CMP302 Game Mechanics Report

Lukas Gouveia 2103219

Summary

The mechanic that I have created is a fighter jet which is easy for the user to control and fly around the map. Alongside the jet itself, the mechanic also comes with missiles that the player can shoot that track down targets and explode on impact.

The following is a detailed breakdown of the mechanics and systems within it:

* The Fighter Jet which can be controlled by the player.
	+ The missiles:
		- Explode on impact with anything they collide with.
		- Explode on their own after a set timer.
		- Has two distinct firing modes:
			* Tracking:
				+ Using the targeting system, the missile will shoot towards a target.
			* Free fire:
				+ The missile will simply be shot in the direction the jet is facing.
	+ A targeting system which can:
		- Find all enemies within the player’s range.
		- Store all the enemies in an array from closest to furthest.
		- Add UI elements to enemies to show what is being locked on to.
	+ Toggle free look where the user can look around them without interfering with the plane’s movement.
	+ UI elements that show details about the jet such as altitude, speed, and thruster level
	+ Upon colliding with anything in the map:
		- An explosion is spawned at the jets crash location.
		- The jet respawns at the start position with their speed reset.

Link To my demonstration video: <https://youtu.be/vWpdJ122i6E>

Requirements and specifications

Introduction

The mechanic I have created takes inspiration from games like the Ace Combat Series and popular multiplayer game War thunder as it fits into the category of an arcade flight game. The purpose of this project is to build a base jet to build understanding on how the mechanics of an arcade flight game work. This leaves plenty of room for expansion with the addition of difference weapons, planes and much more once the programmer understand the basics.

The goal of this project was to make a fighter jet which the player controls and is capable of flying around a map and be able to interact with the environment around it. Adding a crash feature when the player collides with objects, missiles so the user can shoot at targets in the scene are example of player interactivity that will be demonstrated.

Mechanic Overview

Key Features:

* Allow the user to affect the Roll, Pitch, and Yaw of the jet.
* Allow the user to control the thrusters power level.
* Allow the user to activate a free look camera to monitor the map around them.
* Allow the user to shoot missiles from the jet.
* Allow the user to toggle on and off the homing on the missiles.
* Allow a future designer to make other weapon types while using the same targeting system.

Implementation of the design:

Programming:

This project is aimed towards programmers who are familiar with using Unreal Engine 5.

Sound Design & Modelling:

As this is a programming project the 3D models and sound effects that have been used are all sourced from the Unreal Engine marketplace as well as other sources on the internet. These sources are all referenced below.

Mechanic Features

Jet Movement

Description:

The ability to control the jets rotation on all axis to change the roll, pitch, and yaw of the aircraft. Alongside this the speed of the aircraft is also controlled by the user, if the jet goes below a certain speed gravity will begin to affect the aircraft pushing it towards the ground.

Input:

The gameplay of the mechanic will require input from the programmer’s choice of mouse and keyboard or a controller.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functionality:

*Receive Input:*

The jet must be able to receive input from the user in order to operate.

*Update Roll:*

The jet must rotate to demonstrate rolling. The jets ailerons must also rotate in response.

*Update Pitch:*

The jet must rotate to demonstrate a change in pitch. The jets flaps and elevators must also rotate in response.

*Update Yaw:*

The jet must rotate to demonstrate a change in pitch. The jets flaps and elevators must also rotate in response.

*Thruster Speed:*

The jet speed must stay consistent with the thrust speed which can be increased and decreased by the user.

Jet Sound effects

Description:

Sound effects involving the jet.

Input:

The gameplay of the mechanic will require input from the programmer’s choice of mouse and keyboard or a controller.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functional Requirements:

*Thruster Pitch:*

The pitch of the thruster sound effect must increase and decrease according to the jet’s thruster speed.

*Thruster Volume:*

The volume of the thruster sound effect must increase and decrease according to the jet’s thruster speed.

*Missile launch:*

A launch sound effect must play each time the user fires a missile.

Jet Respawn

Description:

If the jet is to ever crash into anything within the map an explosion is to be played at the location followed by the jets speed and position resetting to what they were at the beginning of run time.

Input:

This feature does not require any input to be received from the user.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functional Requirements:

*Start Transform:*

Set a start transform for the jet which is equal to the jets starting transform.

*Collision Detection:*

The jet must be able to detect when it has collided with any other objects in the map.

*Explosion effect:*

Visual and audio effects of an explosion must be played at the location where the jet crashed.

*Position Resetting:*

Set the jets transform to the previously established start transform.

*Speed Resetting:*

Set the jets speed back its default value.

Jet Thruster Effects

Description:

A Niagara particle system which is a simple glowing sphere inside the jets mesh’s thruster.

Input:

This feature does not require any input to be received from the user.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functional Requirements:

*Niagara Particle System:*

Create the Niagara particle system for the glowing sphere.

*Attach to mesh:*

Attach the Niagara particle system to the mesh at the location of the thruster.

Jet UI

Description:

UI elements presented to the user showing the jets altitude and speed as well as a crosshair at the centre of the screen for shooting the missiles.

Input:

This feature does not require any input to be received from the user.

Workspace:

The functionality of the jet will be developed with the unreal engine blueprint editor.

Functional Requirements:

*Display current Speed:*

Get the jets current speed and display it to the screen for the user.

*Display current Altitude:*

Convert the jets z axis value into meters and display it to the user.

*Display homing mode:*

Display whether or not the missiles are set to be homing or not.

*Display Thruster level:*

Use a progress bar to display the level of power currently being used by the jet’s thrusters.

Player Freelook

Description:

Upon activating free lock, the user can move the location of the camera around the jet to get a better look at the map around them.

Input:

The gameplay of the mechanic will require input from the programmer’s choice of mouse and keyboard or a controller.

Workspace:

The functionality of the jet will be developed with the unreal engine blueprint editor.

Functional Requirements:

*Input Toggle:*

Toggle the free look mode on when the player is holding down the button assigned to the system.

*Allow camera movement:*

Allow the camera to move independently without affecting the jets movements.

Missile Object

Description:

The missile actor that the jet will be able to shoot.

Input:

This feature does not require any input to be received from the user.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functional Requirements:

*Missile mesh:*

Add a mesh of a missile to the pawn.

*Projectile Movement:*

Include the projectile movement component and define the speed along with other variables.

*Homing Component:*

In the projectile movement component enable homing to make the missiles a homing projectile.

*Time out:*

Add a function so the missile explodes on its own after being in play for a certain amount of time.

*Explosion Effect:*

Add collisions to the missile so it explodes on collision with any object in the map.

Missile Launching

Description:

The ability for the jet to be able to shoot out missiles.

Input:

The gameplay of the mechanic will require input from the programmer’s choice of mouse and keyboard or a controller.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functional Requirements:

*Fire Input:*

There must be a designated input for firing the missile.

*Set homing mode:*

There must be an input designated to allowing the user to turn on and off the homing of the missiles.

*Set homing target:*

Once the missile has been launched the missiles homing target will be set to whatever target is currently being locked onto.

Targeting System

Description:

The targeting system will grab all enemies with the jets range and fit them all into an array from nearest to farthest away. The system will then set the closest target as the current target for the homing missiles and apply a UI widget to indicate which target is being targeted.

Input:

The gameplay of the mechanic will require input from the programmer’s choice of mouse and keyboard or a controller.

Workspace:

The functionality of the jet will be developed with the unreal engine blueprint editor.

Functional Requirements:

*Switch Target Input:*

There must be a targeting input to allow the user to switch targets.

*Get targets in range:*

Grabs all the target pawns within range of the jet and stores them in an array.

*Get Closest Target:*

Sort through the array of targets and find which target is the closest to the jet and sets that target as the current homing target.

*Target Actor:*

Add a UI targeting reticule on top of the target so the user can see which target is being targeted.

*Clear UI Reticle:*

Remove the lock on reticule from a target that is no longer being targeted.

*Clear Targets:*

After a few seconds delete the current target’s array to allow for more accurate targeting the next time the player triggers the input.

Target Jets

Description:

A target jet that is placed within the map for the user’s jet to lock onto for testing the missiles.

Input:

This feature does not require any input to be received from the user.

Workspace:

The functionality of the jet will be developed with C++ visual studio in combination with the unreal engine blueprint editor.

Functional Requirements:

*Jet Mesh:*

Add a jet mesh to the targets to fit the theme of the project.

*Lock on Reticle:*

Add a lock on reticle widget to the targets but make them hidden from display at the beginning of the game.

*Target Interface:*

Create an interface that sets the targets as being “Targetable” to allow for easier targeting for the targeting system.

Method

Jet Pawn

The jet pawn is the most essential element of the project as it is the pawn that the player controls. It is made up of several static meshes that all attach to a static mesh root component through sockets that have been created on the jet’s model. Alongside the static meshes there is a spring arm with a camera and audio component attached to it. The camera in this pawn is the main camera the user looks through while running the project and the spring arm is set up in such a way that as the plane speed up the camera is moved slightly away from the jet. The audio component is for the jet thrusters sound effect, attaching the component to the camera makes it so that there aren’t any audio inconsistencies when the camera is moved away from the mesh by the spring arm. Lastly there are also four scene components placed on the pawn as markers for where the Niagara effect for the thrusters should go and markers for where the missiles will be shot from. The jet pawn is set up in c++ and is structured as show:

* M\_fuselage (static mesh component)
	+ m\_rudderR (static mesh component)
	+ m\_rudderL (static mesh component)
	+ m\_elevatorL (static mesh component)
	+ m\_elevatorR (static mesh component)
	+ m\_aileronR (static mesh component)
	+ m\_aileronL (static mesh component)
	+ m\_glass (static mesh component)
	+ m\_flapR (static mesh component)
	+ m\_flapL (static mesh component)
	+ m\_missileBayL (scene component)
	+ m\_missileBayR (scene component)
	+ m\_LThruster (scene component)
	+ m\_RThruster (scene component)
	+ SpringArm (spring arm component)
		- JetLoop\_Cue (audio component)
		- Camera (camera component)

Jet Movement:

The forward movement of the jet is calculated by taking in the users desired input and increasing or decreasing the thrust speed of the jet. While the jets thrust speed is less than the current speed a new speed variable is calculated by interpolating between the thrust speed and the current speed to allow for a smoother deceleration of the jet. Otherwise, the jets current speed is set to the thrust speed.

The new position of the jet is calculated by taking the forward vector of the jet and applying the current speed. The amount of gravity that is being applied on the aircraft is also calculated by clamping a value between the defined gravity variable based on the current speed and where it fits between zero another predefined variable for the minimum speed the jet needs to not start falling.

An offset is applied to the jet pawn, the x and y positions of this offset coming from the new position previously calculated, and the z position is calculated using the applied gravity and the z value of the new position.

The roll of the aircraft is calculated by interpolating between the users input and the current roll being applied to the aircraft. The interpolation is done to avoid the aircraft rotating unnaturally and gives a smoother look to the movement. This calculated roll is then added to the local rotation of the actor on the correct axis. The ailerons are then also rotated on the aircraft around the y axis like they would on a real jet. This logic is how the pitch and yaw of the aircraft are also calculated and how all the other static mesh components are moved.

Jet Sound effects

The dynamic pitch and volume of the jets thruster sound effect is achieved by calculating a float which is the result of getting a clamped value between 0 and 1.5 where the inserted value into the clamp is the current speed and the range of the clamp is zero and the jets maximum speed. These calculations are all done using the FMath library in C++ code.

Similarly, to how the movement of the jet is calculated this clamped value is used in an interpolation calculation before applying the new pitch and volume to the audio.

Jet Respawn

The respawn is a system added to the jet to make sure the user has some kind of consequence for crashing the aircraft. Although not applying any penalty to the player this system can add a small layer of realism as it demonstrates how the user is not completely immortal while playing the game.

When the project begins the jet records a “startTransform” variable based on the initial transform of the jet. Upon colliding with anything within the scene the visual and sound effects of an explosion are played at the collision’s location. This is followed by resetting the jets current transform to equal the “startTransform” value. The speed of the jet is also reset back to what it was at the time the project started.

Jet UI

The jets UI is a user widget which is created by the jet and is attached to the camera’s viewport in its “BeginPlay” event. The UI shows the user their current speed and altitude as an integer and the current level of thrust being used in the form of a progress bar. The UI widget is structured as show:

* Canvas
	+ Crosshair
	+ Text (Speed)
	+ Overlay
		- UI Image
		- Speed Text variable
		- Thrust Progress bar.
	+ Text (Altitude)
	+ Overlay
		- UI Image
		- Altitude Text variable

In the jet blueprint there is a code to update the text variables by directly taking the jets current speed and z position. For the progress bar there are calculations to get the current thrust percentage by getting the current thrust power and comparing it to what the max thrust the jet can achieve.

Player Freelook

The free look camera is implemented as a way to allow the user to look around the map without being restricted to the locked camera the jet has by default. When the user pushes the input assigned to triggering free look the spring arm attached to the jet is able to be moved by the players using the x and y values of the users input. This means that freelook needs to be assigned to an input with multiple axis of input such as a mouse or controller analog stick. In my design I choose to use a controller.

Missile Launching

The missile is the only weapon that the jet is capable of firing with both normal and homing launch modes.

Once the player presses the input to fire the missile the missile actor spawns at the locations of the m\_missileBay scene components alternating between the left and right component each time a missile is fired. Sound effects for the missiles launching are also played at the same location. Immediately after the missiles are launched the jet pawn casts to the blueprint for the player controller which controls the targeting system to grab the currently targeted enemy and sets the enemy character as the missiles homing component.

Targeting System

Targeting Sphere

The targeting sphere is in charge of grabbing all the targets within the jets range and putting them into an array to be sorted later on in the system. The targeting sphere is an actor with the singular component of a large collision sphere.

The sphere starts off by casting to the player controller in order to access and modify the array of actors within the controller. The sphere functions by grabbing all pawns that are within it with the “targetable” interface and adding them into an array of actors named “Targets in range” if they are a unique instance. At the same time, if the array “Recently Targeted” in the player controller does not contain the instance of the target, that target is added into the array “Fresh targets in range”.

Once all targets in range have been scanned and added into the arrays. If the array “Fresh targets in range” is empty, the sphere clears the player controllers “Recently Targeted” array and returns the “Targets in range array”. Otherwise, the sphere returns the “Fresh targets in range” array.

Player Controller

After the sphere gives the player control the array of targets the controller sorts through the array and finds the target which is closest to the jet pawn.

To ensure there are no errors in calculation the actor variable “Current Target” is reset before being assigned the actor that was just deemed to be the closest.

After 3 seconds have passed, the “Recently Targeted” array is cleared as this information might be outdated if the user has not fired a missile in an amount of time. However afterwards the current target is added as the first variable in the cleared “Recently Targeted” array.

This system allows the user to be able to switch between targets from closest to farthest away while looping back on itself when needed by using the “Fresh targets in range” array.

Target Jets

The target jets are a simple pawn with the main functionality being the interfaces within them that are used in the targeting system. The two functions within the interface that the target has are the “IsTarget” function and the “Lock Reticle” event.

The “IsTarget” function is a boolean function that returns true when called to show that the targets are able to be targetable.

The “Lock Reticle” event is used to decide if the enemies lock on reticle widget will be displayed to the user or not. The event is called in the player controller and is set to false when the target is no longer being targeted and set to true when they are being targeted.

UML

Development Process

For my mechanic I choose to start out by creating the entire project in blueprints to get a clear understanding of how the mechanic will work and how all the classes link together. Once implemented I choose to convert the jet and missile into c++ as these two objects do not rely on back and forth communicating with other classes so can operate on their own without interruption.

Conclusion

In conclusion I feel as though my ability to code in blueprints and C++ in Unreal Engine has significantly improved over the course of this project and the outcome was a mechanic that I can be proud of. I feel as though my implementation of the jet went well as a majority of the class was successfully converted into c++ and is able to function as intended.

Even though the project went well I still have a few things that I did not manage to complete and feel I could have done better. If given more time to work on the project I would have liked to implement the targeting system into c++. I was not able to achieve this task because of the complexity of the system and how often the player controller communicates with targeting sphere. Initially I also had a trail on the niagara system for the thruster, this trail also increased and decreased according to the thruster speed of the jet. While converting it into c++ I encounter an error that I could not fix where the system would no longer animate as intended.

References

*Jet Model, Movement, and thruster visual effects*: Channel, R. (2020). Unreal Engine 4 - Jet Plane Tutorial. Available at: https://www.youtube.com/watch?v=SkJNG52H\_IE&t=1073s.

*Jet UI implementation*: Poseidon, U. (2022). UE5: Fighter Jet tutorial| Part 4- HUD. Available at: https://www.youtube.com/watch?v=DpG9T\_Vnz-E.

*Jet Sound effects*: Poseidon, U. (2022). UE5: Fighter Jet tutorial| Part 7- Sound Effects. Available at: https://www.youtube.com/watch?v=59V7T7WnFZk&t=439s.

*Target Reticle*: AstrumStudio (2021). Lock On Target Indicator Reticle | Unreal Engine Action RPG Tutorial. Available at: https://www.youtube.com/watch?v=bXf62Z5o7aw&t=1s.

*Targeting system*: Laley, R. (2022). Unreal Engine 5 Tutorial - Tab Targeting Part 1: Targeting System. [online] Available at: <https://www.youtube.com/watch?v=6SUK1vxYTU4>.

*Jet Thruster Sound effect*: Pixabay (2022). Jet loop 01. [Sound effect] Available at: https://pixabay.com/sound-effects/jet-loop-01-32474/.

*UI Elements*: SunGraphica (2022). Sci Fi Game UI collection FREE version. Available at: https://sungraphica.itch.io/sci-fi-game-ui-collection-free-version.

*Missile model, explosion sound and visual effect*: Adia Entertainment (2014). Military Weapons Dark. [3D Model and Texture] Available at: <https://www.unrealengine.com/marketplace/en-US/product/military-weapons-dark?sessionInvalidated=true>.

*Target Model*: Vigilante (2022). Su30 Fighter (East). [3D Model and Texture] Available at: https://www.unrealengine.com/marketplace/en-US/product/su30-fighter-east.

‌