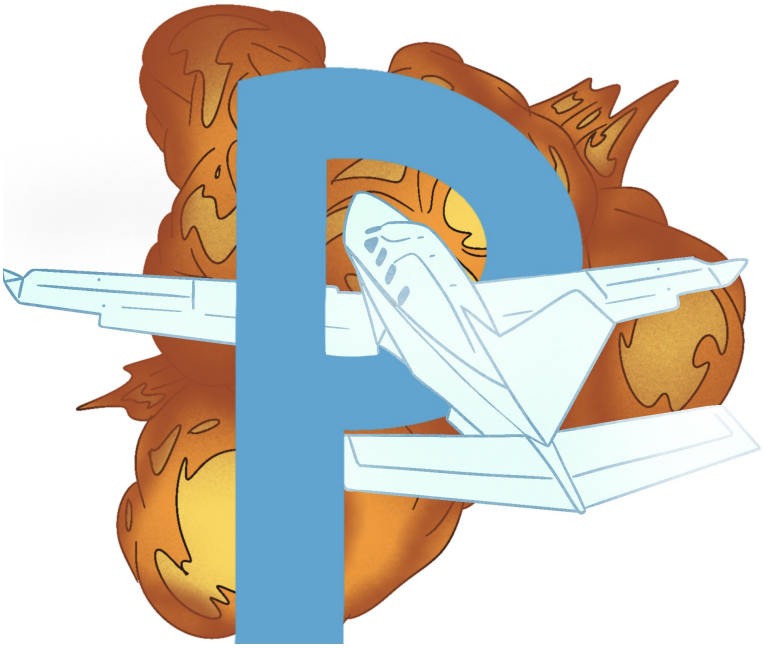
Achille Guerard
Térence miralves
Timothy Pearson
Léopold Tran

Project Pegasus

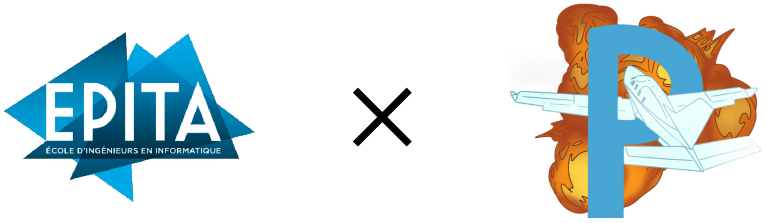
FIRST REPORT



EAF

prépa sup

11th March 2022



Project Pegasus

Contents

1 Introduction **2**

2 Global progressions **3**

3 Task distribution **4**

4 Gameplay modifications **5**

5 Specific progress **5**
5.1 Physics . . . 5
5.1.1 Simple skeleton . . . 5
5.1.2 Lift . . . 5
5.1.3 Drag . . . 6
5.1.4 Angular drag . . . 6
5.2 Multiplayer . . . 6
5.3 AI . . . 7
5.4 Gameplay . . . 8
5.4.1 Targets . . . 8
5.4.2 Cameras . . . 8
5.4.3 Collision . . . 8
5.4.4 Animation/particles effect . . . 9
5.4.5 Pause menu . . . 9
5.5 Modelling . . . 10
5.5.1 3D modelling . . . 10
5.5.2 Textures . . . 11
5.6 Level design . . . 12
5.7 Website . . . 13
5.7.1 Frontend . . . 14
5.7.2 Backend . . . 14

6 Conclusion **15**

Bibliography **16**

1 Introduction

This first report will serve as a follow-up to the book of requirements, as well as a description of the tasks completed since the project's inception. The difficulties that were encountered and the solutions that were adopted will be explored. This report will underline what we have achieved since the beginning of the project and report the issues and mistakes we have encountered and see how we solved them. This report will also highlight the different role our team members have and the co-operation between ourselves.

As a quick remainder, Project Pegasus is a multiplayer air combat game which includes a multiplayer mode as well as a single player mode. We, the EAF team, aim to make this game as realistic as possible through physics and design. Project Pegasus will also feature an AI that will control plane movements.

For this first stage of the project, we implemented the basics of our game. To see the specific achievements and their date, you can refer to the timeline presented in our website. We started off by having a flight simulator ready because it was the top one priority on our list. What would be a plane game without a flight simulation ? Nothing. After that we needed the multiplayer aspect of our game to work because it is the main purpose of the Project Pegasus. Once the multiplayer was ready, we thought that it was time for our plane models to be implemented because we needed to test the link between Blender and Unity. Right after the models, we tackeled down the map, every creative aspect was now done. We were at that point quite ahead of schedule so we implemented a good part of the AI controlling our plane as well as target ballons to show durring the presentation, we also added texture to the models so that they would really be finished. The last thing we did was the pause menu, it was mainly symbolic considering the fact that after the first defense, we could have a short «pause».

2 Global progressions

During those first two month we gathered a lot to talk, brainstorm and debate about our respective visions. These moments of cohesion were crucial to the well-development of our game. We, as the Epita's Air Force team are glad to work and progress together. This section is detailing how well our advancement is going and the problems we encountered during.

To begin with, we thought that the best asset of our game should be the realism it induces. We, thus, chose to make every feature respect the rule of realism. Moreover we tried as much as possible to rely on the book of specifications we handed out 2 months ago. However we are all agree to change feature when it is better to and add new thing that were not specified, we think that flexibility in a project is a great quality to have and we try to be as flexible as possible in the realisation of this project.

Generally speaking, our game is on schedule, the various parts of our game are advancing in synergy which is the most important to us, every feature is working on its own but also with the others. We worked a lot to make everyone work respect the other's, to do this we used GitLab

Throughout the development of our project we have decided to use Gitlab because it enables us to use a system that we already know while discovering new features. Gitlab is the perfect match for us as we can evaluate each others code while working on different branches. The advantage of Gitlab is also that we have access to the website letting us the freedom to easily download the zip files from different commits and different branches.

For the first section we have created the base of our game. The very first step was the physics that allows the plane to fly. This had to be done to match as close a possible real life aircraft flight so a set of functions are used to calculate and then apply the individual forces that allow flight and those that are consequences of flight. The next stage was the creation of the model planes and map. The group had many discussions in order to agree on the visual side of our game. We decided on a larger cargo plane and a smaller spitfire like plane. For the map, we decided on a relatively flat forest terrain similar to some woodlands found in eastern Europe. Therefore, the map is primarily covered in trees which allows us to use minimal textures for the ground. The next advancement was creating the multiplayer. We have created a version that involves two planes that are able to fly around the map and shoot at one another reducing the player's health shown on a health bar. The key to having a functioning multiplayer was to ensure that game objects are instantiated and animations are visible on both players screens. This includes the planes themselves, bullets and explosions.

Task	Prevision	Actual progress
Multiplayer	60%	70%
AI	30%	50%
Gameplay	40%	40%
3D modelling	50%	70%
2D modelling	50%	70%
3D animation	0%	0%
Physics	70%	85%
Particles effect	20%	20%
UI	20%	20%
Level design	30%	30%
Tutorial	0%	0%
Sound effect	0%	0%
Music	0%	0%
Website	80%	80%

3 Task distribution

Each team member either had an idea of what skills they wanted to improve thought the project or was already advanced in a certain field. This made the assignment of tasks fairly quick and easy.

Tim is passionate about physics and mechanics however lacks experience with Unity and programming compared to the rest of the group. Furthermore, he had finished his last two years of highschool focusing on physics and mechanics so it only made sense for him to take the role of designing the flight physics. The tutorial for the game is another take that he decided to implement which will further build his knowledge with unity. Web design was completely new to him however he decided to take the opportunity to build the website for the project knowing that Leo would be able to help with any issues along the way.

Leo is an experienced programmer especially in the field of web design. Since Tim decided to build the website, Leopold applied his skills creating the LateX documents for the whole project whilst aiding Tim with the website. In order to differ from websites, Leo wanted a part that involved only backend programming, even though he had no prior knowledge in networking and artificial intelligence he expressed his interest in both roles. He has always been curious how artificial intelligence works in the fascinating ways it does. In our game, the AI and physics work hand in hand as the enemy plane uses the same script as the player however instead of the user input controlling the plane, a further script sets the inputs in order to follow the plane. Taking on the AI would also give Leo some extra insight to the physics aspect of the game.

Achille has also built up his programming skills before joining EPITA. He is experienced in Python, HTML, Javascript and CSS meaning he would also be great help with the website. Aside from the knowledge he has developed prior to the project, Achille took an interest in the creative and artistic side of the game to learn about new and different aspects of computer science. The 3D modelling and animations was the perfect option for him to open up to new subjects. Achille spent a lot of time learning how to properly use Blender making him more productive than ever in modelling.

Terence has plenty of experience in game development using Unity as well as unreal. He is also an air combat game enthusiast having spent many hours playing War Thunder. This meant that game play was an ideal task for him. Furthermore, he has become very familiar with how aircrafts should fly in video games. This proved extremely useful when it came to testing the physics and making sure that it behaves as it should. On the other hand, he chose to take the level design task as he has always wondered how games create realistic and visually appealing terrain for the player. The roles assigned to Terrence have given him the opportunity to explore new fields while still using his skills to benefit the group.

We believe that the diversity of our team is what makes it strenght, furthermore we all value communication above all. Indeed, we spend time during our weekly meeting to discuss what has been done during the week, set goals for the next week. Additionally, we take the time to explain different aspects of the project to each other so that this project can grow us as developpers, designers, and teamates.

4 Gameplay modifications

First of all we think, as a group, that flexibility is the best asset that we can have. That is why we allowed the game to change slightly from what we had originally imagined. In fact, we had a lot of ideas in the first weeks of the project and might have been too ambitious. This is why we decided unanimously to remove the inside of the plane point of view. We did not measure the amount of time that it would take and we think that removing that part will enable us to focus on other aspects of the game. This time could be put to good use, we could implement plenty of other features each improving our game in the best ways. For example, we are planning on adding more gameplay and more modelling. Moreover, the 3D animation of the work is going to be transferred to animating explosion as well as animating smoke through an engine. To be perfectly clear, this slight change is not changing our main focus, the inside of the plane point of view was a good feature but we think that relocating our energy on something else is clearly worth it.

Firstly it will be through the AI implementation, we are trying to make it as «intelligent» as possible so that the difficulty of the game can be a whole spectrum. Secondly, the 3D animation is going to be as realistic as possible, the explosions feature was not at all planned in the beginning but we think it will give our game a lot more sense and realism. Moreover, the various different parts of the plane will have a separate life bar, this life bar will need to match with both the flying system (for the pilot), the shooting aspect (for the gunner) and the plane aspect. For example engines will look different if they are at full health or mid life, many effects will be added, including for example particles to make a realistic smoke.

5 Specific progress

5.1 Physics

5.1.1 Simple skeleton

The first goal we pursued was creating a simple version of a flight simulator which does not include aeroplane physics. Initially, all we wanted was a simple model that experienced a force towards the nose of the plane when the spacebar was pressed and a torque for both the pitch and roll using arrow keys. Then we started creating the functions that are needed for the complex physics. Before writing any code a lot of research was needed to understand all the factors that make a plane fly as well as the consequences of flight.

5.1.2 Lift

The pitch and yaw angle of attack is the angle between the forward local velocity and the vertical local velocity. The yaw angle of attack is the angle between the forward local velocity and the horizontal local velocity. They are used to calculate lift forces from the rudder and the wings.

$$L = C \times \frac{r \times V^2}{2} \times A$$

With C being the lift coefficient.
In the formula for lift, the lift velocity (V) is the local velocity projected on the YZ axis. The wing area (A), air density (r), and the lift coefficient (Cl) are all dependencies that do not match the model exactly. Instead, the liftpower variable is a well tested estimation that gets multiplied by a second variable, lift coefficient, that relies purely on the pitch angle of attack. The calculated angle of attack is evaluated by the animation curve below:

The lift coefficient peaks when the angle of attack at 30' and has a negative effect when the angle is less than 0'. The lift coefficient ,lift power and lift velocity squared are multiplied to create a 3 dimensional vector. Induced drag is a result of vortexes being created at the tips of the wings and is also known as parasite drag.

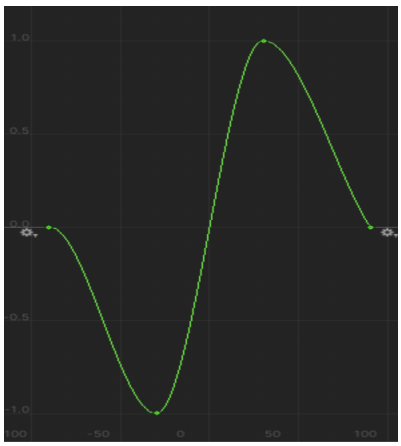


Figure 1: Animation curve of Angle of attack.

$$Cd = \frac{CI^2}{\pi \times AR \times e}$$

As a result, the actual lift force is angled backwards and not vertical. It is proportional to the lift coefficient squared and lift velocity and a constant that is chosen by the forward local velocity evaluated by the drag animation curve. We struggled with understanding what animation curves were and how they helped to calculate lift and drag. Furthermore, we spent a couple of hours trying to find an error in the code while all that was needed was a translation of the animation curve by one unit up in the y direction.

5.1.3 Drag

Drag is the force due to air resistance and causes the plane to slow down. It is proportional to the velocity squared of the plane. Initially we thought that drag would only need to act in the backwards direction of the plane but found that it should be a 3 dimensional vector so that it is also apparent when the plane is falling flat. So we split the velocity into x, y and z components and calculated the drag coefficients independently with the drag animation curves and multiplied each component by a coefficient depending on the surface area from the corresponding orientation. Furthermore we took into account that the drag may change whether the plane is moving forward or falling backward. The final vector was applied as a force ignoring the mass of the plane.

$$FD = \frac{1}{2} \times pv^2 \times Cd \times A$$

5.1.4 Angular drag

The default angular drag in unity had some limitations so we created another function that allows us to fine tune the result. Similar to drag, it is proportional to the angular velocity squared. The calculated drag is then multiplied but another hand tuned variable and torque is then applied in the opposite direction to the angular velocity.

5.2 Multiplayer

We choose to build our multiplayer system relying on Mirror because it's a free and open source networking package for Unity that matched all our demands. The way our network system works is that instead of having all players log into a server we have the main player, the pilot, being the host, thus both a server and a client. The other players connect to the host. Our game contains a network manager that takes a

prefab and instantiates a new object every time a new client logs in. Two main problems emerged.

The first problem was that we had to differentiate players so that the controls of one player did not affect the controls of the other. In order to do this we separated the script of the player into multiple parts. All players rely on two different scripts, one called Pilot that enables the user to control the plane, the second being the weapon system that enables the player to shot and instantiate a bullet from the Bullet class. Both classes inherit from the NetworkBehaviour class enabling us to check if the player is the Local Player, if he is then his inputs are updated else nothing happens.

Another issue was having a new camera for each object, this meant that we had to instantiate a new camera every time a new client logs in. But how do we know which camera belongs to who? The solution to this problem is the following lines of code :

```
if (!isLocalPlayer)
{
    camera2.gameObject.SetActive(false);
}
```

In the start function we check if this is not the player, then we disable all other cameras.

The last issue we had with networking was that of other players interacting and instantiating objects. We noticed that what happened on one screen did not happen on the others. To fix this we dealt directly with the server by sending messages to it. We built one command that calls a RPC function. This RPC function deals with the bullet as you would expect but as it deals with the server every player is effected by it. This solved the issue of shooting bullets, being damaged and crashing.

We are very happy with the multiplayer part not only because it works the way we expected but also because it allowed us to enrich our knowledge of software development.

5.3 AI

The AI's primary objective is to aim its nose towards the target. This one behavior is responsible for the majority of the dogfight's atmosphere. When the player is in front of the AI, the AI turns repeatedly to get behind the player, which has the unintended consequence of dodging the player's assaults.

It's not difficult to figure out which way to go. Because the plane is most maneuverable when pitching up, the AI seeks to align itself by rolling in the direction of the goal.

The steering function begins with the AI aiming for a specific spot. This is either the target plane or the computed lead to hit the target with cannon fire if in cannon range. The plane's local space is converted into this position.

It makes no difference where the two planes are in world space. The target plane's position in local space is used to compute steering.

To get pitchError, the error value is flattened into the YZ plane. This is provided to Vector3.SignedAngle, which calculates the angle required to pitch up or down and point at the destination. The plane receives a pitch input of 0 when the target is immediately in front of it.

If that angle is a large pitch down, it gets converted into a pitch up maneuver. For example, if pitchUpThreshold is 15 degrees, then when the target plane is more than 15 degrees below AI plane, the AI will always try to pitch up. If the angle is less than the threshold, it will pitch up or down normally. This avoids situations where the plane would try to pitch down by a large amount, which would result in a slow maneuver.

rollError is calculated from the error on the XY plane. However this rollError is used to calculate either the roll input or the yaw input. If the AI plane is pointing within a small angle, fineSteeringAngle of

the target, then it will aim using the rudders for finer control. This gives the AI the ability to aim it's cannon.

If the AI is more than fineSteeringAngle, the roll error will cause the plane to roll, to allow a large pitch up maneuver. The roll error is 0 when the target is directly overhead.

Finally, the targetInput is clamped on each axis to prevent something like an input of (360, 0, 0). We use Vector3.MoveTowards to add a slight delay to the inputs generated by the steering function.

5.4 Gameplay

5.4.1 Targets

After completing the shooting side of the game, we wanted to have something to shoot at. Therefore we decided to go for red hot air balloons as floating targets.
The challenging side of the target's balloons was to allow communication between the bullet and the balloon as both objects destroy themselves at the contact of any other objects. It was difficult getting the message to send to target hat it had to be destroyed.

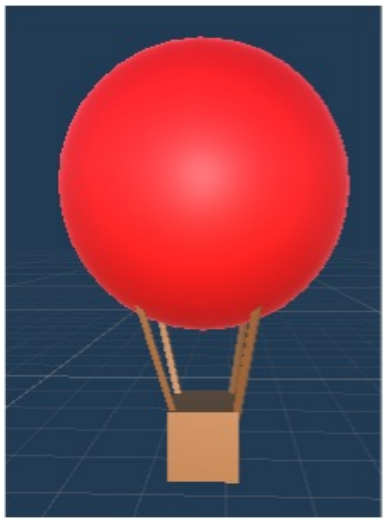


Figure 2: Ballon prefab.

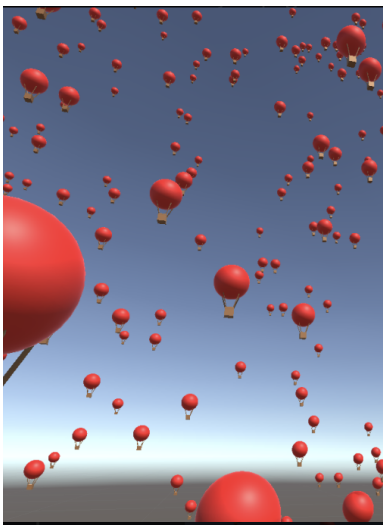


Figure 3: Ballon spawner

5.4.2 Cameras

The pilot's camera is a vital particle of the game as it will be responsible for giving the player a feeling of flying an actual plane. To do this we added different cameras. One situated at the rear of the aircraft which allows the player to see the entirety of the plane that will ensure the player can follow the movement when performing chaotic maneuvers and the second is a cockpit view that will be used as a more precise and accurate view for aiming at the targets. The difficulty here was to smoothen the rear camera as it is a child of the plane because of the server which changes the way we had the camera sett up in the beginning.

5.4.3 Collision

As we were testing our plane, we came across a problem which needed to be solved. The plane can go through the ground. After a discussion with the group, we agreed that, for the moment, the plane would explode if it touches the ground and that perhaps the player would re-spawn with less health as to not frustrate the player with an instant death. For now the only collision the plane has is with the ground so implementing this was fairly easy since we have both the y position of the terrain and the plane so once the plane's y axis passes below the terrain we only destroy the plane and launch the explosion animation at its coordinates.

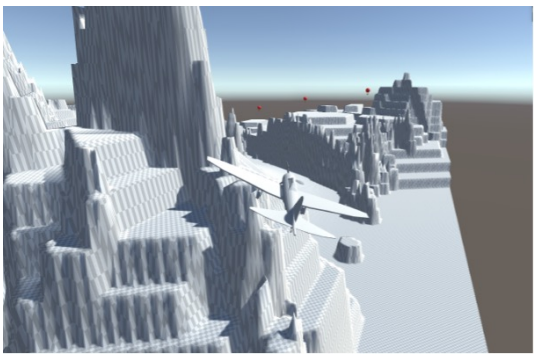


Figure 4: Rearview

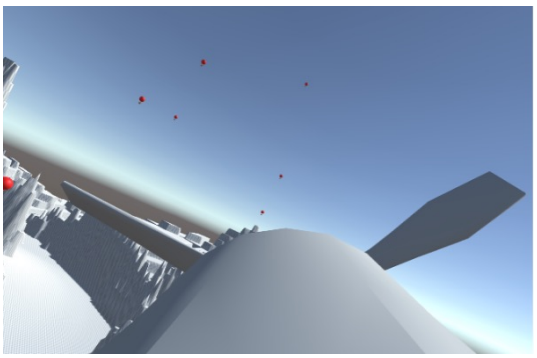


Figure 5: Cockpit view

5.4.4 Animation/particles effect

For the collision we wanted to have an explosion so we started to research how it could be done. At first, we tried to create them ourselves with a unity particle system but it was too difficult and not up to our standards so we decided to use some explosions on a free unity asset store pack as a temporarily replacement. The explosions are used by the plane when it touches the ground and by the balloons when they are hit. We also used some really basic animation for the movement of the propeller, a single script which rotates the propeller infinitly.

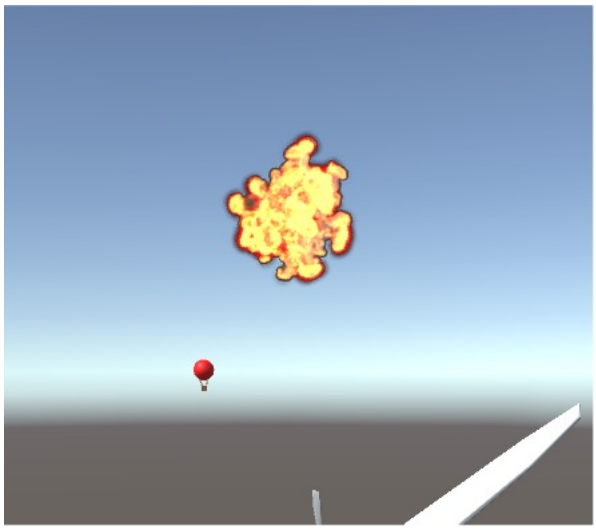


Figure 6: Explosion - Particle effect

5.4.5 Pause menu

Every game has to get a pause menu for the player to change settings or to take a break after some hectic match. There are several different ways to implement a pause menu. The first being to stop every thing in the game while on the menu behaving like a true pause and the other being to let the game continue in the background while the player is unable to move. The first is the most common way used in multiplayer games knowing that if one player goes to his menu the game will stop for everyone else which can break the immersion really quick but as our game is a cooperative game, we figure that it wouldn't be a problem as the players are working as a team and we also know that the game play can be really tense so giving the possibility for the players to stop and figure out a way to complete the level is what we decided on. The pause menu is really basic at the moment composed of only one button to resume as there is no way to change the setting or to get back to the menu just yet we as didn't see it as a necessity.

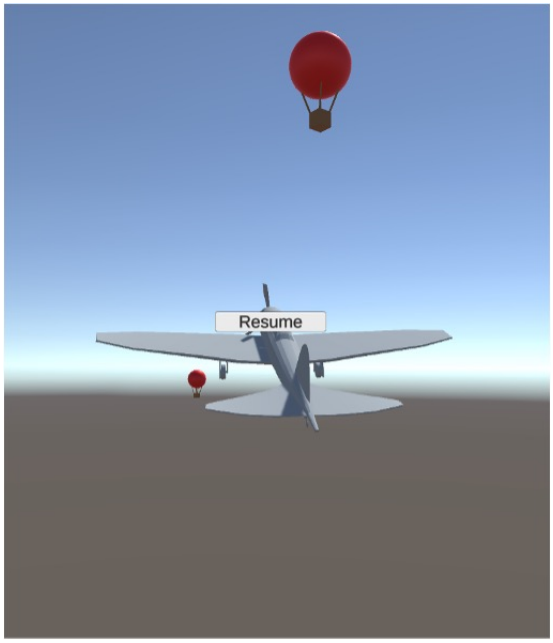


Figure 7: Pause menu

5.5 Modelling

One big part of the modeling was to understand the shapes and forms of things, for basic objects this is easy, but when it comes to planes, turrets or even propellers, We needed to find a way of making this process easier. The hard part of modeling is to be careful for all three dimensions because our brain is used to work, especially with a computer, in only 2 dimensions. One solution to this problem that we applied during the modeling of both plane is the use of 2D images as reference. In fact importing 2D images to blender is easy and it helps a lot. To represent the three dimensions that we need for every model we used 3 images. each one relying on a plan. XZ for the front view, YZ for the side view and XY.

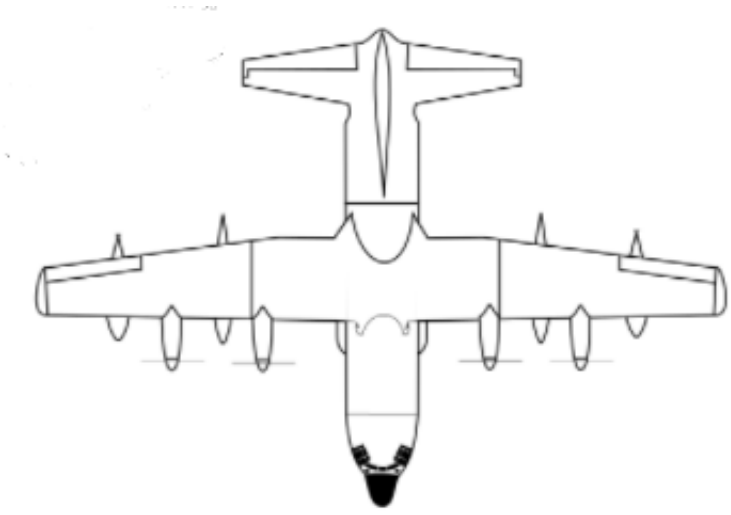


Figure 8: Reference image

5.5.1 3D modelling

During the development of our project we needed to put a lot of effort in the 3D modeling because we want our game to look as realistic as possible. Indeed, after the physics part that made our flying real, we modeled our planes so that they look stunning. Moreover, we decided to design our models ourselves instead

of taking them from the internet, it will give our game the creative aspect that we want. As said in the book of specifications, we used the software Blender in the version 3.0.0. You can find here a few examples of things we had to model.

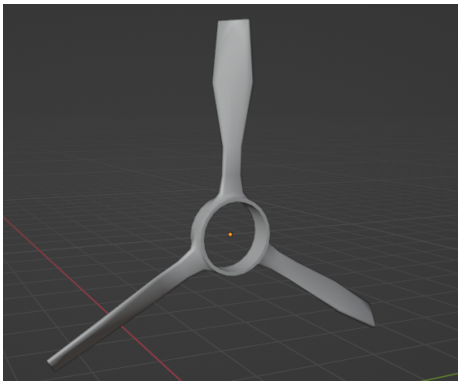


Figure 9: Propeller

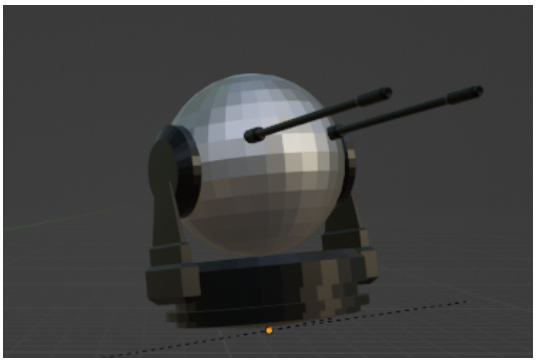


Figure 10: Turret

In fact, even though the main part of the work is to model planes, a lot of other feature is required, for example the turret you can see here.

This part came with loads of difficulties because learning how to use a new software is always a struggle, here are some examples :

- Understand how to use every basic blender feature to model efficiently (move, scale, rotate, extrude...)
- Learn the basic shortcuts to have a better workflow.
- Understand the importance of all 2D viewpoints and reference images.
- Learn how to create, import and apply texture.
- Exporting blender files to unity.

The solution to each of these problems was research. In fact, every solution to our problem was online. Furthermore, the solution was also hard work because to memorise every useful shortcut or technique in blender the only way is to use it a lot. The inspiration and finding references part was also an important part of the modeling. In fact, the use of references images makes the modeling a lot easier because then you don't have to understand the 3D forms of each and every element of the model, you can instead use the combination of three 2D images (top view, side view and front view). This part of the work required communication between all members of the team because we wanted to have the same vision of our game. Even thought only one person was in charge here, the improvement in design and modelling came from everyone's participation.

5.5.2 Textures

After the 3D modelling was done on the first two planes, we decided to add the textures so that both planes would be completed for the first demonstration of the game. Texturing is particular and different for every kind of object since from the organic skin of an apple to the metallic paint of a plane there's a whole world of possibilities. To do the plane texture we insisted on two main parameters. Firstly, and obviously, the metallic component was put to at least 70% on every texture. Secondly, the roughness was set to different degree depending on parts of the plane. For example, the metallic aspect of the propeller comes from the fact that its roughness is at around 30% which means it reflects a lot of the light it receives. This parameter is often called albedo. For the texturing we used the shading module of blender and this new module was sometimes hard to understand. Here are some example of our struggles :

- Understand the differences between edit mode and object mode in this module and how it is best to use both.
- Learn how to import textures to unity and which files of textures are required.
- Study all the parameters that can change the textures and identify the useful ones in our case.

As texturing is a more technical thing than modelling, the online resource was harder to understand at first but it was in the end very useful. To solve these problems we once again did a lot of research and spend a lot of time trying different setups for our textures. The coordination of the team played also a big role in the realisation of this task as Achille and Térence, respectively responsible for blender and unity, worked together to test models and find the best way to implement them.

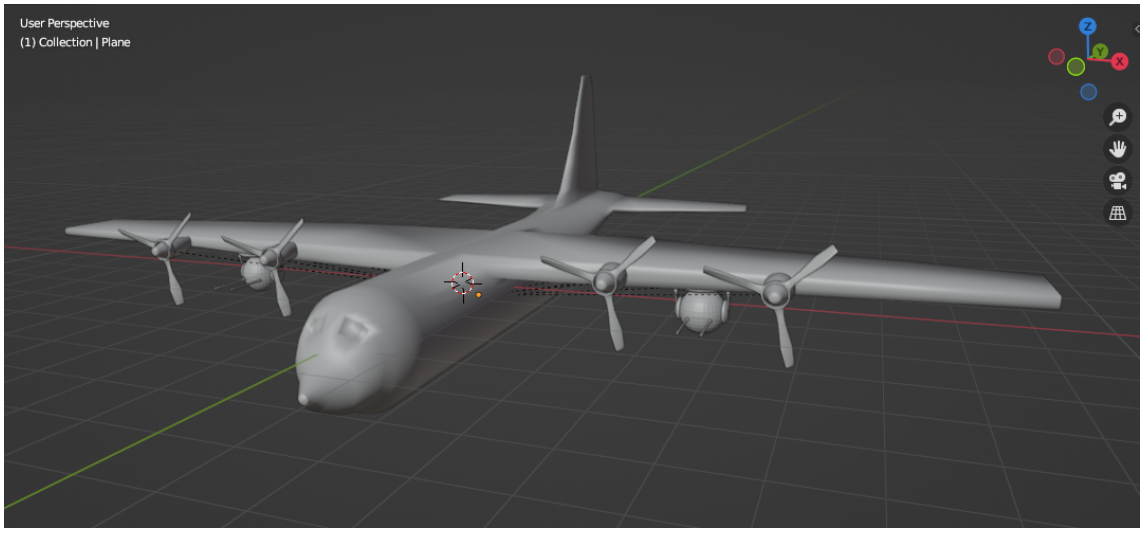


Figure 11: Naked cargo plane

Figure 12: Textured cargo plane

5.6 Level design

While making the map many new things appeared which came with many new problems. These problems were based around getting the map to appear as we imagined it. Indeed, in order to give the map a credible look, multiple layers of different textures are needed on the ground which requires logic. You have to take into account that some areas are more likely to be rocky because of their slope or perhaps they are more muddy because they are located in the middle of different trees or in a dip. Keeping an eye on the scale is vital and to do so we placed a 2m high cube to act as a human size reference. So thought the creation of

the map we had to be mindful of scaling, keeping a credible appearance and sticking with the original idea. These challenges proved themselves to be harder than expected however, there was a comforting aspect of being aware of every detail on the map and that these details were accurate enough to meet the groups expectations.

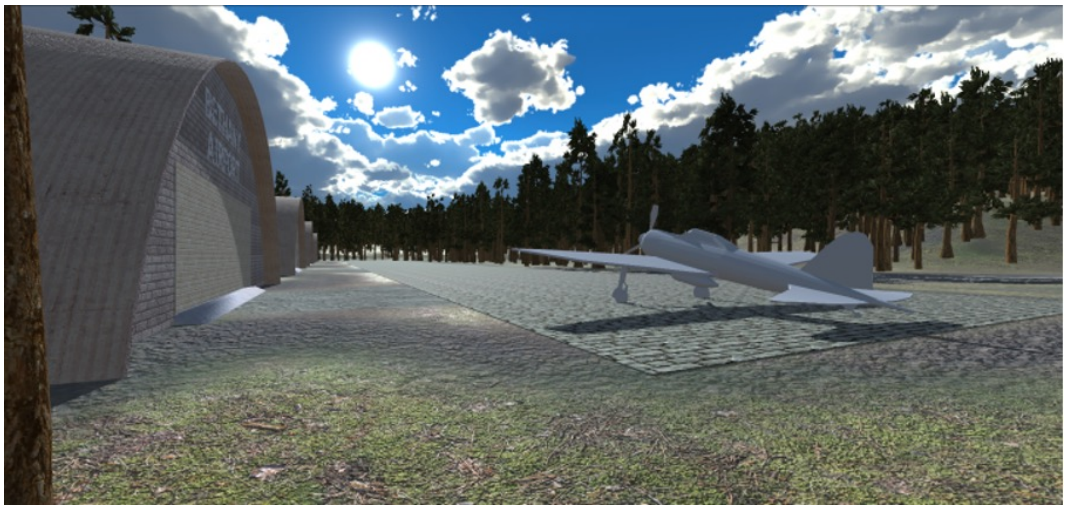


Figure 13: Close up of the map

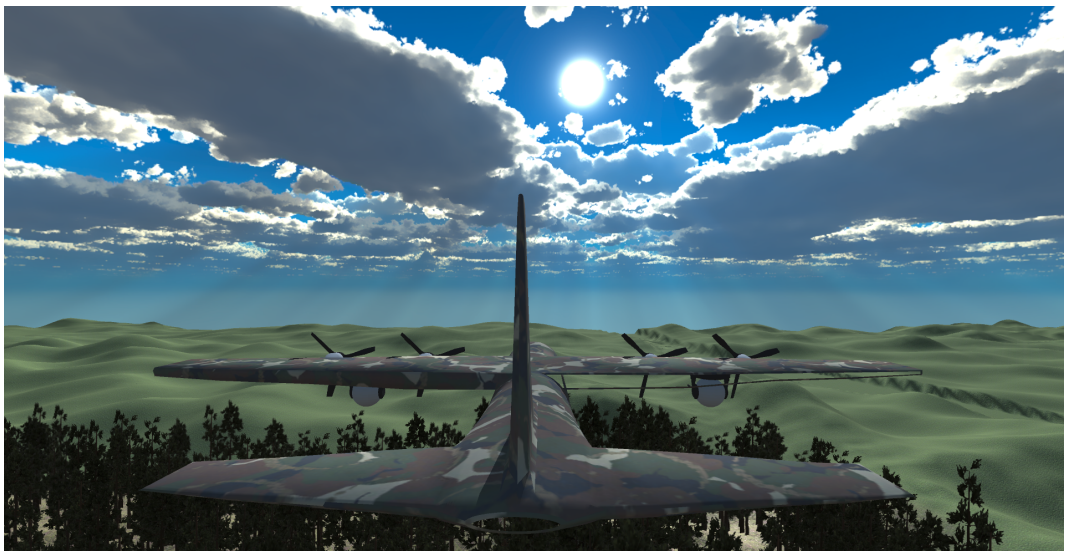


Figure 14: Global view of the map

5.7 Website

In addition to the game we developed a website that would allow our team and anyone who wishes to see the progress of our project. The website is also a place where one can download our book of specifications, reports, and latest version of our game. The website also presents our game and the team members in it.
The website is available at https://projectpegasus-eaf.github.io/.
We decided to host the website on GitHub pages because it allows us to easily access files while having a free domain to host it.
The website is an important tool for visitors to learn about our project and play the game. As a result, we created four pages containing critical project information. There's a home website, a project page with two distinct timelines. One of them is the timeline regarding EPITA's schedule (creation of group, book

of specifications, first defence, etc...) and contains our project's executable, and some documents like the book of specifications and the first report. The other timeline is based on our progress and allows us to have an overview of the progress we have made since the start. . There are extra sections dedicated to the team, as well as credits for the software used.

5.7.1 Frontend

The website was created with the use of HTML and CSS for the theming and layering. The frontend took a long time to create. Indeed, we elected to create a website from scratch, without using any templates or assistance such as Bootstrap. We used CSS grid elements in order to keep everything neat and tidy not only on our computer but on every device. We used vw and vh units instead of classic pixel so that the website stay's responsive no matter the size of the screen. We strongly relied on the documentation of HTML and CSS and used Firefox Developper Edition to ease the development.

5.7.2 Backend

The backend is handled by Github Pages, which provides hosting for the website, server backend, and error handling, as indicated previously in the book of requirements. The sole prerequisite for launching the website was that it be stored in a Github repository, which can be found at https://github.com/project-pegasus/website/.

Figure 15: Homepage

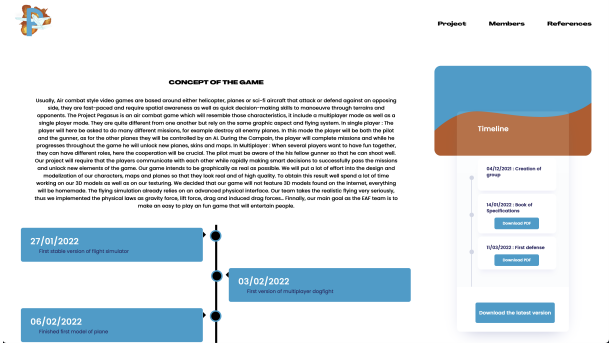


Figure 16: Project page

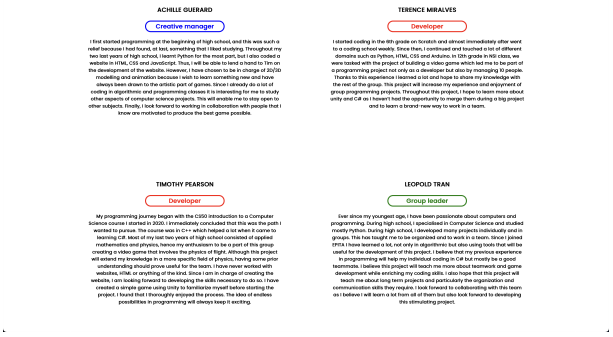


Figure 17: Team page

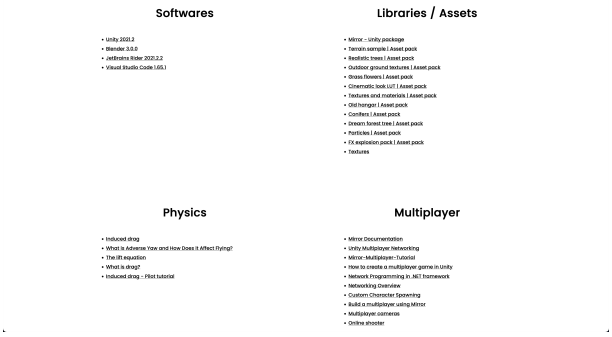


Figure 18: References page

6 Conclusion

Despite some minor setbacks that have been successfully overcome, the team is currently on track with all tasks.
This indicates that roughly a third of the project has been completed.

To summarize: we have a flight simulator that follows realistic properties, a network system that allows two players to fight each other. On the front end we have a cargo plane and a WWII plane both modeled from scratch, each of them having their own textures. We also have an AI capable of following an airplane. Additionally we have a nearly-homemade map and a game mode with a target generator. Finally we have a website coded from scratch that is host to our resources.

By the next defense we hope to modeled another jet fighter as well as another map while continuing to build on to this one. The flight simulator will also be improved always keeping the realism of the game. We hope to have the AI be able to shoot follow a player, avoid the ground and shoot down a player. We plan to have the multiplayer part we able to enable team cooperation. We also plan to have a turret being able to move around and shoot in that direction while developing the UI.

In the future, we plan to build on to the realistic feature of the flight simulator and develop the network part so that multiple players can control different part of the same plane. We believe that our foundations are solid and that we have found a schedule and way of operating for our team. Until the last defense we plan to build on top of what we already have in order to produce a realistic and challenging multiplayer flight simulator and air combat game.

Bibliography

[1] *A.I. learns to fly*. URL: https://www.youtube.com/watch?v=D5xX6nRWDko.

[2] *Ace Combat 7: Skies unknown*. URL: https://en.wikipedia.org/wiki/Ace_Combat_7:_Skies_Unknown.

[3] *Air combat*. URL: https://en.wikipedia.org/wiki/Air_Combat.

[4] *Aircraft principal axes*. URL: https://en.wikipedia.org/wiki/Aircraft_principal_axes.

[5] *Bartendu*. URL: https://areas0.github.io/website/index.html.

[6] *Blender 3D modeling : The ultimate collection*. URL: https://conceptartempire.com/blender-modeling-tutorials/.

[7] *Create a map in unity*. URL: https://docs.mapbox.com/help/tutorials/create-a-map-in-unity/.

[8] GameDevLessons. *Flight SIM control, Terrain Basics, Chase Cam, skybox*. URL: https://www.youtube.com/watch?v=lCulq9J0Y9E.

[9] *GitLab*. URL: https://en.wikipedia.org/wiki/GitLab.

[10] *Gitlab Documentation*. URL: https://docs.gitlab.com/.

[11] *Host (network)*. URL: https://en.wikipedia.org/wiki/Host_(network).

[12] *Local area network*. URL: https://en.wikipedia.org/wiki/Local_area_network.

[13] Renan Oliveira. *How to create a multiplayer game in Unity*. URL: https://gamedevacademy.org/how-to-create-a-multiplayer-game-in-unity/.

[14] Dwight Pavlovic. *Video game genres : HP® Tech takes*. July 2020. URL: https://www.hp.com/us-en/shop/tech-takes/video-game-genres.

[15] Unity Technologies. *Setting up Unity Multiplayer*. URL: https://docs.unity3d.com/Manual/UnityMultiplayerSettingUp.html.

[16] Unity Technologies. *Unity Collaborate*. URL: https://unity.com/fr/unity/features/collaborate.

[17] Vazgriz. *Creating a Flight Simulator in Unity3D*. URL: https://vazgriz.com/503/creating-a-flight-simulator-in-unity3d-part-3/.