One possible feature that can be added is to provide different solution paths that airplanes can choose to follow to avoid collisions or conflicts. This can be done by developing an algorithm that will elaborate an analysis of the possible solutions, choose the optimal ones and show them to the user. As you might notice, the process of getting the optimal solutions can require the implementation of an efficient algorithm due to its complexity problem.

Another interesting step would be to deploy this software to other platforms, such as mobiles, tablets and smart watch. This could add a positive experience to air traffic controllers as they can check the flights status anytime and anywhere and also can be notified of incidents instantly through push notifications.

Finally, more FIRs can be added to the system, as it would let air traffic controllers adapt and learn from different scenarios and learn from them.

12.3. Technical Skills

***-CES1.1**: Develop, maintain and evaluate complex and/or critical systems and software services. [In deep]*

This competence will be treated deeply as our project mainly consists in implementing a software system for air traffic controllers to manage and visualize flights, and also maintaining it.

***CES1.2**: Give solution to integration problems in function of strategies, standards and available technologies. [In deep]*

It is a must to integrate the system in the air traffic control sector. This will be possible by giving the best solutions according to the main aviation standards and using suitable software tools to guarantee the objectives of this project.

***CES1.3:** Identify, evaluate and manage potential risks associated with the construction of the software that may appear. [In deep]*

We should really focus in deep in this skill, as we have to deliver the project to the project director in the established deadline. To do so, several measures are taken during the planning of the project to handle this possible risk.

***CES1.5:** Specify, design, implement and evaluate databases. [Moderate]*

A flat-file database will be designed and built to store information related to the different airplanes and their respective flights in this software system.

***CES1.6:** Administrate databases (CIS4.3) [Moderate]*

A variety of operations will be needed to properly administrate the database such as creating, reading, updating and deleting (CRUD) information related to airplanes.

***CES1.7**: Control the quality and design tests in the software production. [In deep]*

At each step of the implementation process, several tests will be run to check properly that the systems is running as it expected, ensuring that the overall system guarantees an optimal quality. This is fundamental, as the sector where we want to integrate this system requires it to avoid aviation disasters and ensure safety above all.

***CES1.9**: Demonstrate comprehension in the management and control of software systems. [In deep]*

Thanks to the knowledge received by doing the specialty of software engineering, a correct management and control of software systems and deep comprehension will be demonstrated to the client.

***CES2.1**: Define and manage software requirements. [In deep]*

Before proceeding to the project implementation, there will be a specific task which will consist specially in defining the software requirements, as it deals with establishing the needs of stakeholders that need to be solved by the software. Thus, it is fundamental to spend time on it before doing further tasks.

***CES2.2**: Design appropriate solutions in one or more domains of applications, using software engineering methods that integrate ethical, social, legal and economic aspects. [In deep]*

Our domain of application is air traffic control management, where it is crucial to analyze these aspects as it is strongly related with the safety and security of passengers and, therefore, they should be strongly integrated using software engineering methods.

13. Acronyms

- ATC - Air traffic Control
- CRUD - Consult, Read, Update and Delete
- CSV - Comma-separated Values
- FAA - Federal Aviation Administration
- FIR - Flight Information Region
- LSA – Lowest Safe Altitude
- MSL – Mean Sea Level
- MVC – Model View Controller
- TA - Traffic advisory
- TCAS - Traffic Collision Avoidance System
- RA - Resolution advisory
- UI - User Interface
- UX - User Experience

14. Glossary

- **ATC**: Air traffic control (ATC) is a service provided by ground-based controllers who direct aircraft on the ground and through controlled airspace, and can provide advisory services to aircraft in non-controlled airspace.

- **Aircraft**: An aircraft is a machine that is able to fly by gaining support from the air.

- **Transponder**: A transponder is an electronic device that produces a response when it receives a radio-frequency interrogation. Aircraft have transponders to assist in identifying them on air traffic control radar and this is useful in collision avoidance systems for detecting aircrafts at risk of colliding with each other.

- **TCAS**: A traffic collision avoidance system is an aircraft collision avoidance system designed to reduce the incidence of mid-air collisions between aircraft. It monitors the airspace around an aircraft for other aircraft equipped with a corresponding active transponder, independent of air traffic control, and warns pilots of the presence of other transponder-equipped aircraft, which may present a threat of mid-air collision (MAC).

- **Knots:** The knot is a unit of speed equal to one nautical mile (1.852 km) per hour.

15. References

[1] Federal Aviation Administration, Accident and Incident Data. [online]. URL: http://www.faa.gov. Accessed 26/08/2016 at 16:00

[2] Federal Aviation Administration. Introduction to TCAS II. Version 7.1. [online]. URL: http://www.faa.gov/documentLibrary/media/Advisory_Circular/TCAS%20II%20V7.1%20Intro%20booklet.pdf. Accessed 28/08/2016 at 18:45

[3] Department of Research of EUROCONTROL. TCAS II. [online]. URL: http://www.eurocontrol.int/articles/tcas-ii-version-71. Accessed 05/09/2016 at 17:30

[4] FlightGlobal. TCAS and its limitations [online]. URL: https://www.flightglobal.com/news/articles/tcas-and-its-limitations-133771/. Accessed 05/09/2016 at 18:00

[5] AviationKnowledge. Traffic Alert & Collision Avoidance System (TCAS) [online]. URL: http://aviationknowledge.wikidot.com/aviation:tcas#toc11. Accessed 10/09/2016 at 16:00

[6] Henely, S., Collins, R. TCAS. [online]. URL: http://www.davi.ws/avionics/TheAvionicsHandbook_Cap_18.pdf Accessed 11/09/2016 at 17:00

[7] Kuchar, K., Drumm, C. (2007). The Traffic Alert and Collision Avoidance System [online]. URL: https://pdfs.semanticscholar.org/c35e/0023e8c2ae4e14a8aed2c996f21ffcbcbaf9.pdf Accessed 15/09/2016 at 14:30

[8] International Civil Aviation Organization (ICAO) [online]. URL: http://www.icao.int/annual-report-2013/Pages/ES/the-world-of-air-transport-in-2013_ES.aspx. Accessed 17/09/2016 at 19:00

[9] Department of Research of EUROCONTROL. FIR and UIR charts. URL: https://www.eurocontrol.int/articles/firuir-charts. Accessed 17/09/2016 at 19:00 Accessed 17/09/2016 at 19:30

[10] Unity. Developer documentation. URL: https://docs.unity3d.com/Manual/. [online]. Accessed 02/10/2016 at 16:30

[11] Blender. 3D Modeling Documentation. URL: https://www.blender.org. [online]. Accessed 20/12/2016 at 17:30

[12] Microsoft. The Model-View-Controller Pattern. URL: https://msdn.microsoft.com/en-us/library/ff649643.aspx. [online]. Accessed 15/10/2016 at 17:00

[13] Wikipedia, the free encyclopedia. Software design patterns. URL: https://en.wikipedia.org/wiki/Software_design_pattern. [online]. Accessed 24/12/2016 at 18:45

[14] OODesign. Observer Pattern. URL: http://www.oodesign.com/singleton-pattern.html. [online]. Accessed 15/10/2016 at 17:15

[15] OODesign. Prototype Pattern. URL: http://www.oodesign.com/prototype-pattern.html [online]. Accessed 15/10/2016 at 18:30

[16] OODesign. Singleton Pattern. URL: http://www.oodesign.com/singleton-pattern.html [online]. Accessed 15/10/2016 at 18:30

[17] Wikipedia, the free encyclopedia. Lowest Safe Altitude. URL: https://en.wikipedia.org/wiki/Lowest_safe_altitude. [online]. Accessed 25/10/2016 at 18:00

[18] Wikipedia, the free encyclopedia. Federal Aviation Regulations. URL: https://en.wikipedia.org/wiki/Federal_Aviation_Regulations. [online]. Accessed 07/11/2016 at 17:50

[19] Volére. Plantilla de especificación de requisitos. URL: http://www.volere.co.uk/pdf%20files/template_es.pdf. [online]. Accessed 29/12/2016 at 17:00

[20] TechTerms. Flat-file databases. URL: http://techterms.com/definition/flatfile. [online]. Accessed 01/12/2017 at 14:00

[21] SKYbrary. Mean Sea Level (MSL). URL: http://www.skybrary.aero/index.php/Mean_Sea_Level_(MSL). [online]. Accesed 02/01/2017 at 14:30

Appendix – User Manual

In this section we are going to focus on describing the different parts of the system and how users can interact with them by using keyboard and mouse controls. Those parts are classified as the menu, the simulation scene and, finally, the results scene.

6.1. Menu scene

When the graphical application is opened, the first thing that can be seen is a menu, which provides many options to start using the systems by showing three main buttons: Settings, Edit Visualization and Start Visualization. Each button enables diverse panels and options that are detailed in the next subsections.

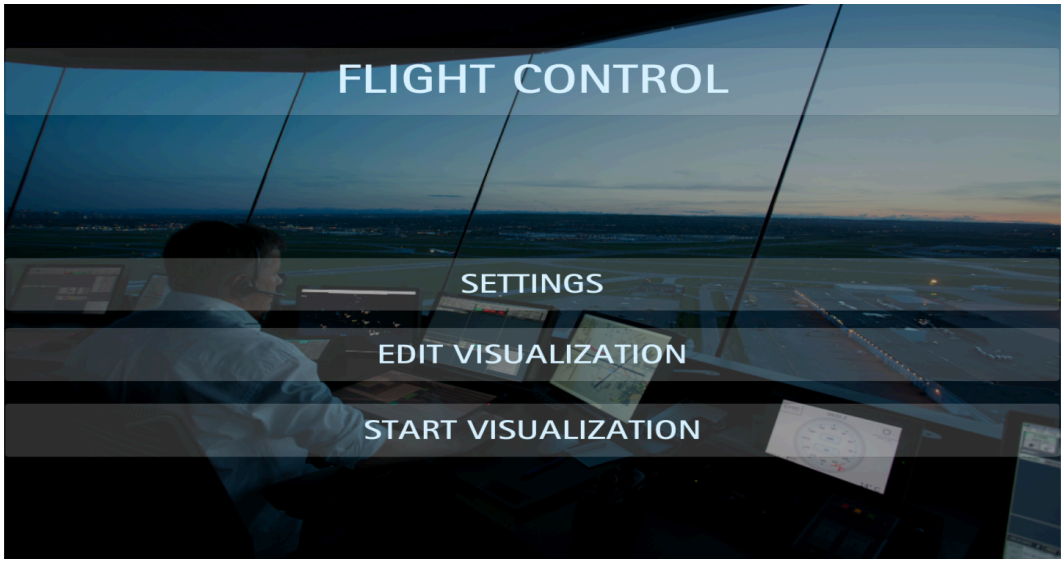

Figure 21: *Menu*

6.1.2. Settings panel

In this panel of the application, a variety of parameters can be set to customize airplanes future positions, speed and set up the range of detection of conflicts. These settings will be applied to all the airplanes during the visualization.

- **Maximum future airplane distance**: This parameter is used to determine the distance between each future position of the airplane. In other words, with this parameter we can set the amount of minutes that airplanes will be able to predict in the future. The value of the maximum future airplane distance can range from 1 to 5.
- **Long Area Detector Radius:** Sets the radius of the long-range detector of each airplane.
- **Short Area Detector Radius:** Sets the radius of the short-range detector of each airplane.
- **Airplane's speed:** Sets the speed that all airplanes will have during the visualization.

Figure 22: *Settings*

6.1.3. Edit Visualization Panel

The Edit Visualization Panel is a user interface that lets the user see the existing airplanes in a list and their trajectories, which will be used for the visualization. Moreover, air traffic controllers can add airplanes by clicking in the corresponding buttons, as well as editing and deleting the existing ones. Apart from that, it is also possible for them to see how airplanes' trajectories get updated when making changes on them.

Show Airplanes

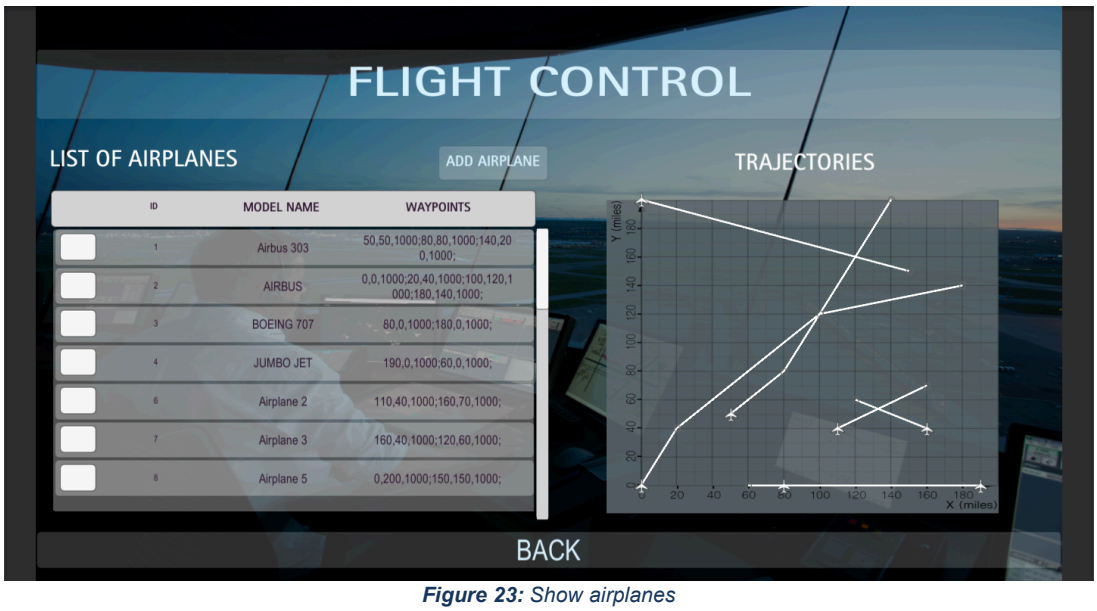

Figure 23: *Show airplanes*

Delete Airplanes

In case of willing to delete an airplane, air traffic controllers just need to check its corresponding checkbox in the list of airplanes and click the delete button. Once done that, the airplane will be deleted from the list together with its trajectory

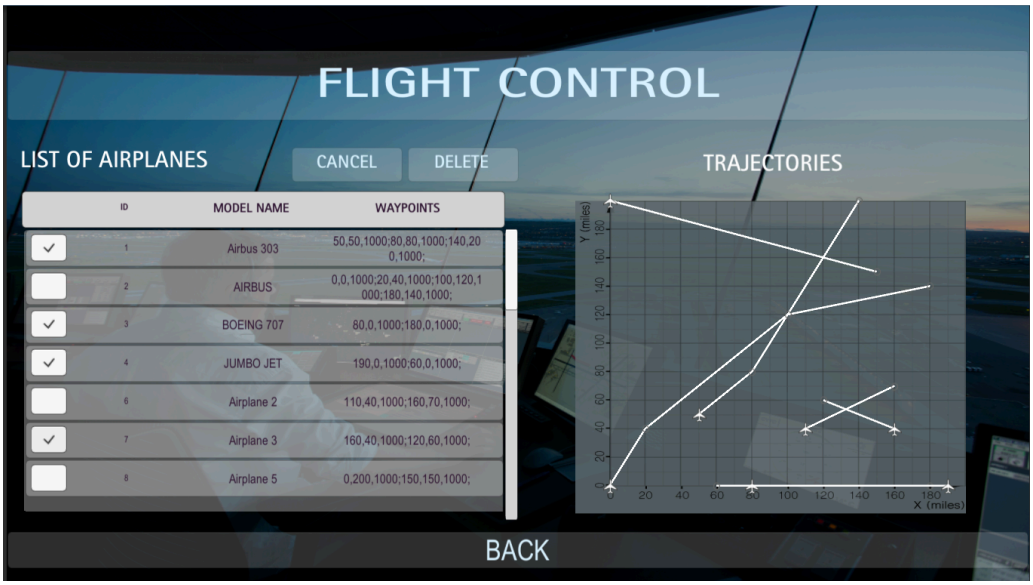

Figure 24: *Delete airplanes*

Add and Update Airplanes

When the user wants to add an airplane, the system shows an interface which can be seen in Figure 25, where we can enter its model name and define its waypoints by introducing its X, Y and Z coordinates and then click the Add Waypoint button.

Note that the X and Y coordinates can range from 0 to 200 miles, and the height Z from 1000 to 25000 feet. Also, the first waypoint added will be the initial position of the airplane and the last waypoint added will be the destination.

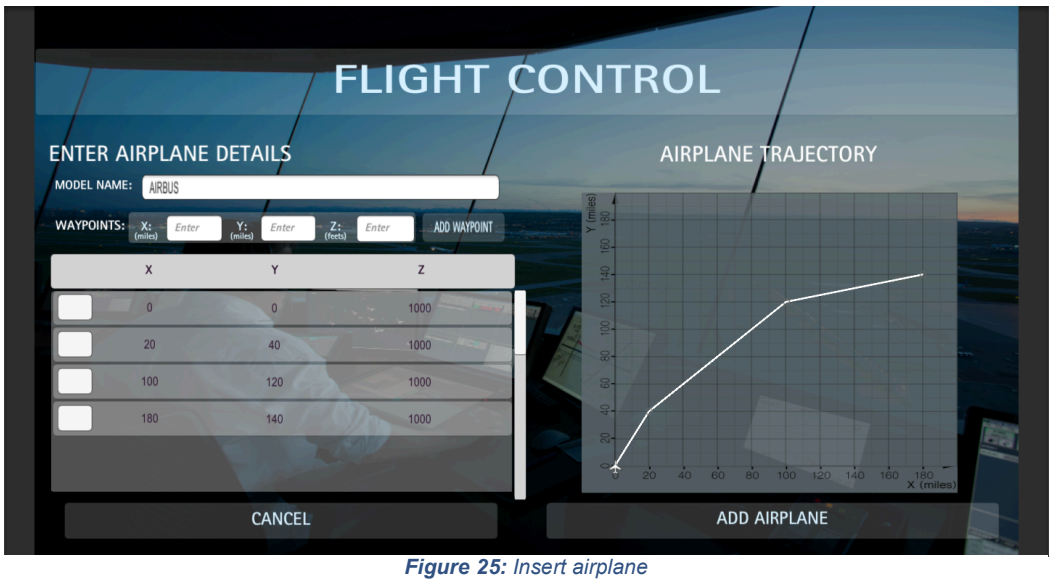

Figure 25: *Insert airplane*

If we need to delete a waypoint, we can simply select the corresponding checkbox and click the delete button. Every time a change is made in the airplane's waypoint, its trajectory gets updated in the map. Once set all this data, we can save the airplane in the system by clicking the Add Airplane button

For updating an airplane, the same interface will be shown, but this time the model name and waypoints will be already filled with the existing data.

6.2. Visualization Scene

The system will create a 2D and 3D environment where it is possible to visualize the flights heading to its destination, and other features such as conflicts and collisions among them. Different aspects can be observed in the visualization depending if it's seen in 2D or 3D, but there are some aspects that are common in both of them.

6.2.1. Common aspects in the visualization

One visible feature is the possibility to accelerate the visualization by increasing the value of the slider. By changing that value, airplanes will arrive faster to its destination.

Collisions and conflicts may happen anytime during the visualization and it should be notified effectively to air traffic controllers. In our system airplanes are painted of three different colors depending on the level of safety:

- Green color: Represents that the flight is safe by the moment.
- Orange color: Indicates that the airplane's long-range detector has detected the presence of another airplane.
- Red color: indicates that an airplane's short-range detector has noticed the presence of another airplane.

Moreover, any type of risks detected among airplanes will be logged in the conflicts window located in the bottom area of the visualization scene.

Users can also choose to finish the visualization before airplanes arrive to their destination by clicking on the top left corner of the application.

6.2.2. 2D Visualization

The first thing we can observe in the scene is a 2D view showing a small part of the flights area, where some flights can be seen from top and their waypoints. In order to better orientate users, a 2D map of airplanes has been designed showing all the airplanes moving and following their trajectories in the top left corner of the application.

In case that users need to see a flight that is not present in the visible area, they can move horizontally and vertically around the flights area using the W (up), A (left), S (down) and D (right) keys. To see the flights from closer or further, it is possible to zoom in and zoom out by using the Z and X keys, respectively. This will let users to be able to observe all the sectors of the flight area.

Figure 26: *Simulation 2D View*

6.2.3. 3D Visualization

When the air traffic control wants to see more details of a specific airplane, we can enable its 3D view by selecting it from the 2D view using the left mouse button.
After doing that, the system will show the airplane from the back, heading to its next waypoint and show its model name, height and speed in a window.

To change the selected airplane's height, the user needs to press the W and S keys to increase or decrease its height respectively.

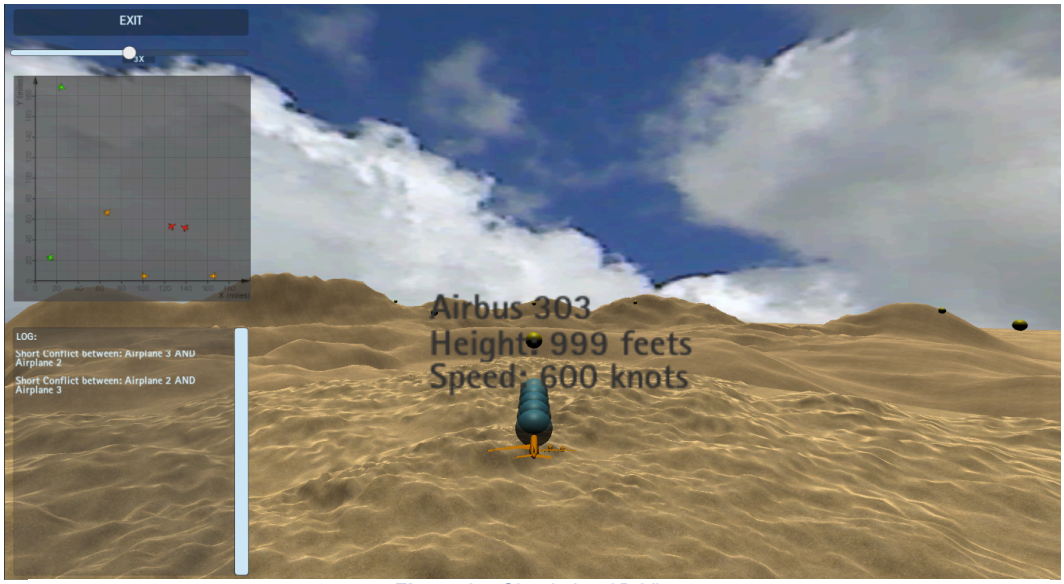

Figure 27: *Simulation 3D View*

6.3. Results Scene

Once finished the visualization, the results panel is shown with the results of the last visualization. These results consist of the visualization time, the total number of conflicts, the total number of collisions, the number of airplanes arrived to destination and, finally, the number if airplanes that are still en route.

Figure 28: *Simulation results*

When analyzed the results of the simulation, the "Go to menu" button can be pressed to arrive again to the menu of the system.