CTIS411 SENIOR PROJECT-I SOFTWARE DESIGN DESCRIPTION (SDD)

Winged Patrol: VR-based Firefighting Aircraft Simulation Game

Team 17

Emre Bener

Ömer Faruk Eş

Engin Kaan Görgün

Mustafa Oğulcan Tekiner

Project Supervisor

Okyay Say

FALL 2020

Table of Contents

Table of Figures

1. ANALYSIS MODEL & PLANNING

1.1. FUNCTIONAL REQUIREMENTS (Use Case Diagram)

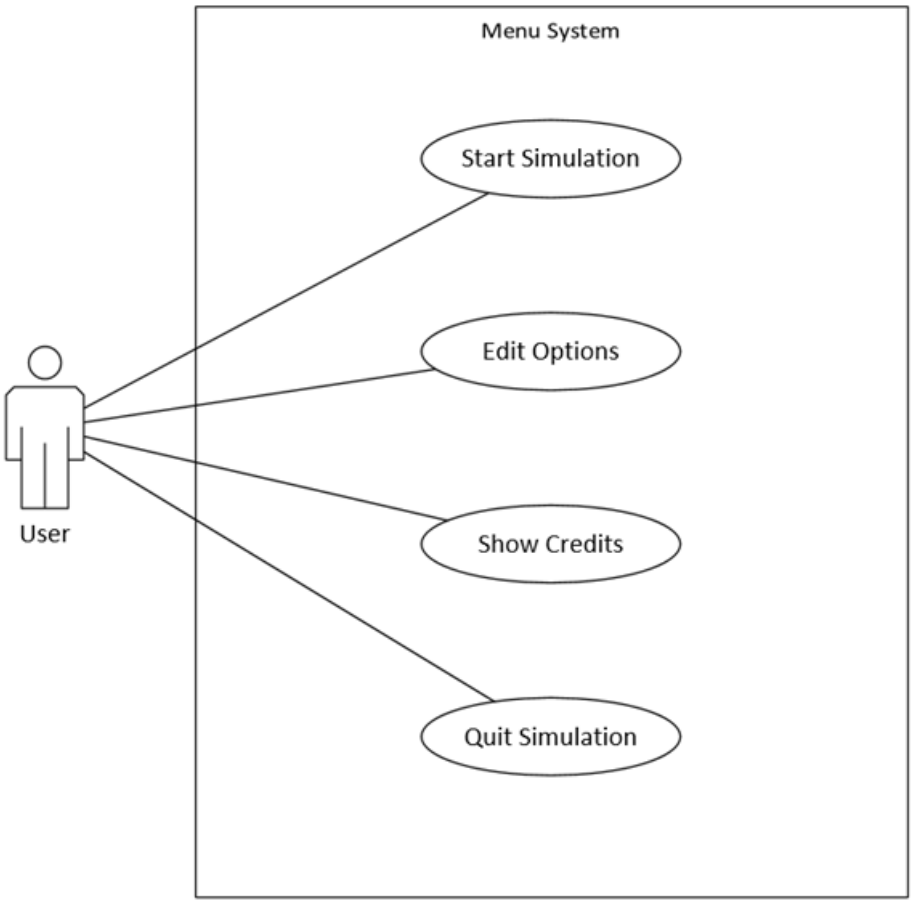


Figure 1: Menu System Use-Case Diagram

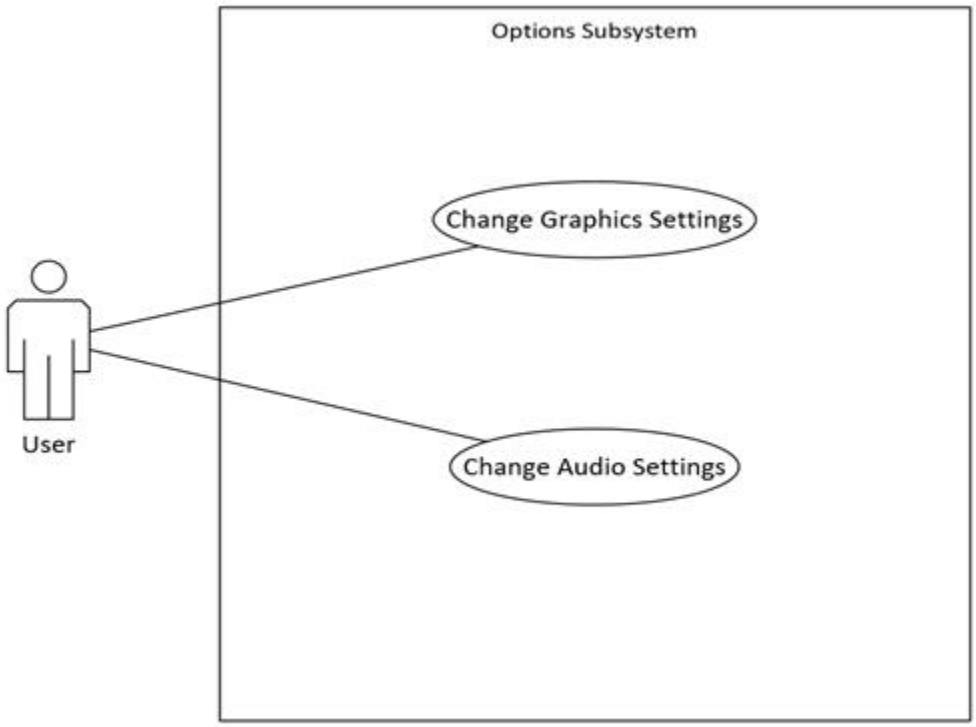


Figure 2: Options Subsystem Use-Case Diagram

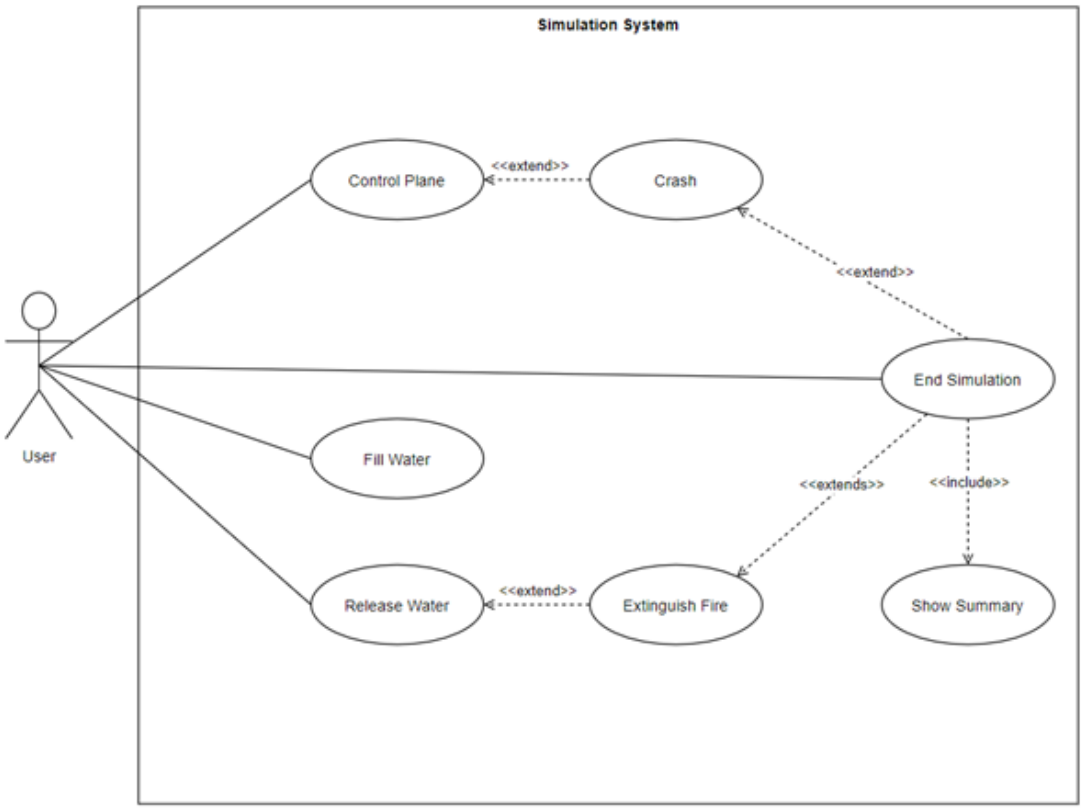


Figure 3: Simulation System Use-Case Diagram

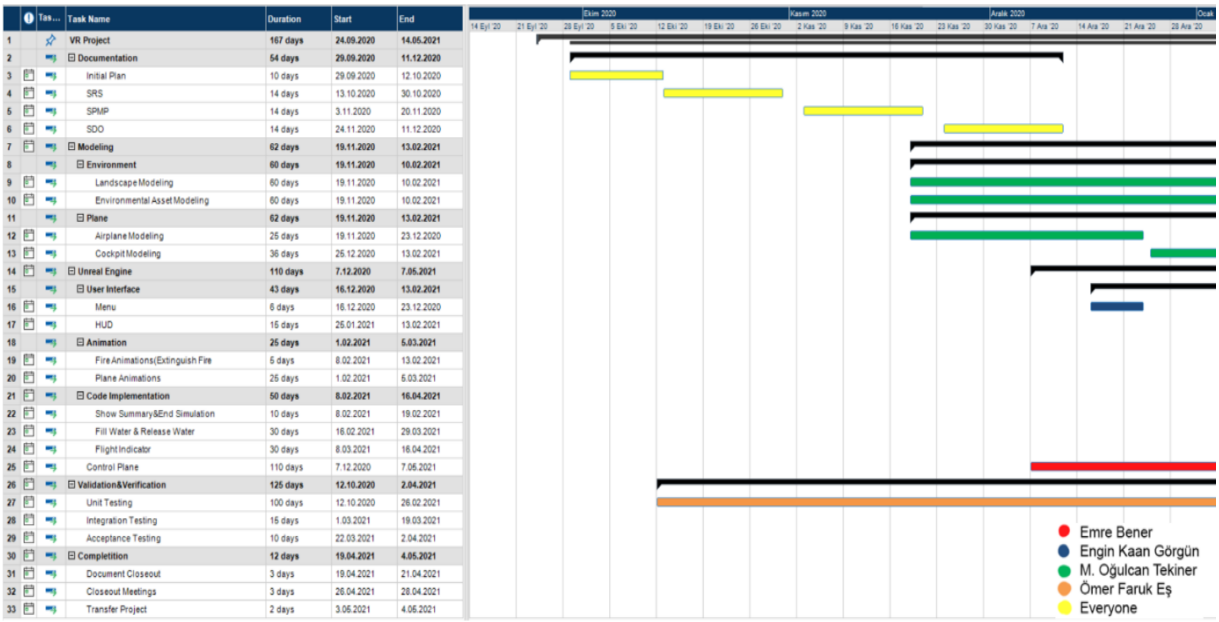
1.2. NON-FUNCTIONAL REQUIREMENTS

Property	**Requirement**
Performance	Each frame shall take no longer than 16 ms(milliseconds) to render Simulation game shall take no longer than 1 minute to start Simulation shall perform at 60 frames per second.
Reliability	Simulation game shall run continuously at least 12 hours before experiencing a failure. Simulation game shall run continuously for at least 4 hours before experiencing a glitch.

Figure 4: Non-Functional Requirements

1.3. SOFTWARE INCREMENTS

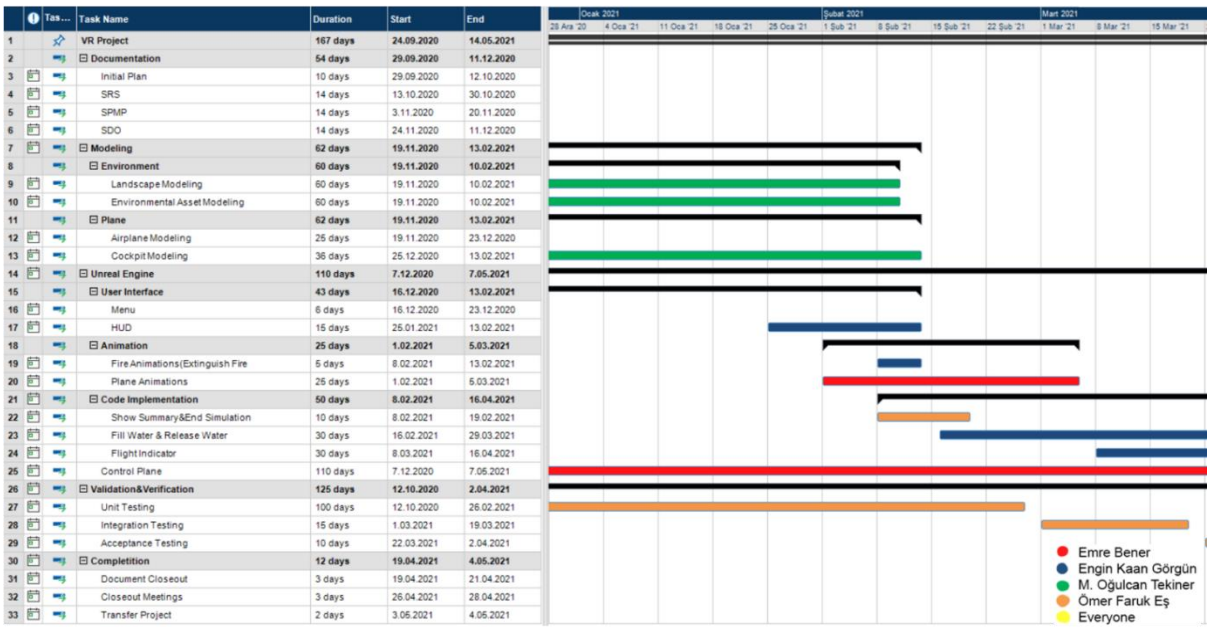
Schedule for the First Increment:



	Task Name	Duration	Start	End
1	VR Project	167 days	24.09.2020	14.05.2021
2	Documentation	54 days	29.09.2020	11.12.2020
3	Initial Plan	10 days	29.09.2020	12.10.2020
4	SRS	14 days	13.10.2020	30.10.2020
5	SPMP	14 days	3.11.2020	20.11.2020
6	SDO	14 days	24.11.2020	11.12.2020
7	Modeling	62 days	19.11.2020	13.02.2021
8	Environment	60 days	19.11.2020	10.02.2021
9	Landscape Modeling	60 days	19.11.2020	10.02.2021
10	Environmental Asset Modeling	60 days	19.11.2020	10.02.2021
11	Plane	62 days	19.11.2020	13.02.2021
12	Airplane Modeling	25 days	19.11.2020	23.12.2020
13	Cockpit Modeling	36 days	25.12.2020	13.02.2021
14	Unreal Engine	110 days	7.12.2020	7.05.2021
15	User Interface	43 days	16.12.2020	13.02.2021
16	Menu	6 days	16.12.2020	23.12.2020
17	HUD	15 days	25.01.2021	13.02.2021
18	Animation	25 days	1.02.2021	5.03.2021
19	Fire Animations(Extinguish Fire	5 days	8.02.2021	13.02.2021
20	Plane Animations	25 days	1.02.2021	5.03.2021
21	Code Implementation	50 days	8.02.2021	16.04.2021
22	Show Summary&End Simulation	10 days	8.02.2021	19.02.2021
23	Fill Water & Release Water	30 days	16.02.2021	29.03.2021
24	Flight Indicatr	30 days	8.03.2021	16.04.2021
25	Control Plane	110 days	7.12.2020	7.05.2021
26	Validation&Verification	125 days	12.10.2020	2.04.2021
27	Unit Testing	100 days	12.10.2020	26.02.2021
28	Integration Testing	15 days	1.03.2021	19.03.2021
29	Acceptance Testing	10 days	22.03.2021	2.04.2021
30	Completition	12 days	19.04.2021	4.05.2021
31	Document Closeout	3 days	19.04.2021	21.04.2021
32	Closeout Meetings	3 days	26.04.2021	28.04.2021
33	Transfer Project	2 days	3.05.2021	4.05.2021

Figure 5: Gantt Chart: First Increment

Schedule for the Second Increment:



	Task Name	Duration	Start	End
1	VR Project	167 days	24.09.2020	14.05.2021
2	Documentation	54 days	29.09.2020	11.12.2020
3	Initial Plan	10 days	29.09.2020	12.10.2020
4	SRS	14 days	13.10.2020	30.10.2020
5	SPMP	14 days	3.11.2020	20.11.2020
6	SDO	14 days	24.11.2020	11.12.2020
7	Modeling	62 days	19.11.2020	13.02.2021
8	Environment	60 days	19.11.2020	10.02.2021
9	Landscape Modeling	60 days	19.11.2020	10.02.2021
10	Environmental Asset Modeling	60 days	19.11.2020	10.02.2021
11	Plane	62 days	19.11.2020	13.02.2021
12	Airplane Modeling	25 days	19.11.2020	23.12.2020
13	Cockpit Modeling	36 days	25.12.2020	13.02.2021
14	Unreal Engine	110 days	7.12.2020	7.05.2021
15	User Interface	43 days	16.12.2020	13.02.2021
16	Menu	6 days	16.12.2020	23.12.2020
17	HUD	15 days	25.01.2021	13.02.2021
18	Animation	25 days	1.02.2021	5.03.2021
19	Fire Animations(Extinguish Fire	5 days	8.02.2021	13.02.2021
20	Plane Animations	25 days	1.02.2021	5.03.2021
21	Code Implementation	50 days	8.02.2021	16.04.2021
22	Show Summary&End Simulation	10 days	8.02.2021	19.02.2021
23	Fill Water & Release Water	30 days	16.02.2021	29.03.2021
24	Flight Indicator	30 days	8.03.2021	16.04.2021
25	Control Plane	110 days	7.12.2020	7.05.2021
26	Validation&Verification	125 days	12.10.2020	2.04.2021
27	Unit Testing	100 days	12.10.2020	26.02.2021
28	Integration Testing	15 days	1.03.2021	19.03.2021
29	Acceptance Testing	10 days	22.03.2021	2.04.2021
30	Completition	12 days	19.04.2021	4.05.2021
31	Document Closeout	3 days	19.04.2021	21.04.2021
32	Closeout Meetings	3 days	26.04.2021	28.04.2021
33	Transfer Project	2 days	3.05.2021	4.05.2021

Figure 6: Gantt Chart: Second Increment

Schedule for the Third and Last Increment:

	Tas...	Task Name	Duration	Start	End
1		**VR Project**	**167 days**	**24.09.2020**	**14.05.2021**
2		**Documentation**	**54 days**	**29.09.2020**	**11.12.2020**
3		Initial Plan	10 days	29.09.2020	12.10.2020
4		SRS	14 days	13.10.2020	30.10.2020
5		SPMP	14 days	3.11.2020	20.11.2020
6		SDO	14 days	24.11.2020	11.12.2020
7		**Modeling**	**62 days**	**19.11.2020**	**13.02.2021**
8		**Environment**	**60 days**	**19.11.2020**	**10.02.2021**
9		Landscape Modeling	60 days	19.11.2020	10.02.2021
10		Environmental Asset Modeling	60 days	19.11.2020	10.02.2021
11		**Plane**	**62 days**	**19.11.2020**	**13.02.2021**
12		Airplane Modeling	25 days	19.11.2020	23.12.2020
13		Cockpit Modeling	36 days	25.12.2020	13.02.2021
14		**Unreal Engine**	**110 days**	**7.12.2020**	**7.05.2021**
15		**User Interface**	**43 days**	**16.12.2020**	**13.02.2021**
16		Menu	6 days	16.12.2020	23.12.2020
17		HUD	15 days	25.01.2021	13.02.2021
18		**Animation**	**25 days**	**1.02.2021**	**5.03.2021**
19		Fire Animations(Extinguish Fire	5 days	8.02.2021	13.02.2021
20		Plane Animations	25 days	1.02.2021	5.03.2021
21		**Code Implementation**	**50 days**	**8.02.2021**	**16.04.2021**
22		Show Summary&End Simulation	10 days	8.02.2021	19.02.2021
23		Fill Water & Release Water	30 days	16.02.2021	29.03.2021
24		Flight Indicator	30 days	8.03.2021	16.04.2021
25		Control Plane	110 days	7.12.2020	7.05.2021
26		**Validation&Verification**	**125 days**	**12.10.2020**	**2.04.2021**
27		Unit Testing	100 days	12.10.2020	26.02.2021
28		Integration Testing	15 days	1.03.2021	19.03.2021
29		Acceptance Testing	10 days	22.03.2021	2.04.2021
30		**Completition**	**12 days**	**19.04.2021**	**4.05.2021**
31		Document Closeout	3 days	19.04.2021	21.04.2021
32		Closeout Meetings	3 days	26.04.2021	28.04.2021
33		Transfer Project	2 days	3.05.2021	4.05.2021

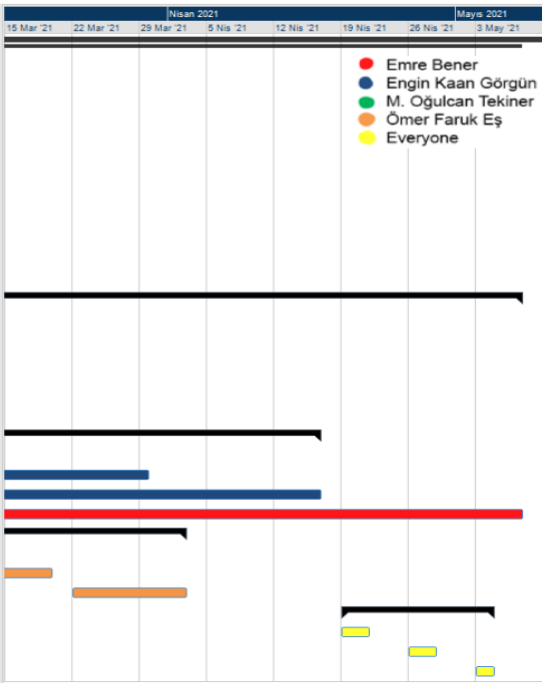


Figure 7: Gantt Chart: Third Increment

1.4. FRAMEWORKS, LIBRARIES, SERVICES, DATABASES, APPLICATION PROGRAMMING INTERFACES (APIs)

We will be developing our project in Unreal Engine, and we are going to make use of Unreal Engine API, which means we will be actively using Unreal Engine's classes, functions and libraries in our code. It will be included in the 1st increment.

We may consider using Substance materials. This will not be included in the 1st increment as currently we are using Unreal Engine's built-in materials.

Landscape will be created in Gaea which will be included in the 1st increment.

The airplane will be modeled in Blender or Maya which will not be included in the 1st increment as we are currently using a placeholder model we found from the internet.

HUD/user interface will be prepared in Photoshop which will not be included in the 1st increment.

Since this project is a VR Simulation Game we will be using Oculus Quest 2 to adapting our game to the VR domain. However, in the first increment, since we do not have access to equipment, any VR related test cases will be done in second and third increments.

2. HIGH-LEVEL (ARCHITECTURE) DESIGN

2.1. LOGICAL VIEW

Unreal Engine Blueprint code uses Unreal Engine API to articulate gameplay mechanics such as Plane Mechanism, Firefighting Mechanism and using & creation of environment and 3D Graphics Design.

Game Metrics will measure the fire propagation, time, score and the fire status from the incoming information from the Unreal Engine API.

The user will interact and use the VR equipment to control the application and from those inputs will be used to calculate Plane's Control.

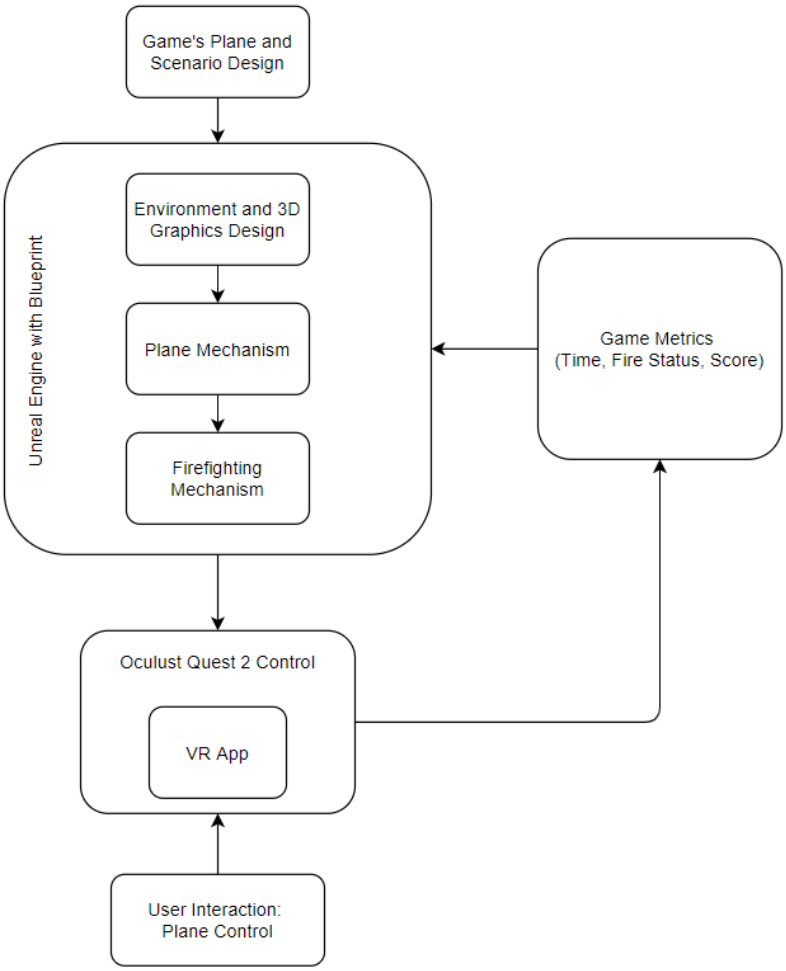


Figure 8: Block Diagram

2.2. PROCESS VIEW

2.2.1. Control Plane

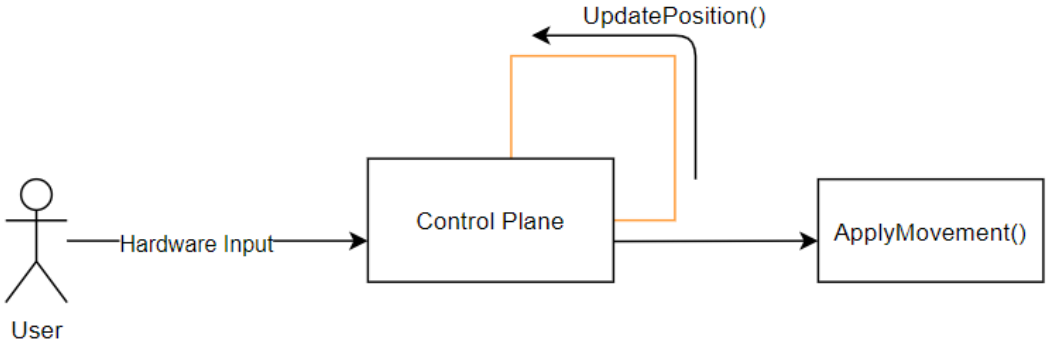


Figure 9: UML Communication Diagram of Control Plane

Use Case Realization : UpdatePosition() calculates current speed, and then calculates new position and the applied gravity based on the current speed and finally applies movement to the plane based on the calculations.

2.2.2. Crash Plane

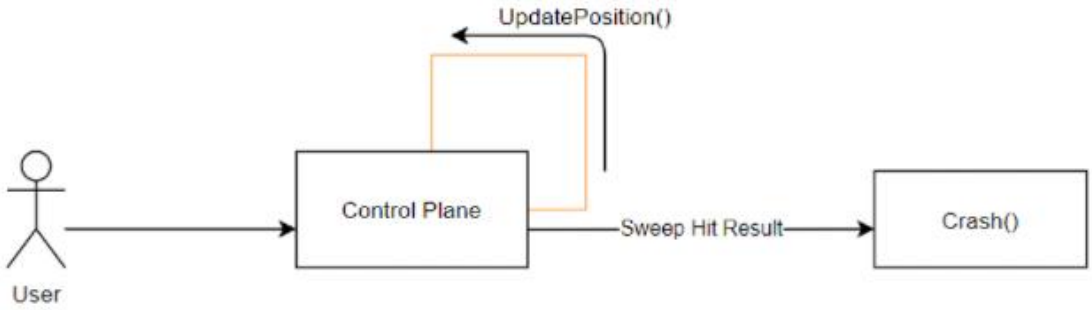


Figure 10: UML Communication Diagram of Crash Plane

Use Case Realization : UpdatePosition() checks for physical collisions every frame and detects a crash if any collision happens. This triggers the Crash() function.

2.2.3. Release Water

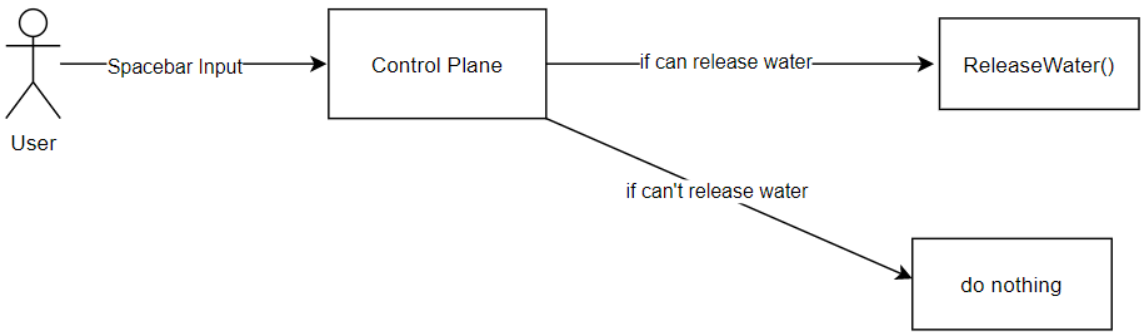


Figure 11: UML Communication Diagram of Release Water

Use Case Realization : When the user presses the spacebar, Unreal Engine detects hardware input and checks if the plane is allowed to release water. Plane can only release water every 5 seconds. If it is allowed to release water, Input will call ReleaseWater() function

2.3. PHYSICAL VIEW

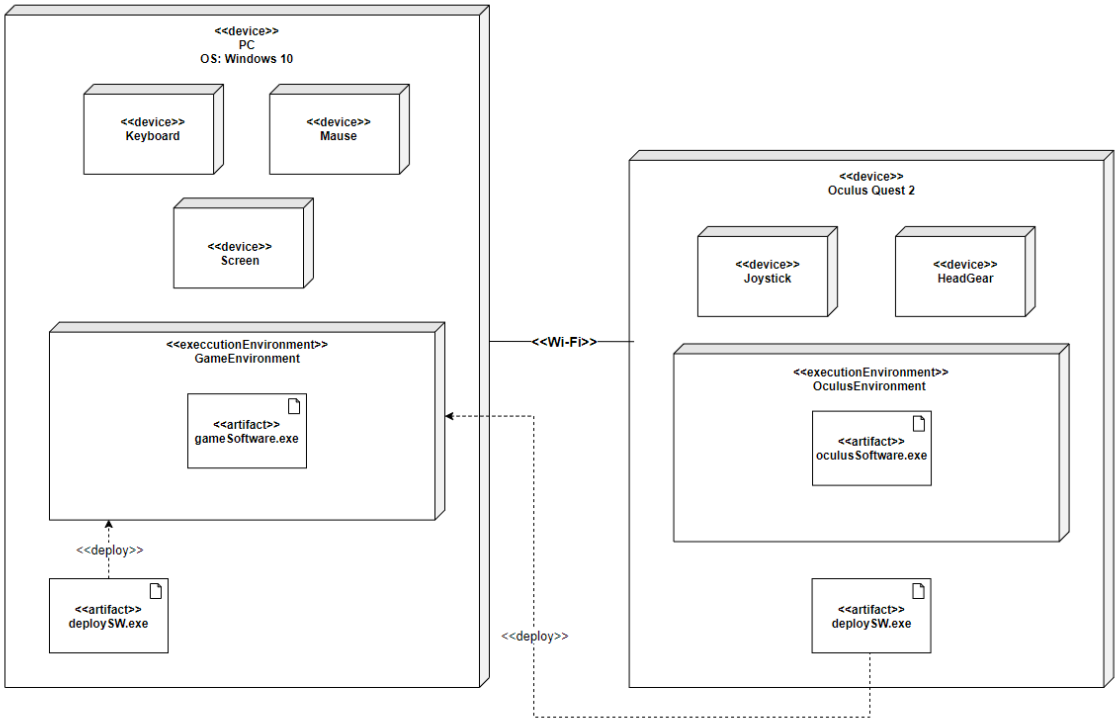


Figure 12: UML Deployment Diagram

2.4. DESIGN QUALITY

Our design will meet the performance requirement by ensuring well optimization. To achieve a good optimization in our project, we will make use of Unreal Engine's LOD (level of detail) system which decides how detailed the objects appear to the camera based on how far you are looking at the objects. Unreal's LOD system automatically replaces meshes with low quality versions as soon as the camera moves away from the camera.The LOD system enables us to gain a lot of FPS by managing the trees' level of detail based on camera distance.

We use similar optimization techniques in different scenarios too such as hiding grass when the player is too far. On top of this, we are ensuring that the geometry of the objects we are using in our scene are not extremely detailed and we prefer to have less polygons/triangles if possible to achieve better performance by sacrificing level of detail in the geometry.

When it comes to lights, we are using a stationary sunlight and we are baking the lightmass information onto the landscape instead of using a dynamic light to save computation power and therefore achieve better performance. This way, the sunlight only casts shadow onto movable objects in the scene which are only the airplane and the animated foliage.

To achieve reliability, we will minimize the amount of bugs in the game. This way, the game will be more reliable to run for long times without experiencing any crashes.

3. LOW-LEVEL DESIGN

3.1. Use Case: Control Plane

Since we have no takeoff functionality in 1st increment, our plane starts the game mid-air with some speed. Therefore, here we initialize speed variables.

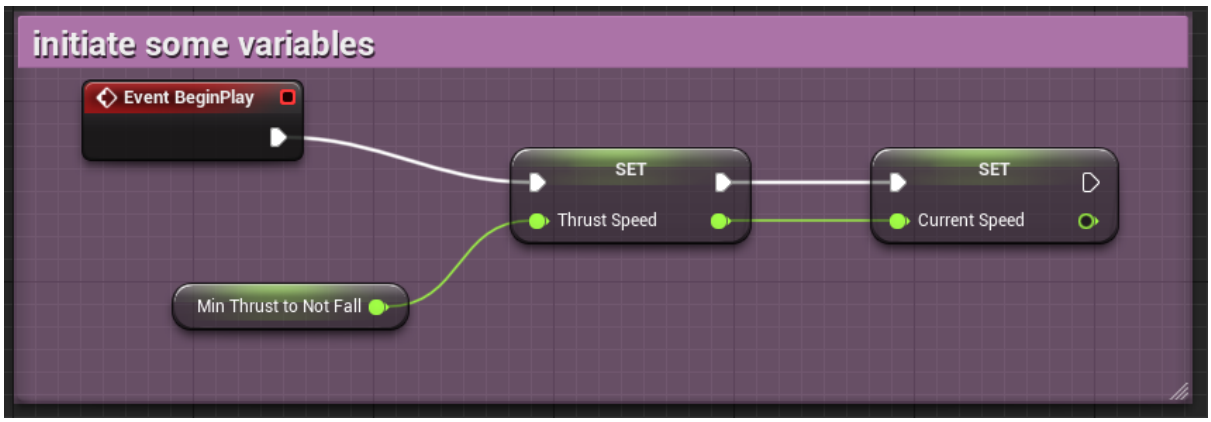


Figure 13: Use Case - Control Plane: Variable Initiation

Here we implement the ability to speed up or slow down based on the axis event we created which listens for keyboard inputs “W” or “S”. “Thrust Speed” is what “Current Speed” takes into consideration in the Tick function which we will explain later.

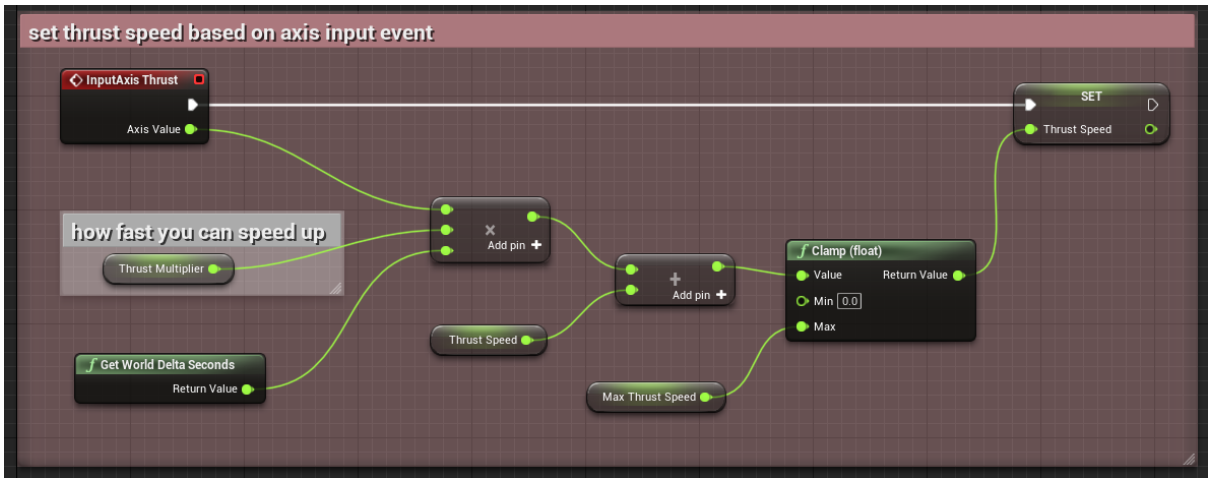


Figure 14: Use Case - Control Plane: Setting Trust Speed

Here we change the yaw and pitch of the airplane (looking up and down and looking left and right) based on the mouse events.

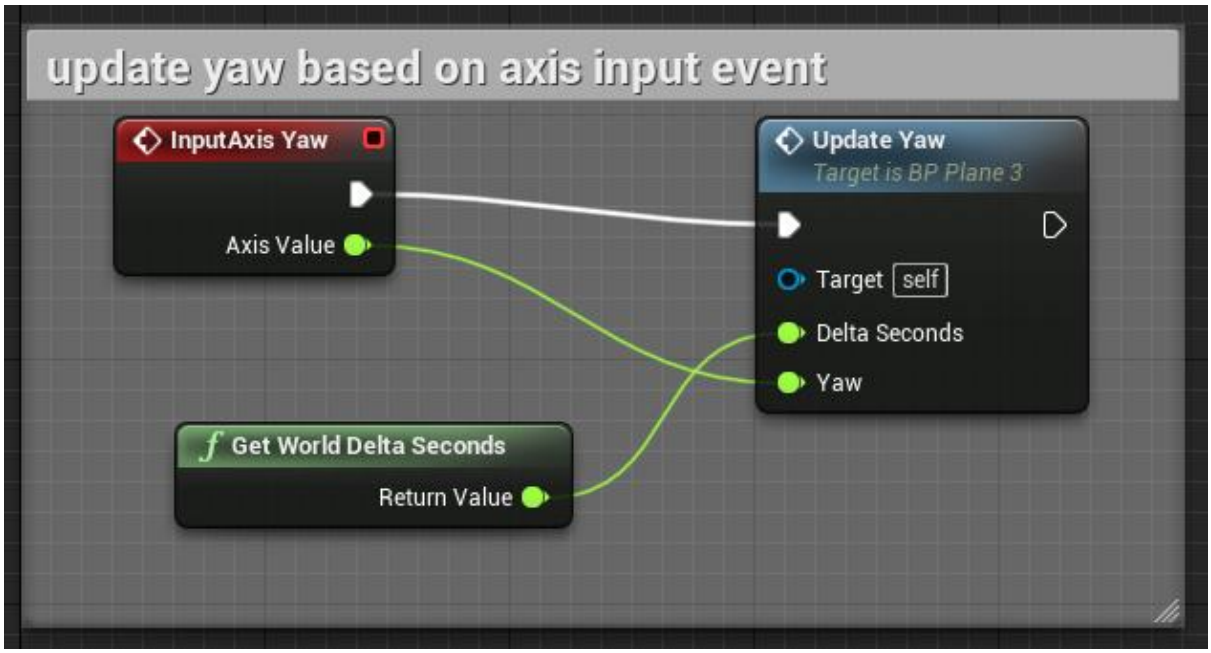


Figure 15: Use Case - Control Plane: Update Yaw

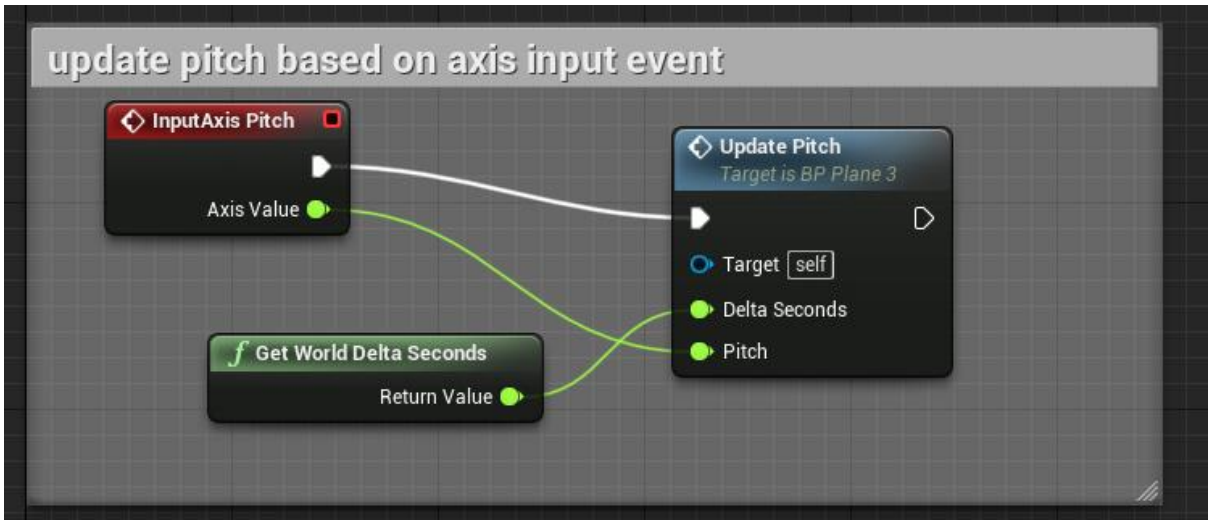


Figure 16: Use Case - Control Plane: Update Pitch

In the Tick event which gets called every frame, we call the function to update position based on Current Speed, we print variables for debugging purposes and finally correct the yaw of the plane for smoother control of the airplane.

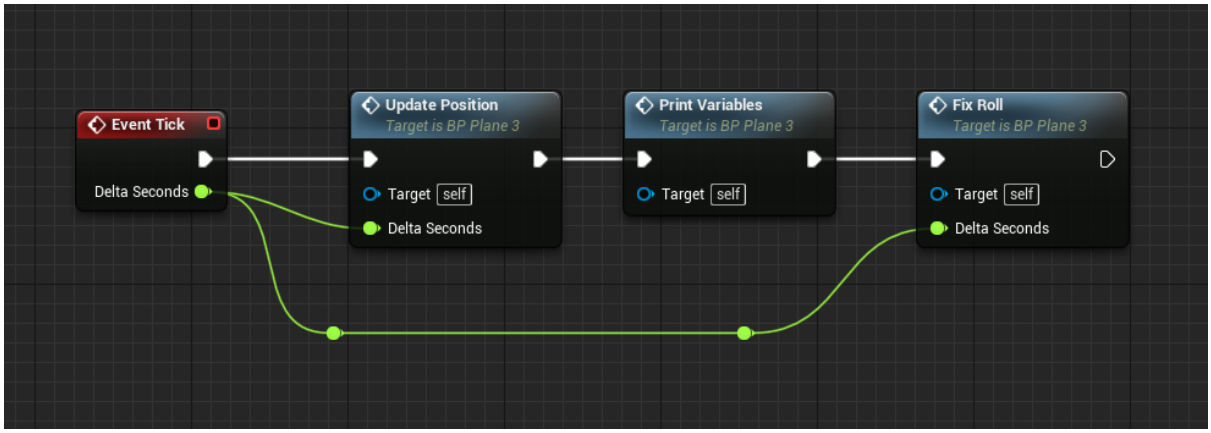


Figure 17: Use Case - Control Plane: Update Position

The “Update Position” function has four functionalities which are calculating the current speed, calculating the new position based on current speed, calculating the gravity to be applied based on the current speed and finally updating position based on calculated new position and the gravity to be applied.

In the current speed calculation area, we make use of float linear interpolation for a smooth transition from current speed to the thrust speed. While doing this, we take into consideration the world delta seconds (which is how many milliseconds have passed since the last frame) and the drag attribute which represents air resistance. The linear interpolation keeps going until the current speed is the same as thrust speed.

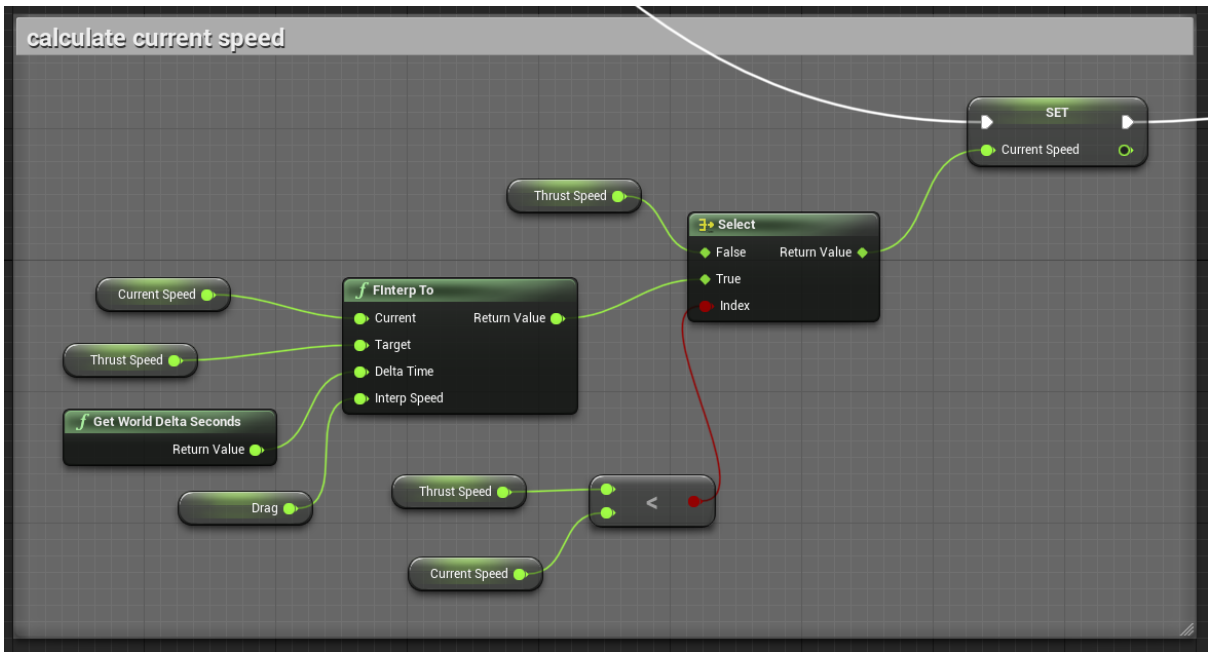


Figure 18: Use Case - Control Plane: Calculate Current Speed

To calculate the new position, we get the forward vector of the airplane and multiply it with the current speed. Since the forward vector is normalized by default (it's magnitude is 1), we will need to multiply it with speed to calculate the new world location. We also need to multiply this value with delta seconds to avoid any stuttering issues.

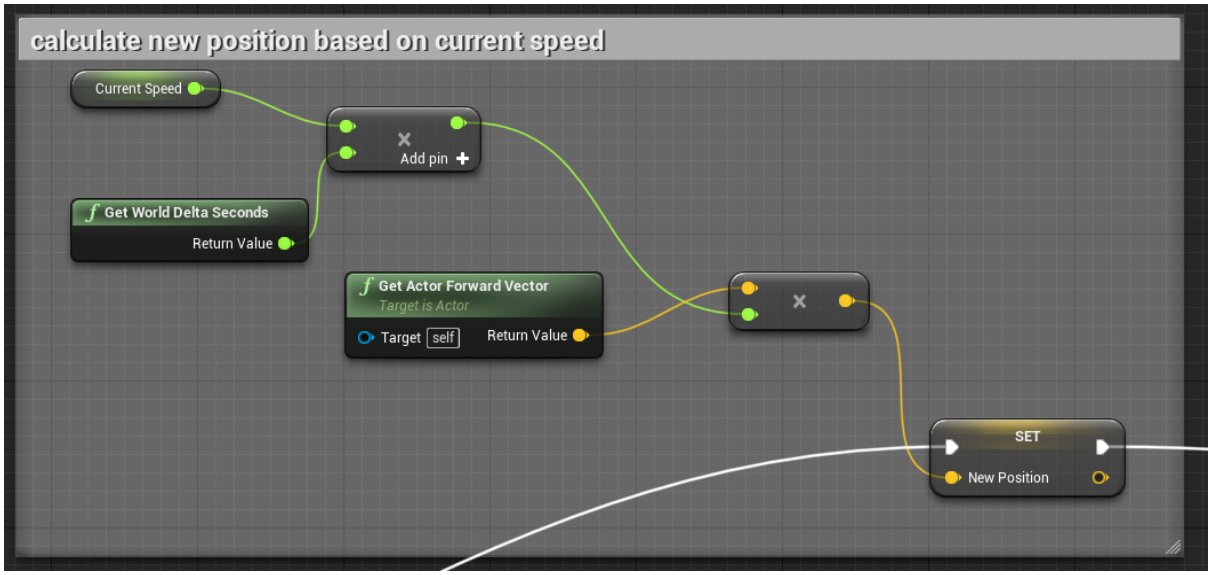


Figure 19: Use Case - Control Plane: Calculate New Position

If the airplane is below a certain speed, we start applying gravity to the plane to make it start falling. If the airplane is fast enough, there will be no gravity applied.

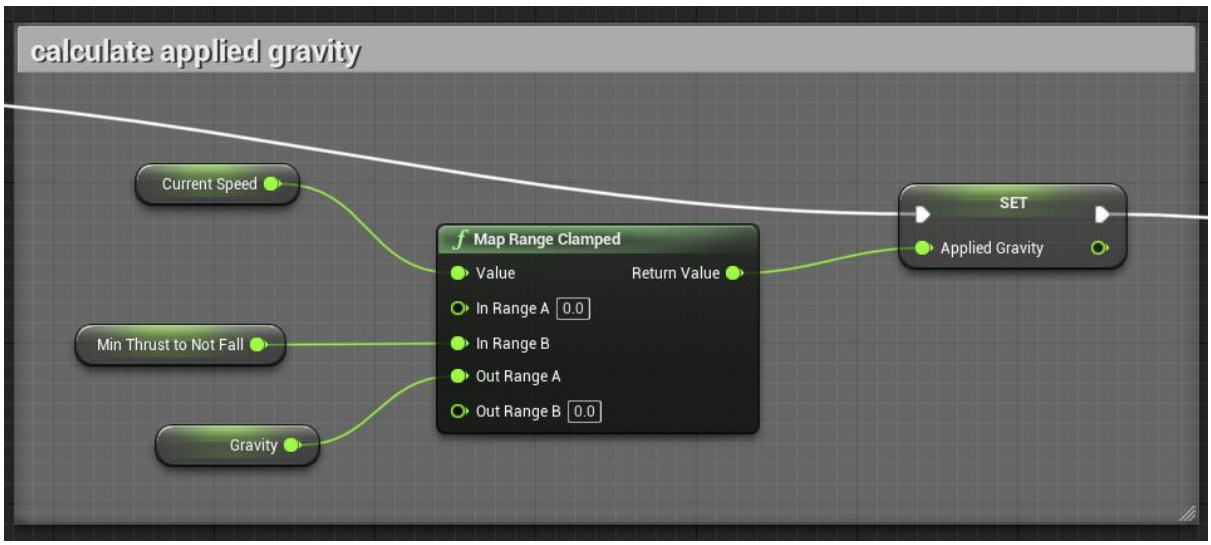


Figure 20: Use Case - Control Plane: Calculate Applied Gravity

Finally, after we calculate all these variables, we are ready to update the position. We break the new world position into x, y and z values and decrement the gravity to be applied from the z and then use this vector to add world offset to the airplane actor. Sweep parameter is set to true so that we can detect if any collision happens.

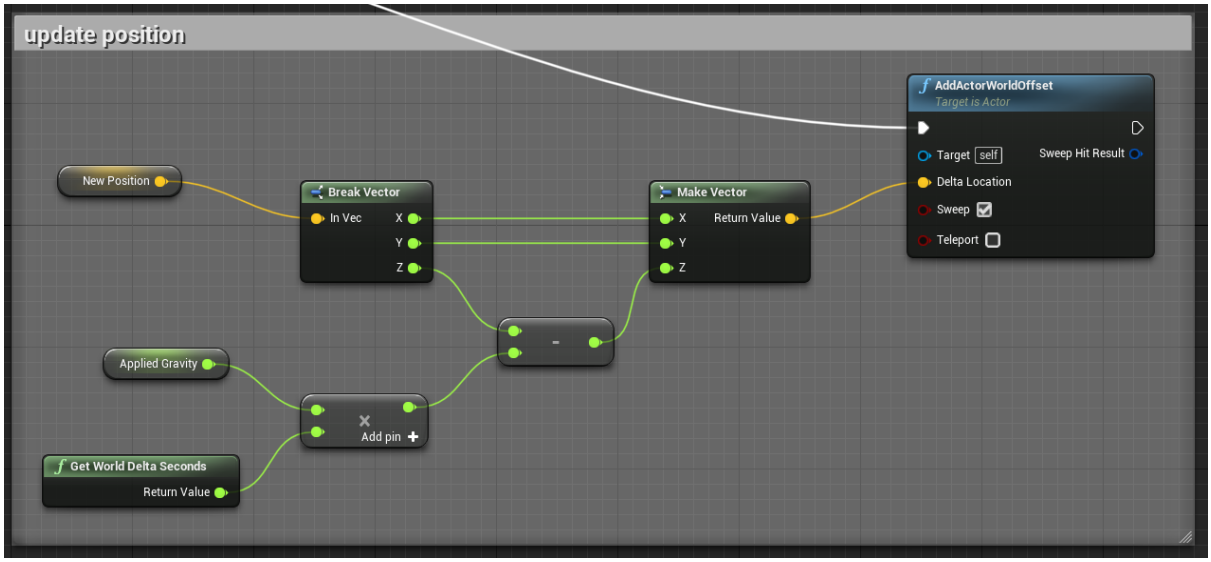


Figure 21: Use Case - Control Plane: Update Position

3.2. Use Case: Crash

We use that sweep hit result to detect any crashes and we spawn an explosion particle effect at that location.

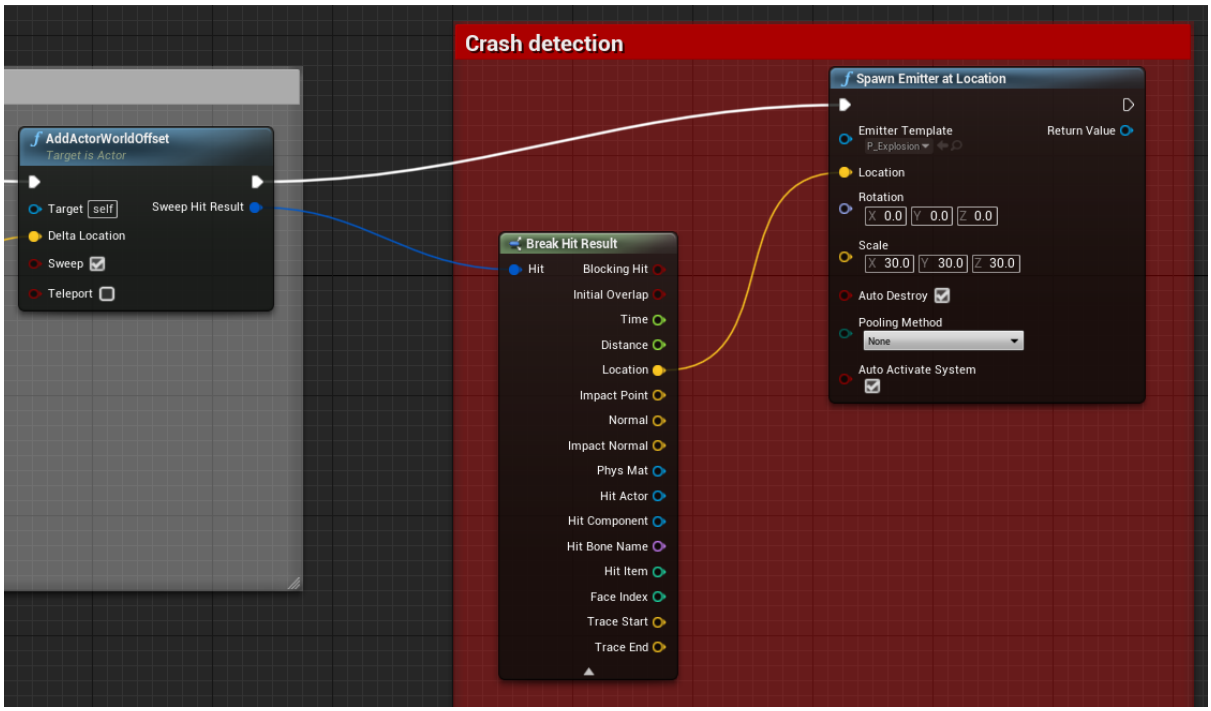


Figure 22: Use Case - Crash

3.3. Use Case: Release Water

Currently, instead of releasing water, we release multiple balls that represent water. When spawning the balls, we use plane's location minus 400 in z axis and we randomize the y axis location by +/- 200. We use the water's hit event to detect if it hits any trees that are on fire.

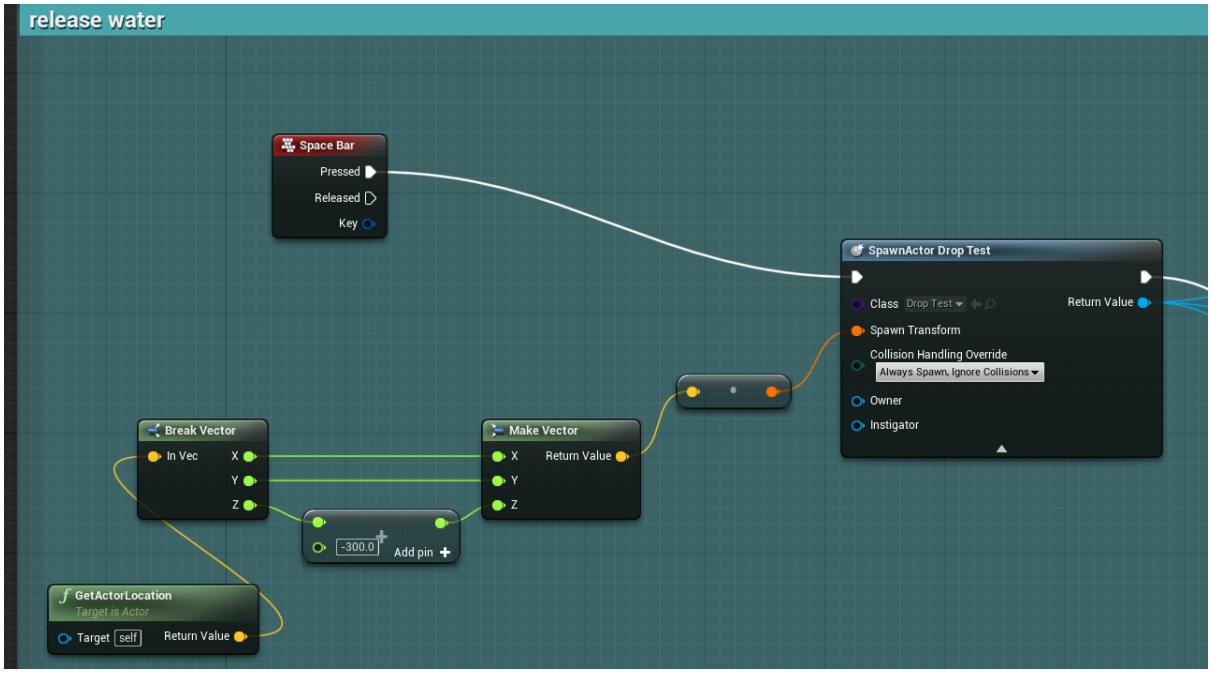


Figure 23: Use Case - Release Water 1

After water is spawned, we give it an impulse (velocity in a specified direction) matching the plane's current speed and facing direction (in short, velocity). Facing direction and speed of each ball (water) is slightly randomized. For these calculations, we use the airplane's rotation, forward vector and current speed.

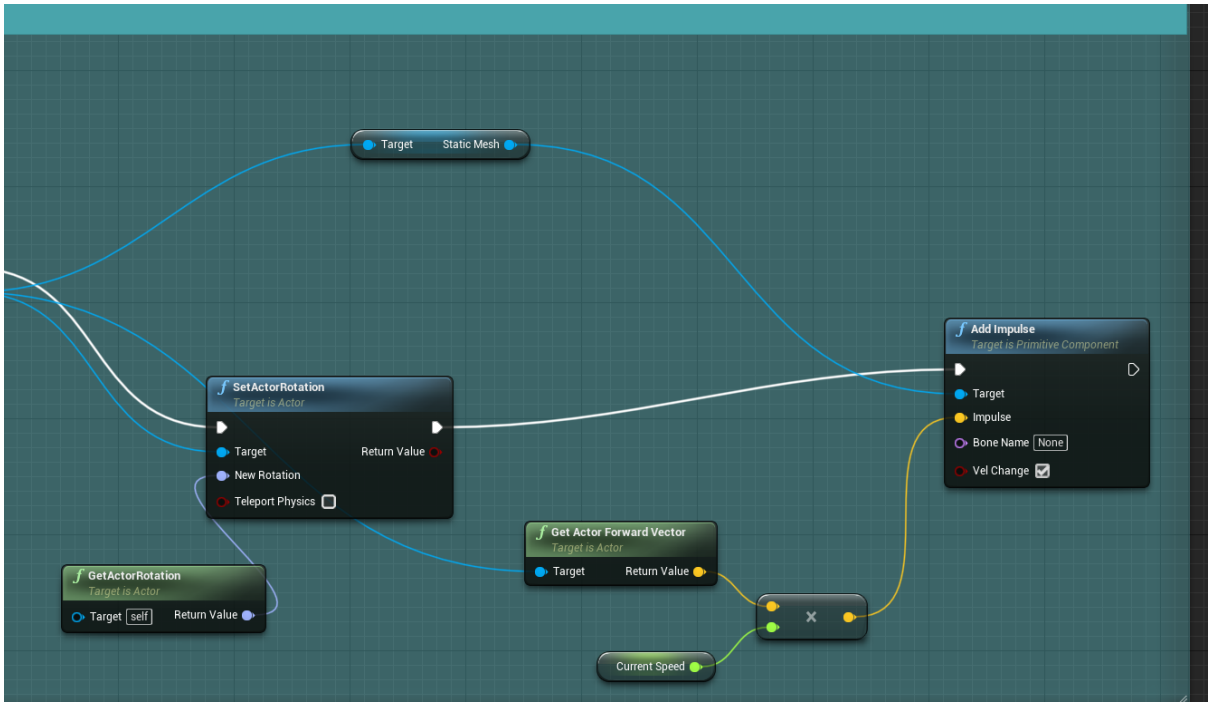


Figure 24: Use Case - Release Water 2

3.4. Use Case: Extinguish Fire

We created a foliage blueprint which we derived from when creating the trees in our environment. This way we have full control over the trees when it comes to gameplay mechanics.

In the Extinguish() function, we deactivate fire animation, set IsOnFire to false, set BeenExtinguished to true, global variable TreesOnFire is decremented by one and clear timer so the tree doesn't attempt to periodically spread it's fire anymore. Trees that have been previously extinguished can't be set on fire anymore.

Figure 25: Use Case - Extinguish Fire

3.5. Set On Fire Function

When a tree is set on fire, IsOnFire is set to true, fire animation is activated, global variable TreesOnFire is incremented by one and the tree is assigned a timer to periodically call the fire propagation event.

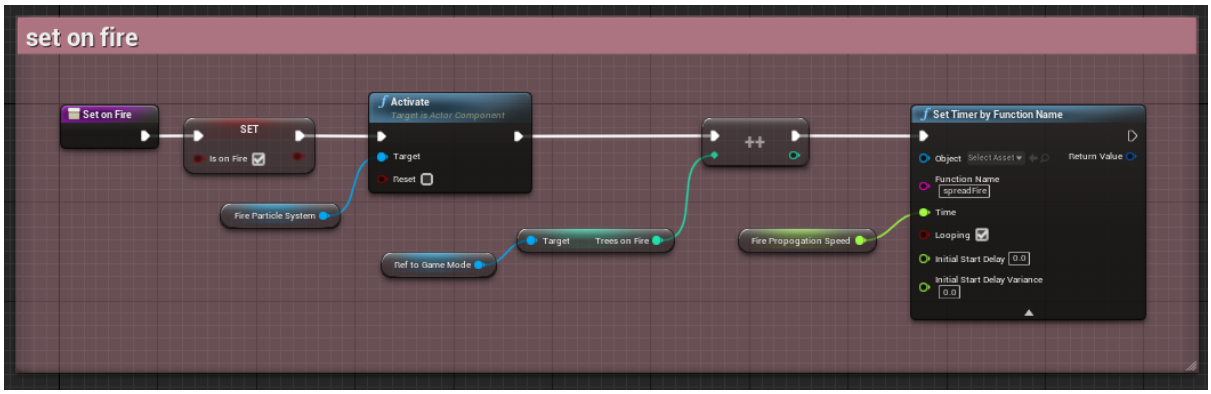


Figure 26: Use Case - Extinguish Fire