TINY TRANSIT

TDD

CREATED BY TEAM WHIPLASH

REVISION HISTORY

Version	Changes	Date
v0.1	- Created the initial design document. - Technical Goals filled out. - Game Objects and Logic filled out.	21/07/25
v0.15	- Edits to the Game Overview section.	22/07/25
v0.2	- IDE filled out.	23/07/25
v0.3	- Package mechanics completed. - Deliveries roughly planned.	24/07/25
v0.4	- Deliveries filled out.	25/07/25
v0.45	- Minor edits to document for better reading clarity. - Hazards roughly planned, need feedback.	26/07/25
v.5	- Ideas on vehicle mechanics.	28/07/25

v1.0	- Updating all sections to fit new game direction, specifically:
	  - Game Overview
	  - Technical Goals
	  - Game Objects and Logic
	  - Mechanics
	  - Packages
		14/09/25

v1.1	- Continuing updates on outdated sections, specifically:
	  - Mechanics
	  - Deliveries
	  - Hazards
		15/09/25

v1.2	- Updated TDD Visuals	19/09/25

v1.3	- Updating and filling out sections to fit new game direction:
	  - Mechanics
	  - Leaderboard & Saving
	  - Maps and Gamemodes
	  - Physics
	  - Audio
		24/11/25

v1.35	- Updated following sections:
	  - Audio
	  - Physics
		3/12/25

v1.36	- Updated following sections based on feedback:
	  - Graphics
	  - Physics
	  - Maps & Gamemodes
		8/12/25

v.1.4	- Updated following sections based on feedback: - Movement/Vehicle Physics	9/12/25
v1.5	- Added Minimum Requirements - Added rest of layout for user interface	14/12/25

v1.6	- Added User Interface Interactions:
	  - Main Menu
	  - Select Level,
	  - Pause menu
	  - Leaderboard
	  - Loading screen.
		15/12/25

1.0 | GAME OVERVIEW

Tiny Transit is a top-down, chaotic, delivery arcade game where the player must make the most amount of money in a certain amount of time. This can be achieved by delivering packages, getting bonuses from drifting, destruction, etc., and participating in mini-games to beat previous high scores. Challenge levels provide a break from the time crunch game mode with their unique goals and leaderboards score values.

1.1 | TECHNICAL GOALS

There are high, medium, and low priority goals to be met for the final product:

Name	Description
High Priority - important for the game to be complete.	
3D Graphics	- LODs, chunk loading, occlusion culling, etc. helps reduce the performance impact of a large map if necessary. Otherwise, a low poly count for each model will suffice.
60 FPS+	- With all optimizations in place and a settings menu for different graphic levels, 60 FPS+ should be achievable for hardware above the minimum requirements.
Car AI	- AI for traffic and cars. Moving along a spline, following traffic rules, braking when close to other cars, and object pooling.
Basic Particle System	- Using Unreal's built-in Niagara system.
Realistic Physics	- Physics for vehicle movement, collisions, hazards, terrain, etc.
Vehicle Controller	- Player controlled vehicle, custom made from the ground up using suspension casts from each of the vehicle's wheels and applying forces to the car's main collider.
Top-down Camera	- Stays at the same angle throughout the game and occludes any objects that would block the camera's view of the player.

Gamemodes	- Time Limit (3, 5, 10 minutes) with several modified variations that include more features/hazards (e.g. Fire Hydrants, Alien Beams). Hard mode is unlocked by achieving the Gold medal in the normal level. - Heat Mode, identical to Time Limit with the added bonus that picking up and dropping off packages will add more time based on the delivery's distance.

User Interface	- Main Menu UI
	  - Profile Select Dropdown
	  - Automatically appears on first game launch when no profile has been created. Prompts users to make a profile.
	  - Allows the player to swap between profiles for score saving purposes.
	  - Start
	  - Transitions to gamemode select screen.
	  - Gamemode Select
	  - Shows more information and options for the gamemode the user selects, such as time duration.
	  - Settings
	  - Adjust player controls, audio, and graphics.
	  - Leaderboard
	  - Stores score information for each gamemode and their modifier variants. Challenge mode may have different leaderboard scores according to the scoring system tied to them.
	  - Position
	  - Icon
	  - Name
	  - Score
	  - Gamemode Specific Details
	  - Quit Button.
	
	- Main level game HUD - Shows information on the timer, money, deliveries, weight indicator, minimap, off-screen indicators, and bonus pop-ups.
	- Deliveries list - Shows a menu to view information

	on current deliveries (value, weight, destination, and name). - End of Run, Top 10 Leaderboard - Acts the same as the general leaderboard but only shows the top 10 scorers and an option to save the run score to the selected profile.
Medium Priority - builds upon existing high priorities.	
Improved third-person camera	- Add zoom and player settings (FOV especially).
Complex AI optimizations	- Object pooling - once a car is out of a certain range from the player, add them back to the pool and spawn a new car nearby (preferably out of line of sight).
Particle System optimizations	- Use smaller materials, less particles, and swap between GPU and CPU depending on which needs optimization.
Low Priority - small changes, not necessary.	
User Interface optimizations	- Event driven UI updates - Only update the UI when necessary through interfaces and binding delegates.

1.2 | GAME OBJECTS AND LOGIC

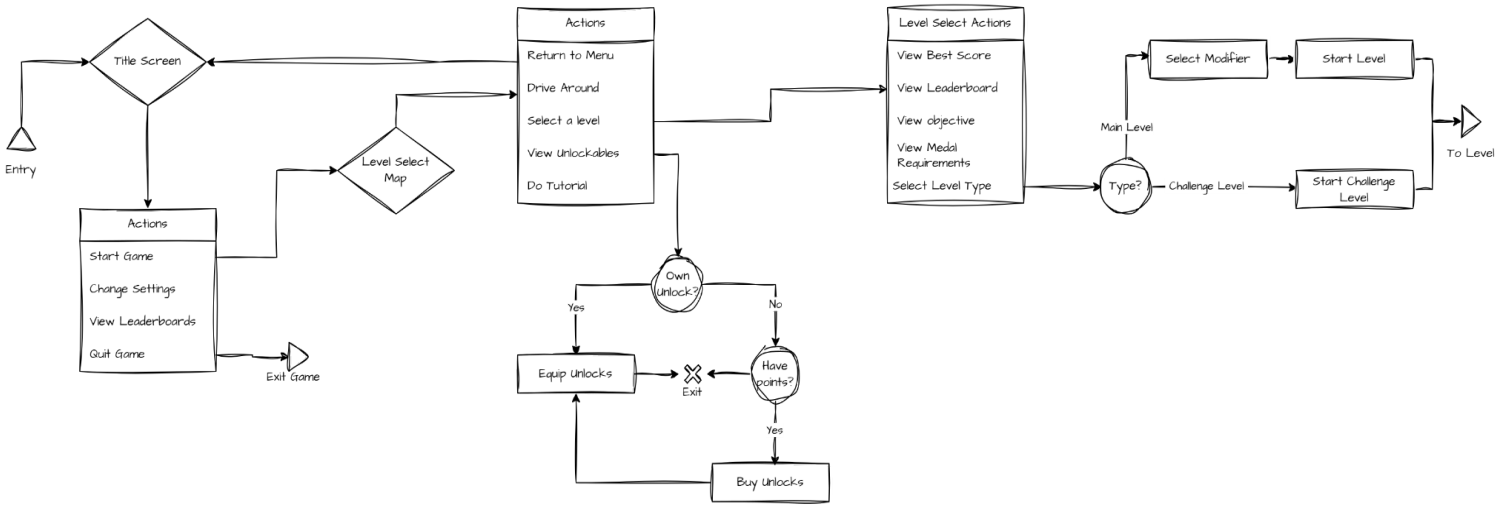
Name	Description/Logic
Player	
Vehicle	- Forward, left, right, and reverse. Drifting and bouncy physics. Extremely fast speeds.
Inventory	- Picking up packages, delivering them, and being able to visually see them appear on the car. Packages can have different modifiers and affect vehicle movement.
HUD	- View packages, their distance, value, and which modifiers are active.
Hazards	
Terrain	- Terrain affects the player's movement by slowing them, making turning harder, increasing friction, etc.
Buildings	- A hazard in the sense that the player comes to a stop when running into them. Not much logic involved.
Modifier Specific Hazards	- Base class for modifier specific hazards to allow designers to place them in the base level but not have them activated unless they meet the Hard Mode requirement. - Fire Hydrants (slippery puddle when knocked over), Alien Tractor Beam (sucks you upwards), etc.
Other	

Packages	- Packages are what must be delivered to set destinations on the map. Packages lose value when dropped and while being delivered.
	- They can have modifiers such as:
	  - Tiny Car - temporarily shrinks the car, making it harder to control.
	  - Double Destruction - increases money earned from destroying the environment.
	  - Turbo - increases the maximum acceleration and deceleration values, making the car

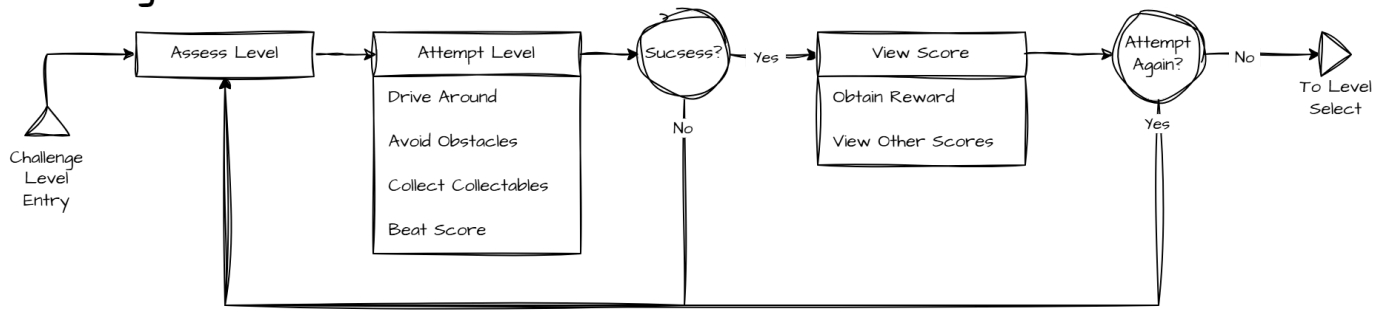
	faster.
Cars	- Cars disrupt the player's movement and can be collided with to send them flying. Requires AI to make them move around the map and follow road rules.

1.3 | GAME FLOW

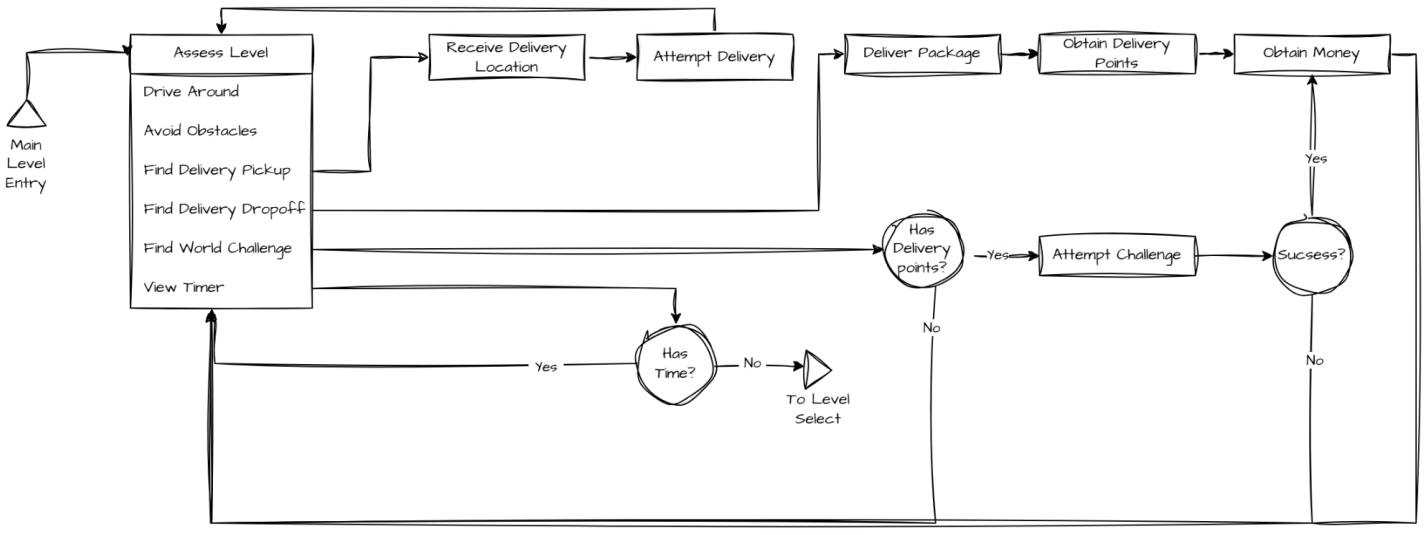
Menus and Level Select



Challenge Levels



Main Level



2.0 | DEVELOPMENT ENVIRONMENT

2.1 | GAME ENGINE

Tin Transit is developed in Unreal Engine v5.4.4, utilising C++ and Blueprints. Later versions of Unreal Engine (v5.5+) have had bugs and development-halting issues causing the team to choose v5.4.4.

2.2 | IDE

The IDE (Integrated Development Environment) being used differs from person to person and depends on their preferences. Below is a table of the programmers, their preferred IDE, and a description:

Name	IDE	Description
Alis	Jetbrains Rider	• Smart integration with Unreal Engine, has auto-complete for UPROPERTY() specifiers, and more. • Requires subscription for commercial use (earning money from games). • Will buy a subscription if the game is published for profits.

Ali	Visual Studio 2022	• I can use other IDEs but I have been using this IDE ever since I started this course.
		• Has a plugin called Unreal Engine Integration tool which when installed to the project will help with IntelliSense.
		  ○ It also enables Unreal engine 5 blueprint support for the project.
		  ○ It has a test adapter section to allow developers to discover, run and manage UE5 debug tests.
		  ○ It has a built-in naming convention checker to keep the developers' code good. It just needs an editorconfig.txt with all conventions in the text file.
		  ○ The plugin also has HSSL Configuration support.

2.2.1 | CODING GUIDELINES

Below are coding conventions that are applied for the duration of the whole project. This results in clear and organized code that both programmers and designers will be able to understand and edit if necessary. The most important guidelines have been extracted from Unreal Engine's official C++ coding standard and recommended asset naming conventions.

Programmers separate conditionals inside if statements by putting them on separate lines if they improve readability.

Conditionals and If statements:

```
if (IsMonday &&
    IsTuesday) {}
```

'IsMonday && IsTuesday' are used as an example even though they do not exceed half a page width.

Only include what is necessary in header files. Forward declarations reduce compilation time as the project gets bigger.

Forward declaration:

```
class TestClassOne

TestClassOne NewVar
```

To handle errors, programmers print UE_LOG errors to clearly diagnose the issue and prevent unwanted crashes.

Error handling:

```
if (errorOccured) {
    UE_LOG(LogTemp, Warning, TEXT
    ("errorOccured returned true while
    checking for errors in [class name]");
}
```

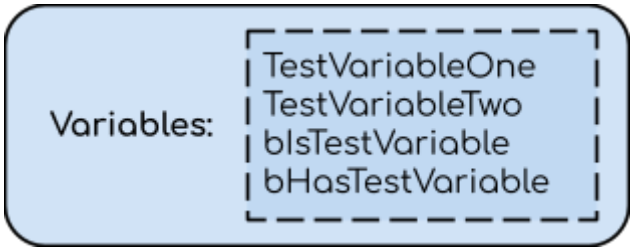
Access specifiers in header files go in order of public, protected and then private. Variables are listed at the top and then functions below them. Grouped functions are separated by an empty line, sorted by similar usages (getters, setters, logic, etc.). Structs and enums are declared above the main class. Below is a basic outline of a class layout in Unreal Engine C++:

```
UENUM()
enum class TestEnum : uint8 {};

USTRUCT()
struct TestStruct { GENERATED_BODY() };

class TestClass : public UObject
{
	GENERATED_BODY()

public:
	// Variables:
	bool bTestBool;

	// Functions:
	TestClass()

	// Returns the bTestPrivateBool, can not be changed.
	bool GetTestPrivateBool() const { return bTestPrivateBool;
}

	// Sets the bTestPrivateBool.
	void SetTestPrivateBool(bool bInBool)
	{ bTestPrivateBool = bInBool; }

protected:

private:
	// Variables:
	bool bTestPrivateBool;
};
```

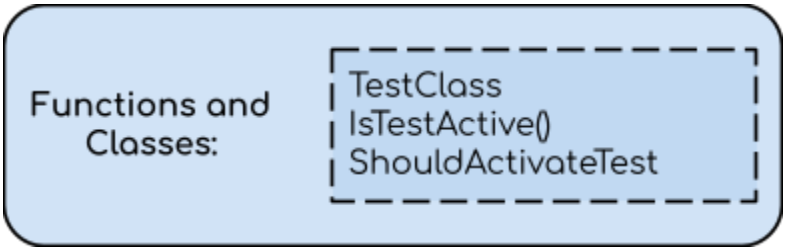
'Variable:' and 'Function:' comment is not necessary.

2.2.1.1 | NAMING CONVENTIONS

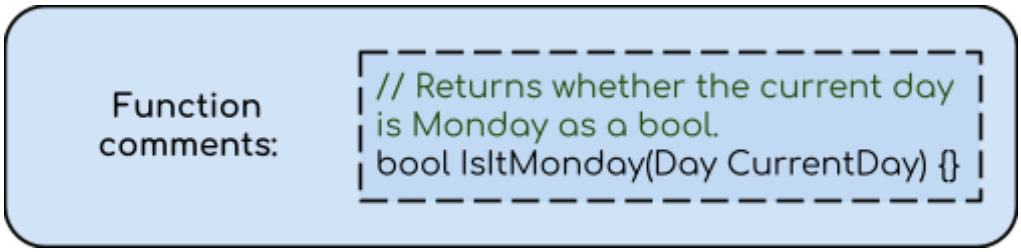
PascalCase is used for variables. Booleans begin with b as their prefix followed by a verb, for example ‘is’ or ‘has’.

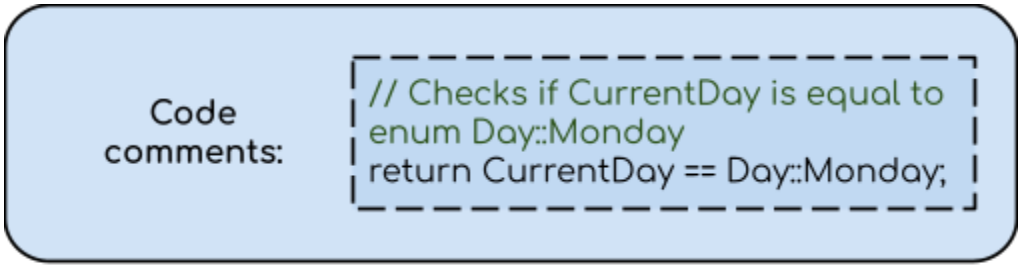


Functions and classes use PascalCase. All functions that return a boolean should ask a true/false question.



A function’s main purpose is described in a comment above its declaration in the header file. Inside the function’s definition, if the code is difficult to understand at a glance, some sections can be commented. However, code should be self-documenting and not include ‘magic numbers’ that other people can not recognize.

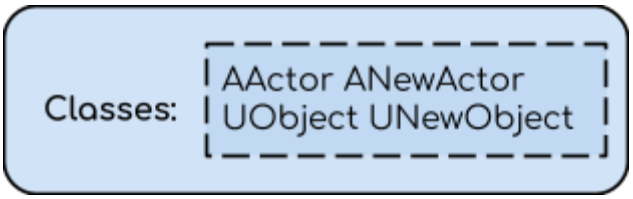




Code above is clear - this is just an example.

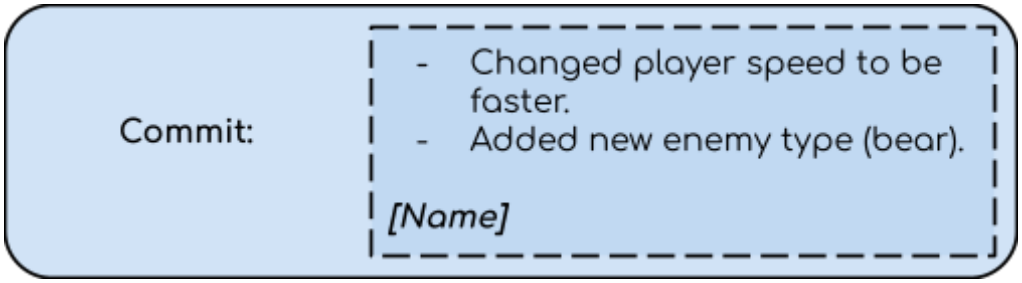
Classes that inherit from the following classes are prefixed by:

- UObject - U
- AActor - A
- SWidget - S
- Abstract Interfaces - I
- Enums - E
- Booleans - b



2.2.2 | SOURCE CONTROL PROCEDURES

Tiny Transit uses Perforce Helix P4V as its source control. Before making changes to any existing files, the user must check out the file. Once complete, the file(s) can be checked back in with a commit detailing a summary of all changes made. The name of the person submitting a commit must be mentioned. Commits should be made frequently, such as at the end of the day, to allow other users to make changes, prevent great losses of data, and reduce merge conflicts.



2.3 | THIRD PARTY LIBRARIES

There are currently no third party libraries used or planned to be used in the future.

2.4 | SOFTWARE

Some software used by designers, artists, and programmers may be shared or used by others in the same field.

Name	Usage
Designers	
Unreal Engine v5.4.4	Game engine to develop Tiny Transit.
Perforce Helix P4V	Source control for project.
Discord	Communication between team members.
Microsoft Teams	Communication between team members.
Google Suite	Paperwork, organisation, and project management.
Figma	UI/UX design.
Paint.net	Develop diagrams.
Draw.io	Flowcharts.
Photoshop	Gifs, images, and diagrams.
Blender	Modelling, UVs.
Substance Painter	Texturing models.
Substance Designer	Creating materials.
FL Studio	Sound design.
Canva	Creating and showing presentations.
Artists	
Autodesk Maya	Modelling, rigging, animations, and UV mapping.
Substance Painter	Texturing models.
Substance Designer	Creating materials.
Procreate	Drawings, mostly UI and design.
Programmers	

JetBrains Rider	IDE for C++ development.
Draw.io	UML diagrams and flowcharts.
Visual Studio 2022	IDE for C++ development.
Obsidian	A Zettelkasten software that holds all the users information on different topics, code, graphs ect; in one place. It links all the information into a graph for the user to see to make developing easier.
Audacity	Sound design.

2.5 | SYSTEM REQUIREMENTS

The minimum requirements for Tiny Transit were tested by playing the game on low spec PCs and laptops.

	Minimum Requirements
Operating System	Windows 10 64 bit
Processor	Intel Core i5-8700 or AMD Ryzen 5 3600
Memory	8GB Ram
Graphics	Nvidia GTX 1060 or AMD Radeon RX 480
Storage	2GB Available Space

3.0 | MECHANICS

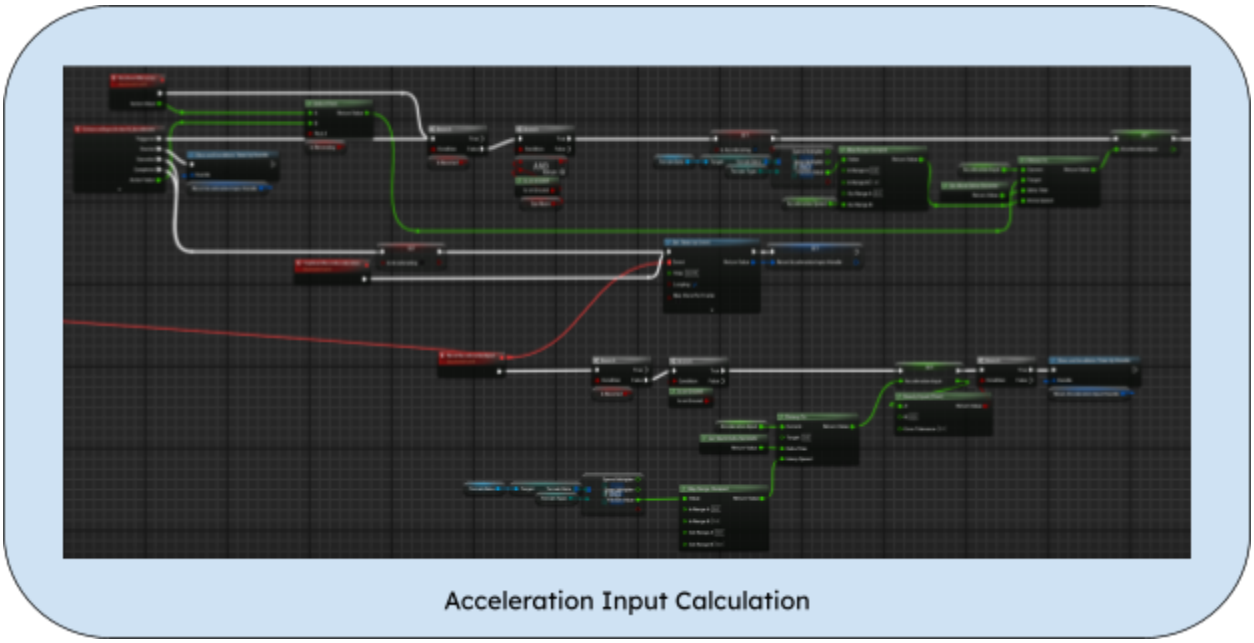
There are 5 main mechanics in Tiny Transit: player movement, packages, delivery system, cars, and hazards. These help make up the core gameplay loop of receiving packages, driving to the drop off location, avoiding hazards, and completing the delivery to earn money.

3.1 | MOVEMENT / VEHICLE PHYSICS

The player controls the vehicle using **[W]** to **accelerate forward**, **[A]** and **[D]** to **steer**, **[S]** to **reverse**, and **[Left-Shift]** to **drift** on Keyboard. All vehicle physics are implemented using custom Blueprints rather than Unreal Engine's built-in vehicle system, with Chaos used only for rigidbody simulation and collision responses.

3.1.1 | ACCELERATION

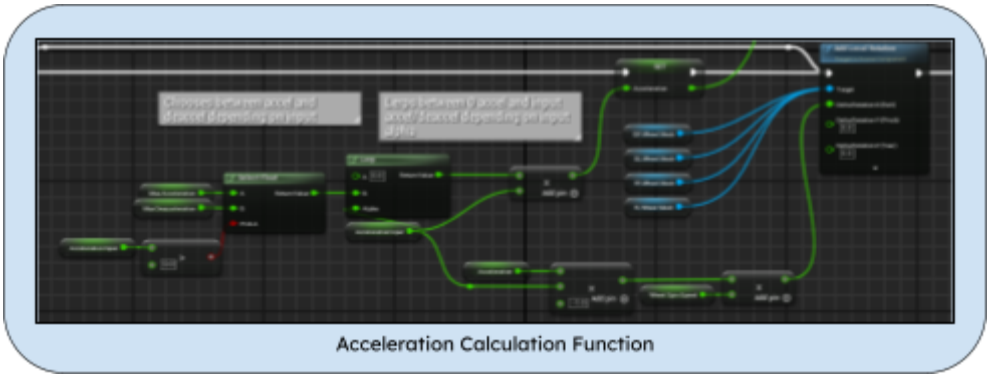
First, acceleration is calculated based on the player's input and whether they are currently on the ground and can move. The terrain affects the interpolation (lerp) speed of the **Acceleration Input** (float, between -1 and 1) to simulate lower friction being harder to accelerate/deccelerate on. The **Acceleration Input** is calculated as:



Acceleration Input Calculation

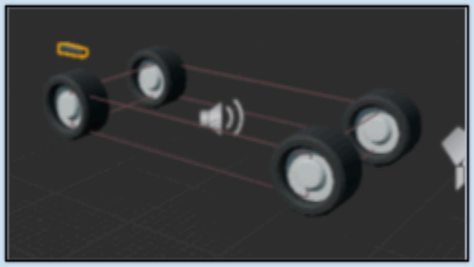
Once the player is no longer accelerating, the '**Reset Acceleration**' function timer is called to interpolate between the current **Acceleration Input** and 0, completely bringing the vehicle to a stop if it is on the ground and not boosted.

Inside the **Acceleration Calculation** function, the vehicle determines whether it should accelerate or decelerate based on the sign of the **Acceleration Input**. If the input is positive, the vehicle is accelerating and vice versa. The magnitude of the acceleration is interpolated (lerped) between 0 and the **Maximum Acceleration** (float, 4000) or **Maximum Deceleration** (float, 2500) based on the **Acceleration Input's** value (alpha). Finally, the interpolated value is multiplied by the original **Acceleration Input**, producing a signed result that correctly applies positive force for forward acceleration and negative force for braking/reverse. This system ensures a smooth scaling of acceleration and deceleration proportional to player input, as seen below:

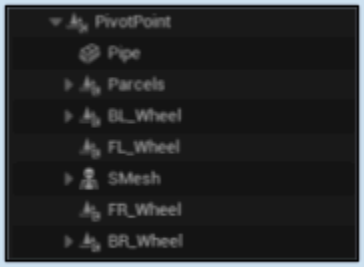


Acceleration Calculation Function

The vehicle chassis is a rectangular collision box with four scene components at each corner, representing the wheels. Each frame, line traces are cast down from these components to detect the ground. When a trace detects a hit, a suspension force is calculated and applied at that location to simulate wheel suspension. This gives the chassis a bouncy feel when moving vertically and ensures the vehicle responds dynamically to terrain.



Vehicle Chassis and Wheels



Wheel Scene Components

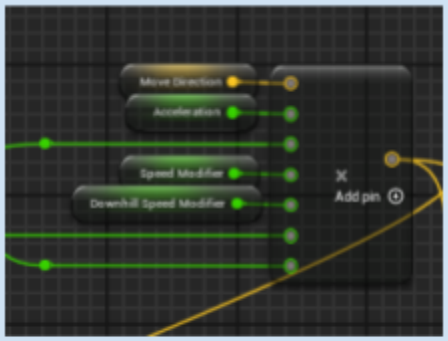
Wheel Cast Function

The **Accelerate Car** function is called inside the **Wheel Cast** function after suspension forces have been applied for that wheel. If the vehicle is allowed to move, the **Move Direction** (vector) is set to the **vehicle's forward vector**, and a force is applied at the wheel's location.

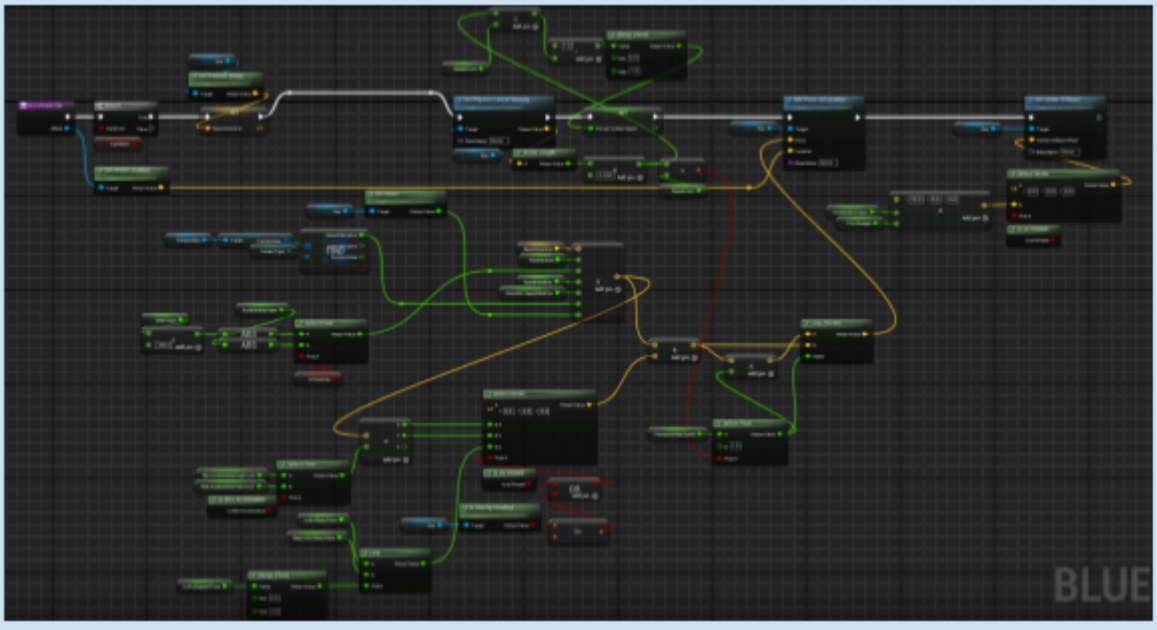
The force is calculated by multiplying together:

- The vehicle's forward vector
- The computed **Acceleration** (float) value
- Either the **Acceleration Input** or **Steering Input** (**Steering Input** is used if the vehicle is currently steering)
- The **Speed Modifier** (float, between 1 and 1.2, applied when a drift boost is active)
- The **Downhill Speed Modifier** (float, applied when the vehicle is angled downward, simulating gravity increasing speed on slopes)
- The **Terrain Speed Multiplier** (float, used to simulate different surface speed types such as slow marsh or fast roads)
- The **Vehicle Mass** (float, 50kg)

All of these factors combine to produce the final movement force that is applied to the vehicle at each wheel, giving the vehicle responsive acceleration while also accounting for boosts, terrain types, and slope direction.

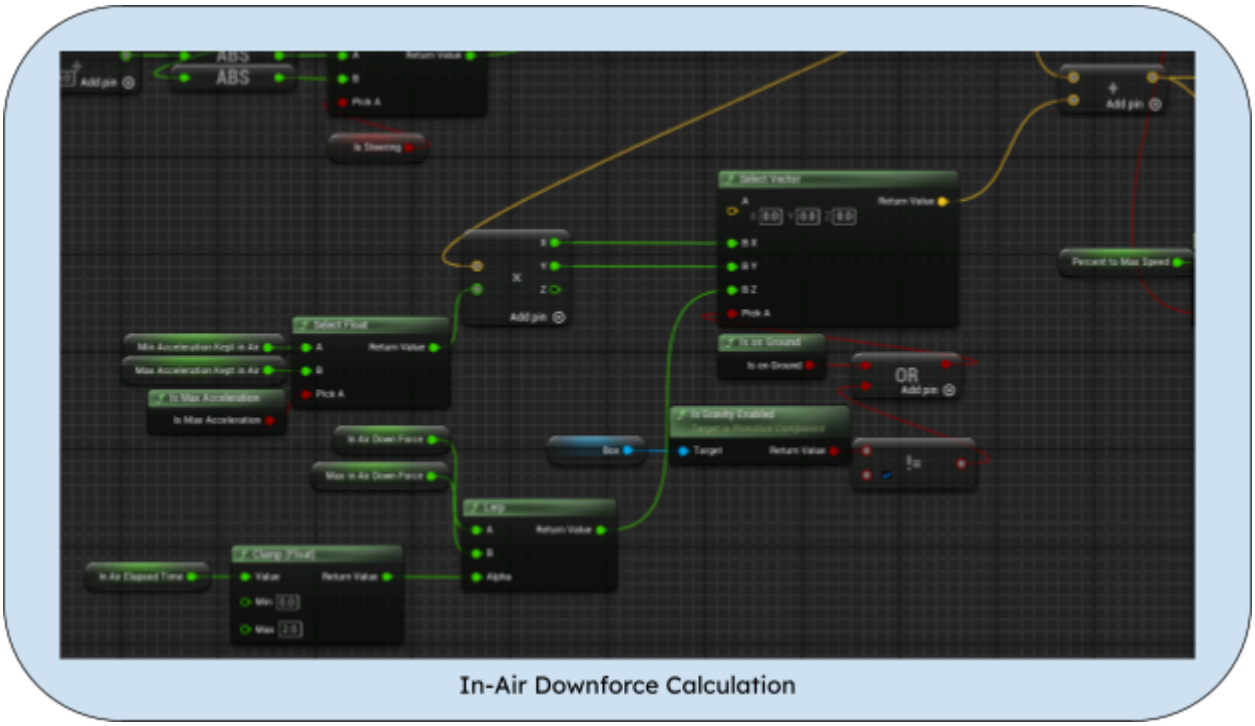


Unmarked green connections from top to bottom are: **Acceleration Input/Steer Input**, **Terrain Speed Multiplier**, and **Car Mass**.



Acceleration Car Function

Changing the project's gravity value caused other objects to feel heavy, and adjusting the vehicle's angular and linear damping would make it more unstable. Therefore, whilst the vehicle is airborne, an additional **In-Air Down Force** (vector, between -150000.0 and -300000.0) is added to the result of the acceleration calculation to push the vehicle downwards due to gravity not applying enough force and causing the vehicle to 'float'. The **In-Air Down Force** lerps between the minimum and maximum value depending on the vehicle's air-time, reaching the maximum after 2 seconds.



In-Air Downforce Calculation

The **Calculate Acceleration**, **Wheel Cast**, and **Accelerate Car** functions are all called in Event Tick during Pre-Physics. The Stabilization functions are also called after the acceleration has been applied, described in [6.0 | Physics]. Whilst calling the vehicle calculations during Physics tick was attempted, it was causing issues with jitteriness and lag during movement, and was therefore reverted.

Event Tick Physics Calculations

3.1.2 | STEERING

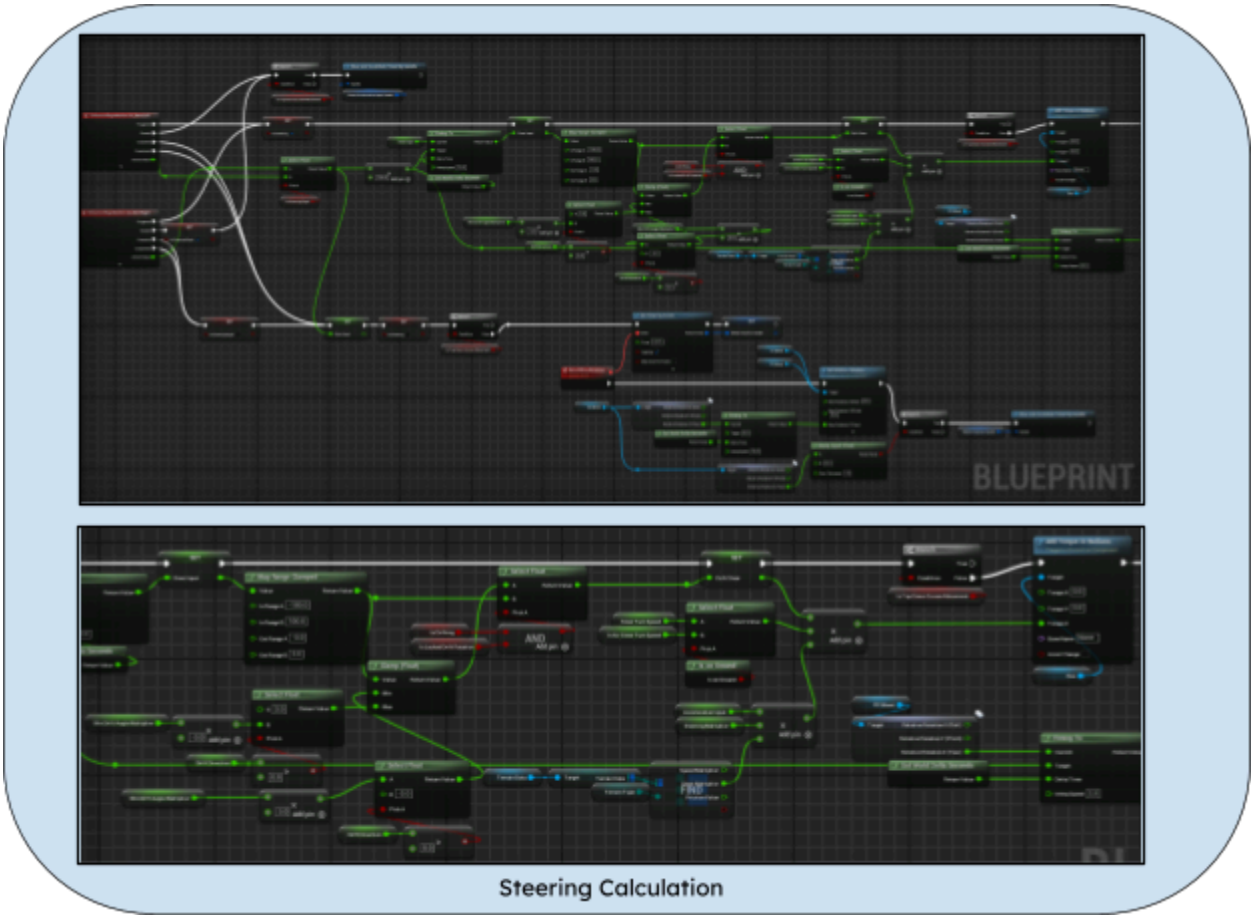
There are two separate input actions for steering, one for left and one for right, because the Input Action Mapping Context requires individual bindings to display correctly in the settings remapping menu. Both actions follow the same execution, where the **Steer Input** (float) is derived directly from the action value.

The **Steer Input** is used to calculate the **Drift Steer** (float), which is first mapped to a range between -3 and 3. Afterwards, it is clamped between a **Minimum Drift Angle** (float) between -0.3 to -3, and a maximum angle of 0.3 to 3. If the vehicle is currently drifting, this clamped value is used. Otherwise, the **Drift Steer** value simply becomes the **Steer Input** clamped between -3 and 3.

Once the **Drift Steer** value is determined, it is multiplied by several factors:

- The on-ground **Steer Turn Speed** (float, 5000000.0) or the **In-Air Steer Turn Speed** (float, 2000000.0), depending on whether the vehicle is airborne
- The **Acceleration Input** (float, between -1 and 1)
- The **Steering Multiplier** (float, between 1 and 1.2), increases to 1.2 while drifting, but otherwise remains at 1.
- The **Terrain Steering Multiplier** (float, greater than 0) (e.g. marsh is slower to steer on)

The final combined value is then applied to the vehicle as torque in radians along the Z axis, producing the desired steering behaviour.



Steering Calculation

3.1.3 | DRIFTING

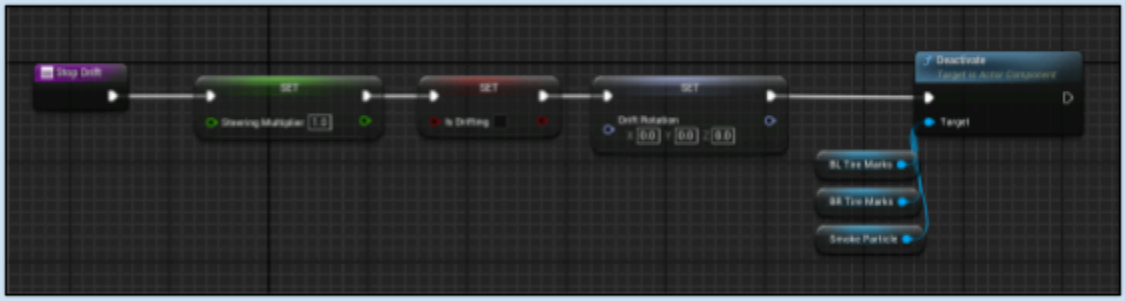
While the player is drifting, the **Start Drift** function is called, where the **Steering Multiplier** becomes 1.2x, the **Drift Rotation** (rotator, (0, 0, 0)) on the Z axis is chosen between 0.1 to -1.0 and -0.1 to 1.0 (depending on the **Drift Direction** (float, between 1 and -1) multiplied by the **Drift Angle** (float, 70.0).

Drifting Pivot Point

Once the player stops drifting, the **Steering Multiplier** is reset to 1.0 and **Drift Rotation** back to (0, 0, 0).

Inside the **Calculate Acceleration** function, the pivot point (parent of the vehicle's mesh, can be seen in Wheel Cast section above) is rotated along the Z axis to mimic the visuals of drifting without changing the rotation of the vehicle chassis, which would affect the handling.

Start Drift



Stop Drift

3.2 | PACKAGES

The player must collect packages and deliver them around town to earn as much money as possible. There is currently a limit of 6 packages. Packages have several properties:

1. **Value** - the maximum potential earnings from delivering the package.
2. **Weight** - affects the handling and speed of the player's vehicle (no longer used).
3. **Modifier** - changes the properties of the package, positively or negatively.
4. **Name** - the name of the package, randomly chosen based on its modifiers + from a list of random objects.
5. **Delivery Destination ID** - The ID associated with the delivery destination.

A **DeliveryManagerSubsystem** GameInstanceSubsystem class handles all packages, existing deliveries, dropoff and pickup locations, and the delivery flow. It is created when the executable launches but only activates once the player starts a level. It stores an object pool of Package Pickups in order to improve performance and prevent destroying/creating actors in large quantities.

The **Inventory** exists within the **DeliveryManager** as a **TArray<FPackageData>**, with helper functions for adding, removing, picking up, and dropping packages. Delegates are broadcasted when package states are changed in order to add/remove weight from the vehicle accordingly and remove package widgets from the Delivery List UI.

All logical data relating to packages is stored in a **PackageData** UStruct class, including all modifier types, package states, regions, and general package data (value, weight, ID, modifier, state, destination ID, name, and icon). The visual data is kept in a **PackageVisualData** UDataAsset class, specifically inside a **TMap<EModifierType, FModifierVisualData>**, with the **FModifierVisualData** UStruct holding TArrays for icons, meshes, and names. Each modifier has several icon, mesh, and name variations to make them distinct from each other.

A **DeliveryManagerData** UDataAsset class holds values for designers to adjust such as **TMap<EModifierType, FModifierValue>** ModifierValues which stores the weight and value multipliers for each modifier. Other settings include minimum/maximum package weight and value, dropped package value loss amount, game start time, maximum package amount, max pickup locations at one time, and the cooldown between activating a random pickup location.

A **PackagePickup** AActor class is the physical package that the player can pick up from the ground when it gets dropped. The delivery manager creates a pickup and stores the Package ID. When the player collides with the package it is added to the inventory using the Package ID and the pickup is returned back to the delivery manager object pool.

(NOTE: In the final version of Tiny Transit, package weight, dropping, and picking up packages has been removed due to being unnecessary).

See 3.3.1 | Package and Delivery UML Diagram for visualization.

3.3 | DELIVERIES

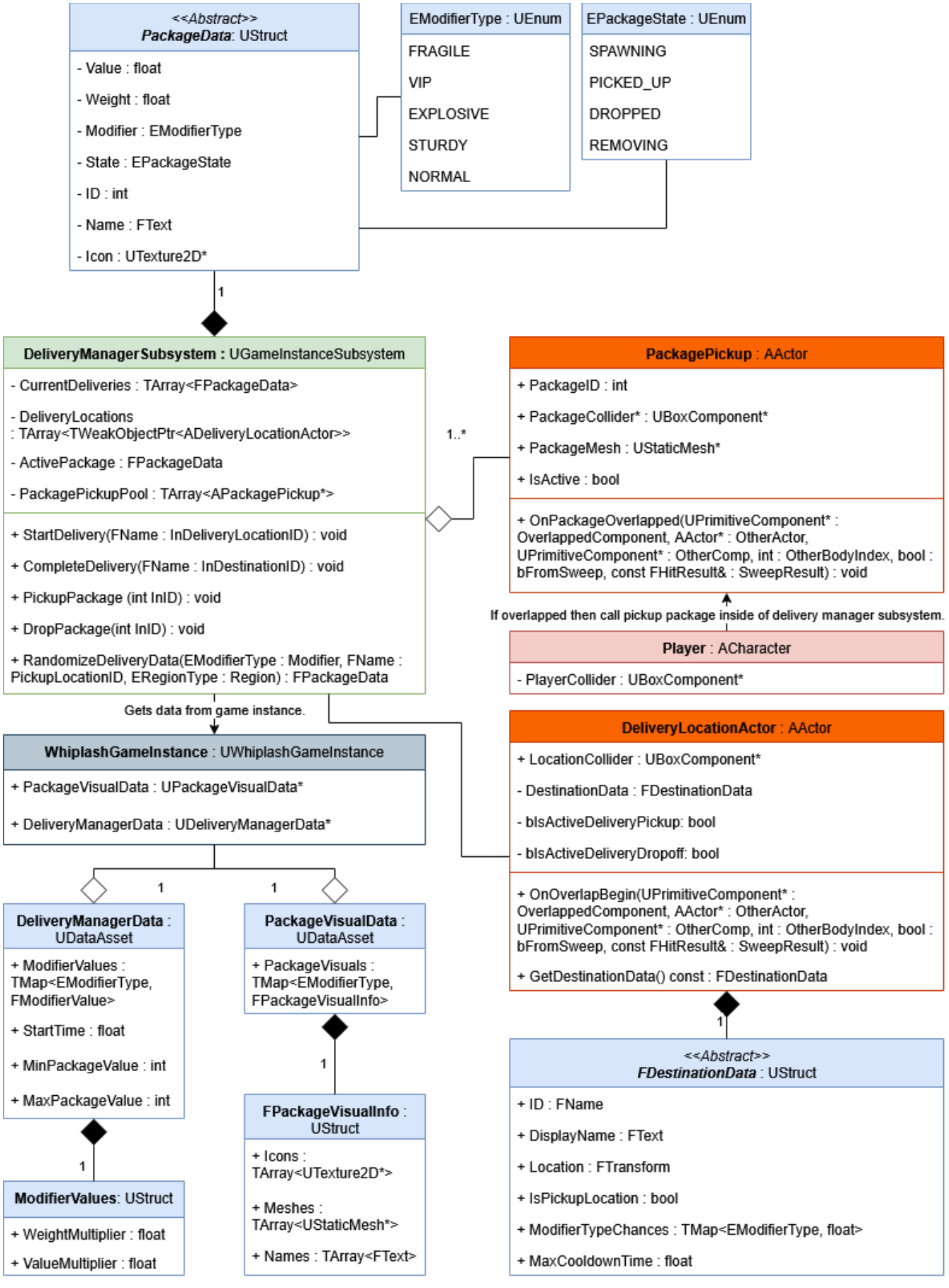
Delivery locations spawn at predetermined locations (chosen by designers) and are either set as pickup or dropoff destinations. Only a certain amount of pickup locations can be active at one time and each has a cooldown before it can be reactivated (set in **DeliveryManagerData** by designers).

The **DeliveryLocationActor** AActor class can be placed into the level by designers at any desired location. It holds a **FDestinationData** UStruct that stores its ID, display name, transform, an IsPickupLocation boolean, max cooldown time, region, modifier type chances, and current modifier. The player can run over an active pickup location to begin a new delivery. Once a delivery is started, the IsActiveDeliveryDropoff flag is set to true and visuals (indicators, particles, icons) are activated for the package's drop off location.

The delivery manager creates a randomized **FPackageData** and adds it to the **TArray<FPackageData> CurrentDeliveries** array (inventory). A dropoff location that matches the package's region (area that the package is picked up from, set by designers - can only be dropped off from a location in the same region) is chosen to be the package's destination.

Once the player overlaps with a delivery location, the delivery manager completes all deliveries associated, removes them from the inventory, disables the location's visuals, and increases the player's total money by their values (temporarily stored in the Delivery Manager and passed off to **UWhiplashSaveGame** for scoring).

3.3.1 | PACKAGE & DELIVERY UML DIAGRAM



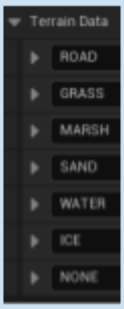
3.4 | HAZARDS

Throughout the game, players may encounter hazards that temporarily interrupt their progression. These can be in the form of unsafe terrain, buildings, and modifier specific hazards

3.4.1 | TERRAIN

Although physics materials do not affect the car due to the way it was programmed, they are still used to detect what kind of terrain is underneath the vehicle. A **DA_TerrainData** UDataAsset allows designers to edit the friction, and speed and steering multipliers for each terrain type within the Unreal Editor.

Speed and steering multipliers are applied directly to the vehicle's acceleration and rotation calculation respectively. Friction affects the interpolation speed of the acceleration input, higher values giving faster acceleration and vice versa.



All terrain types in the game.

3.4.2 | BUILDINGS

Buildings are a static mesh collider that prevent the player from moving past them. Very simple in nature, they don't provide any negatives other than forcing the player to change their route or waste their limited time when driven into.

3.4.3 | MODIFIER SPECIFIC HAZARDS

In conjunction with delivery levels, there are also altered versions that include specific hazards unique to that level. These modifier hazards derive from the **BaseModifierActor** AActor class and are activated/deactivated depending on their gameplay tag. For example, a delivery level at modifier level 1 will activate all modifier hazards that have the gameplay tag 'DeliveryLevel.ModifierLevel1'. Any other modifier actors will be ignored and have no impact on the level (they are disabled by default). Using this system helps designers create their map on a singular level without having to duplicate them.


Fire Hydrant hazard, appears in Delivery Level Modifier 1 and 2.

3.5 | LEADERBOARD & SAVING

The save system implemented in the **UWhiplashGameInstance** manages persistent player data for profiles, gamemodes, high scores, level modifiers, currency, and medals. The system loads or creates a **UWhiplashSaveGame** object and updates a wide array of player-specific and gamemode-specific data structures each time a game session is saved (usually when a run has ended).

The **SaveGame** function records run statistics, updates currency, manages level modifier unlock progression, records tracked stat values with support for higher is better or lower is better scoring, stores individual run histories per profile, and updates profile currency and shop unlock currency. It also maintains medal progression by calculating whether new scores or medal ranks surpass existing records set by the player. The **LoadGame** function retrieves or initializes the save object and doesn't change any save data.

Profile information (active profile and secondary profiles) is saved through the **SaveProfiles** function. There is a maximum of 10 profiles per save, each profile keeping track of currencies, high scores, medals, and unlocked appearances.

Overall, the save system provides player profile saving tied to unique player names, gamemode specific progression through cumulative stats, best runs, tracked stats, and level modifier unlocks, profile management for active profile tracking and multi-profile support, challenge medal recording with custom scoring evaluation rules, and safe fallback behaviour that creates a new save if no prior save game object exists.

For more information, the UI Leaderboard can be seen at 8.2.3 | Leaderboard Interaction.

4.0 | GRAPHICS

4.1 | TEXTURES & MATERIALS

To reduce GPU memory usage, the project relies heavily on two **1024x1024 JPEG** wood textures, one for curved surfaces and one for straight surfaces. These can be found in:

- **/Art/Materials/Woods/Curved/Textures**
- **/Art/Materials/Woods/Straight/Textures**

Transparent versions of these textures also exist, however they are more expensive and only visible during occlusion culling. Additional textures include:

Sticker Atlases

- 8x 2k texture atlases, 4 masked and 4 coloured
- **/Art/Models/Stickers**

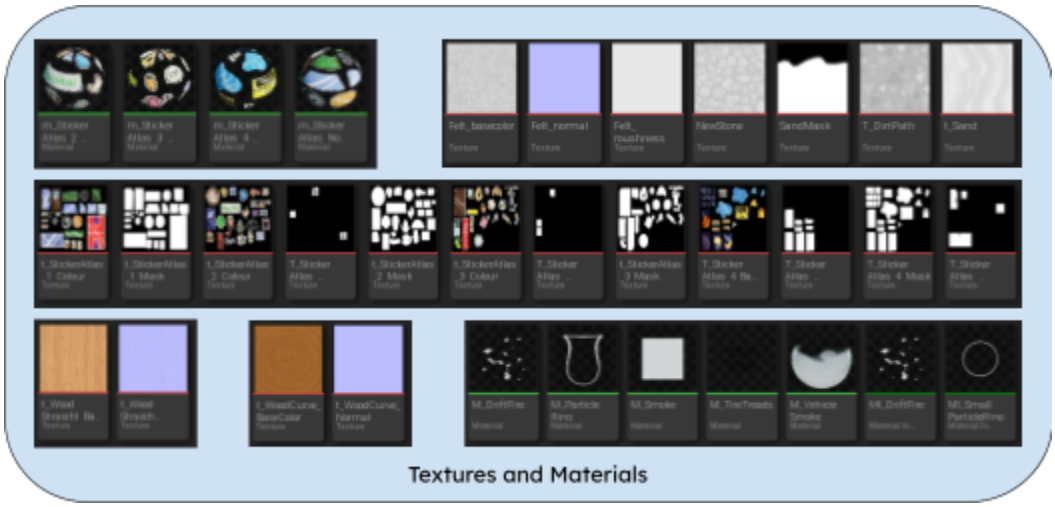
Environment Textures

- Single master material that handles variations in terrain appearance
- **/Art/Materials/Felt/Textures**

Unique Asset Textures

- Various unique textures ranging from 256x256 to 1024x1024
- Some are simple flat colours
- Used for specific props or one-off assets (such as UFO)
- Particle textures (250x250) and materials

All texture assets are driven by master materials designed to adapt one source texture into many different surface types, which reduces the material count and improves performance.

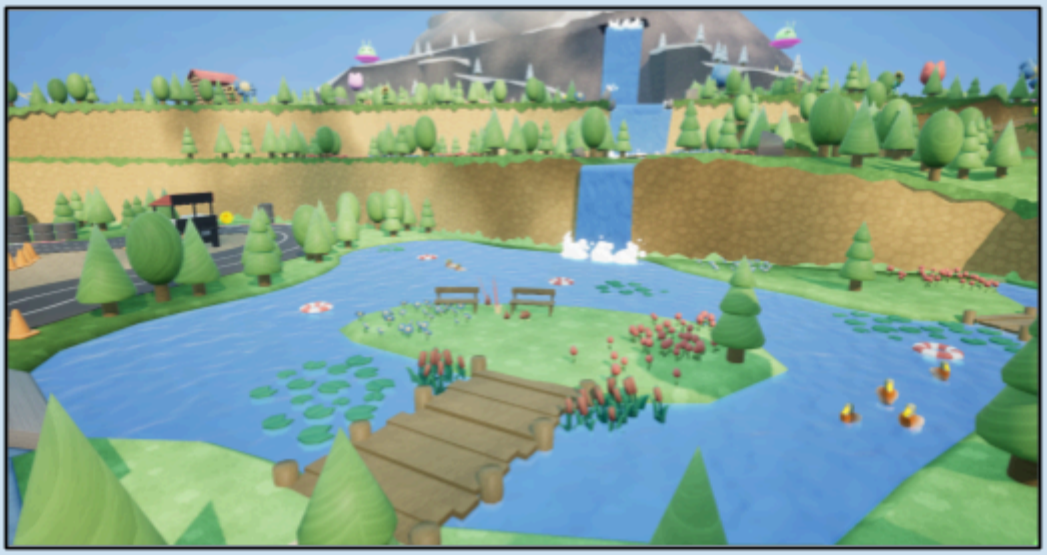


Textures and Materials

4.2 | POST-PROCESSING VOLUMES

Every level uses a post-processing volume to adjust saturation, contrast, gamma, and gain, making the scene seem more warm and vibrant. These colour tweaks add life to the environment and reinforce the bright, child-friendly aesthetic Tiny Transit is trying to achieve. It also helps with unifying the visual style and enriching the overall colour palette.

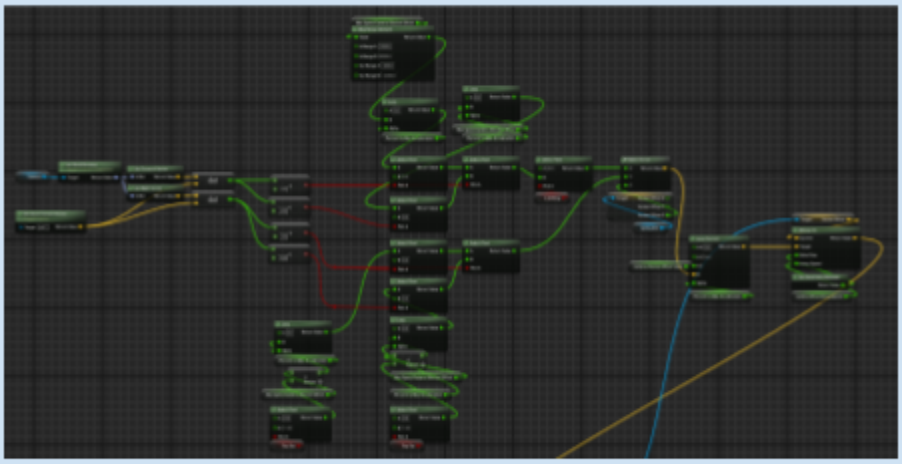
Post Processing



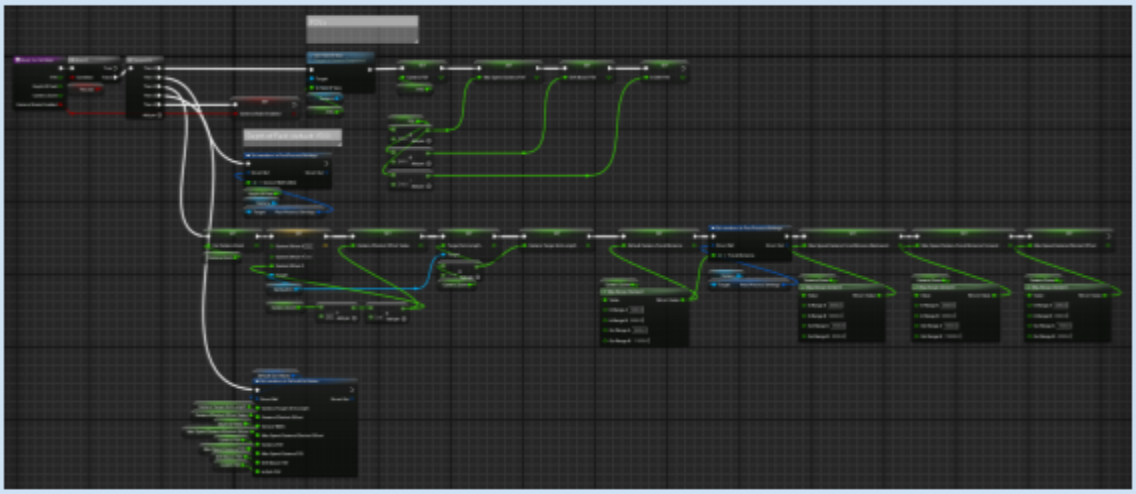
No Post Processing

4.3 | PLAYER CAMERA

The player camera has several exposed configurable parameters such as FOV, Depth of Field, Camera Zoom, and Camera Shake. While applying these settings may seem straight-forward, they influence each other in ways that require additional handling. The camera zoom, in particular, disrupts the focal distance, making the player appear out of focus. To compensate, the map range clamped node is used to translate the zoom values to focal distance values, but because the relationship isn't linear, the results are imperfect across the full range. To ensure consistent visual quality, the settings menu gives access to three preset configurations for close, normal, and far distances which have ideal camera zoom and FOV values to keep the player's care in focus. The camera settings are applied immediately (unless the car is currently in Tiny Car mode).



Camera Offset Calculations



Applying Camera Settings

5.0 | ARTIFICIAL INTELLIGENCE

In Tiny Transit, to make the level feel alive. AI vehicles will be navigating around the level…The AI vehicles will be following road rules that are standard in Australia. The AI vehicles can stop themselves from crashing into each other and stop at traffic lights or turn into a lane at intersections.

5.1 | AI VEHICLE

The AI Vehicle is the standard chaos vehicle pawn and will be controlled by an AI controller.

5.1.1 | AI VEHICLE MOVEMENT

In the pawn. The class takes in a spline reference that the pawn will use to follow. The spline is the road.

With the spline reference. The AI vehicle finds a direction on the spline that's closest to the vehicle. Depending on the speed of the vehicle. The range of the direction might be further out or closer.

```
FUNCTION GetPath()
    INNIT tangetDirection = SplineReference.FindLocationClosestToWorldLocation(Location)

    tangentDirection = tangentDirection.Normalize
    INNIT distance  = MapRangeClamp(currentSpeed,minSpeed,topSpeed, Range, Range*3 )
```

After the AI vehicle knows its direction. It gets the farthest point of the range and uses its location to find the closest point on the spline and converts it into a location.

```
    tangentDirection *= distance
    INNIT newLocation = tangentDirection + Location
```

The location gets converted into a delta, with the delta being used to calculate the steering needed for the vehicle

```
    INNIT LookAtRotation = newLocation - Location
    INNIT Steering = MapRangeClamp(LookAtRotation, -wheelangle, wheelangle, -1, 1)

    Return Steering
END FUNCTION
```

After finding the correct steering, the chaos pawn node SetSteeringInput, takes in the steering and is used on event tick…On Begin Play, the vehicle's throttle is pressed to make the vehicle move when the level loads.

The Vehicle pawn also calculates its own brake and throttle input. It does this by using the same formula as the steering but with a farther range. When it detects a curve on the spline. It will slow down if its speed is more than the minimum speed.

5.1.2 | AI VEHICLE SIGHT DETECTION

The AI Vehicle can process two objects, another AI vehicle in front of them, and traffic lights.

When it detects another AI vehicle in front of it. It will then calculate if it's in front and how close it is...If it is and it's too close the AI Vehicle will slam on the brakes and stop.

After a while when the AI vehicle in front moves away, the AI vehicle will move when it knows it's safe. The sight detection is handled in the AI controller.

—AIVehicleController—

```
FUNCTION AIVehicleDetection

        INNIT frontDirection = 0.94
        INNIT cautionDistance = 1500
        INNIT emergencyStopDistance = 700

        INNIT directionToNearestAIVehicle = NearestVehicle.Location - Location
        directionTonearestAIVehcile.normalize

        INNIT forwardVectorNearestVehicle = NearestVehicle.forward

        INNIT dotDirection = dot(forwardVectorNearestVehicle, directionToNearestAIVehicle

        IF dotDirection > frontDirection AND distanceToNearestAIVehicle < cautionDistance
                SlowDownAIVehicle()
                IF distanceToNearestAIVehicle < emergencyStopDistance
                        StopAIVehicle()

        ELSE
                ThrottleAIVehicle()
```

For traffic light detection. The AI vehicle, when it sees a traffic light in front of them, it will then see what light is selected. If it's a green light, the AI Vehicle will continue driving, if it's a yellow light, the AI vehicle slows down getting ready for the light to change to red. If the light is red, the AI Vehicle will stop before entering an intersection just in front of the traffic lights.

If the light is green but with a turn right. The AI vehicle will continue but follow the turn right lane.

—AIVehicle—

```
FUNCTION TrafficLightDetection
```

```
IF bCanSeeTrafficLight == true
        SWITCH TrafficLight
                CASE 1 REDLIGHT
                        StopAIVehicle()
                CASE 2 YELLOWLIGHT
                        SlowDownAIVehicle()
                Case 3 GREENLIGHT
                        ThrottleAIVehicle()
                Case 4 TURNRIGHTGREENLIGHT
                        ThrottleAIVehicle()
IF END
```

5.2 | TRAFFIC SYSTEM

5.2.1 | ROADS

There will be road pieces in the game that the designer can use to build out maps. Each road piece will have a spline on them for the AI vehicle to follow. There are two main road pieces, straights and curves. But have the same functionality.

At the start of each road piece there will be a collision that the AI vehicle overlaps with that sets the spline reference that the ai vehicle uses for getting the path to steer.

As of now, the game will have only one lane roads. And will follow Australian road rules.

```
—Road—
FUNCTION OnOverlap
        IF ColidingObject == AI Vehicle
                AIVehicle.splineReference = spline
        IF END
```

5.2.1 | INTERSECTIONS

There are two intersection roads in the game with one of them having two variants.

Starting with T-Intersection. It has the same road as a T-Intersection in real life.

On the left side of the road. There will be three collision boxes on all three entry ways of the T-Intersection. There will be six splines for all the different directions the AI vehicle can take. Depending on the collision box the AI Vehicle overlaps with. It will randomize between the two only paths the AI vehicle can take on the direction of the T the AI vehicle is in.

Other than T-Intersection, the game will also have a cross intersection and a cross intersection with traffic lights

Cross intersection follows the same logic as the T-Intersection. But with four directions and three possible paths the AIVehicle can take. This is for suburbs

Cross Intersection with traffic lights is the same as the Cross Intersection with no traffic lights. But with the added traffic lights and the removal of the ability for the AIVehicle to turn right in the intersection, making the possible paths two.

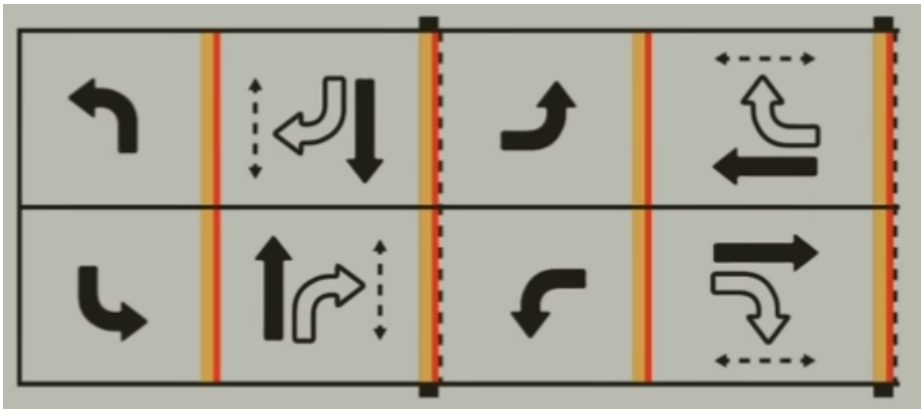
All Intersections are 1 lane as of now.

5.2.2 | TRAFFIC LIGHTS

With the Cross intersection with lights. There are the standalone traffic lights that connect to the cross intersection with the manager of controlling the lights being in the cross intersection blueprint.

The way traffic flows in the game follows the same ring and barrier diagram traffic systems use all around the world.

Ring and Barrier Diagram - Traffic Phases



Each phase lasts about the length of the cars passing through the intersection with breaks in between the 2nd - 3rd and 4th - 1st phase.

Converting this to game logic, each phase will have a countdown and when the countdown ends, the traffic control system moves onto the next phase.

To manage the traffic lights. Each traffic light will have an enum of traffic light values.

```
—TrafficLightsEnum—
ENUM TrafficLight
        RedLight
        YellowLight
        GreenLight
        GreenLightTurnRight
End ENUM

—TrafficLights—
Innit trafficLight

—CrossIntersectionWithLights—

Innit phaseTimer = 0.0
Innit trafficControlPhase = 1

Innit trafficLightUp
Innit trafficLightDown
Innit trafficLightLeft
Innit trafficLightRight

FUNCTION TrafficControl
        INNIT phaseOneTimerValue =  10
        INNIT phaseTwoTimerValue = 20
        INNIT transitionTimerValue = 5
        INNIT phaseTheeTimerValue = 10
        INNIT phaseFourTimerValue = 20

        PhaseTimer  = PhaseTimer + DeltaTime

        WHILE TRUE
                SWITCH trafficControlPhase
                        CASE 1
                                IF PhaseTimer >= phaseOneTimerValue
                                        trafficLightUp.trafficLight = GreenLightTurnRight
                                        trafficLightDown.trafficLight = GreenLightTurnRight
                                        trafficLightRight.trafficLight = RedLight
                                        trafficLightLeft.trafficLight = RedLight
                                        phaseTimer = 0
                                        TrafficControlPhase++
                        CASE 2
                                IF PhaseTimer >= phaseTwoTimerValue
                                        trafficLightUp.trafficLight = GreenLight
                                        trafficLightDown.trafficLight = GreenLight
                                        trafficLightRight.trafficLight = RedLight
                                        trafficLightLeft.trafficLight = RedLight
```

```
                phaseTimer = 0
                TrafficControlPhase++

CASE 3
        IF PhaseTimer >= transitionTimerValue
                trafficLightUp.trafficLight = YellowLight
                trafficLightDown.trafficLight = YellowLight
                trafficLightRight.trafficLight = RedLight
                trafficLightLeft.trafficLight = RedLight
                phaseTimer = 0
                TrafficControlPhase++
CASE 4
        IF PhaseTimer >= phaseThreeTimerValue
                trafficLightUp.trafficLight = RedLight
                trafficLightDown.trafficLight = RedLight
                trafficLightRight.trafficLight = GreenLightTurnRight
                trafficLightLeft.trafficLight = GreenLightTurnRight
                phaseTimer = 0
                TrafficControlPhase++
CASE 5
        IF PhaseTimer >= phaseFourTimerValue
                trafficLightUp.trafficLight = RedLight
                trafficLightDown.trafficLight = RedLight
                trafficLightRight.trafficLight = GreenLight
                trafficLightLeft.trafficLight = GreenLight
                phaseTimer = 0
                TrafficControlPhase++

CASE 6
        IF PhaseTimer >= transitionTimerValue
                trafficLightUp.trafficLight = RedLight
                trafficLightDown.trafficLight = RedLight
                trafficLightRight.trafficLight = YellowLight
                trafficLightLeft.trafficLight = YellowLight
                phaseTimer = 0
                TrafficControlPhase = 1
```

5.3 | AI OBJECT POOL MANAGER

With the AI vehicles driving around the map. There's a chance for the AI Vehicles to create stutter and lower performance…The way to solve this is by managing which AI Vehicles are enabled on and off the screen.

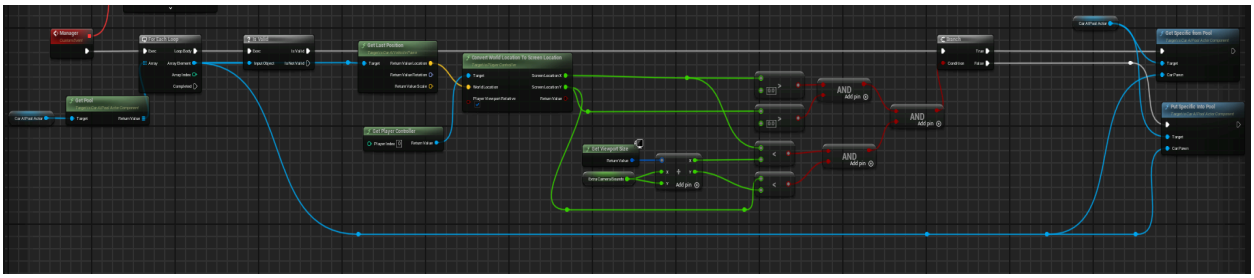
To do this, an object pool will be implemented for the AI vehicles. And a separate class that manages the object pool is used.

With the AIObjectPoolManager. It gets all the AIVehicles in the pool and checks the location and converts it to camera position. If the AIVehicle is near the bounds of the camera it will then be set active. If not it is set to not be active.

–AICarPawn–

Innit bIsActive = false

FUNCTION SetActive (active)
bIsActive = active
DisableCar()



6.0 | PHYSICS

Tiny Transit uses Unreal Engine 5.4.4's built-in Chaos physics for movement, collision handling, and interactions with physics objects. While 3.1 | Movement / Vehicle Physics documents the vehicle's internal systems in depth, this section describes how those systems behave within the Chaos simulation and the additional stabilization features implemented to ensure fun and predictable gameplay, especially for younger players.

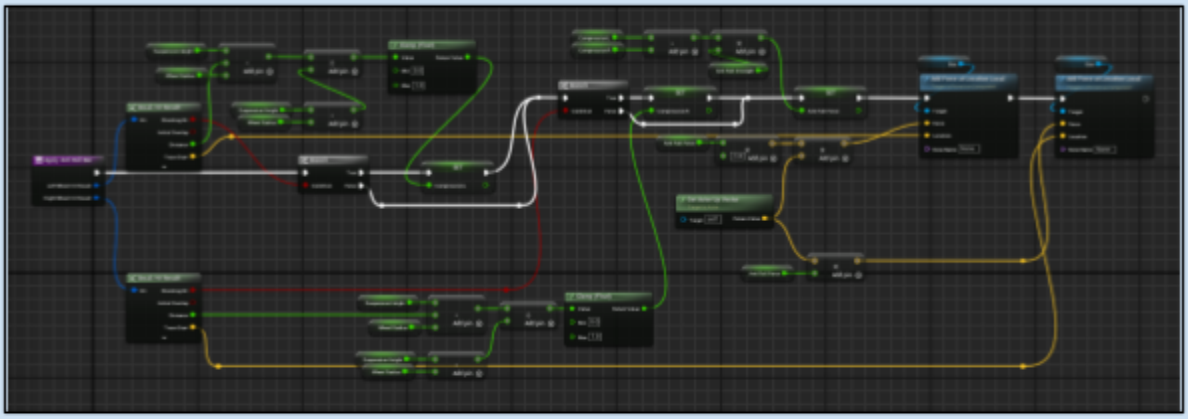
The vehicle system is fully custom-built and does not use Unreal Engine's built-in Chaos vehicle system. Instead, it relies on Chaos solely for rigidbody simulation, collision responses, and the application of forces and torques. All acceleration, rotation, wheel behaviour, suspension, and air control are handled manually within the vehicle's movement code.

During each play-test session, the Chaos physics system caused several issues when combined with the custom vehicle physics, which was frustrating and unpredictable. Collisions with physics objects or uneven terrain could make the car rotate and flip over, occasionally launching the vehicle into the air for several seconds. These interactions disrupted the pacing of the game and caused players to lose valuable time while delivering packages.

Flying Car Issue

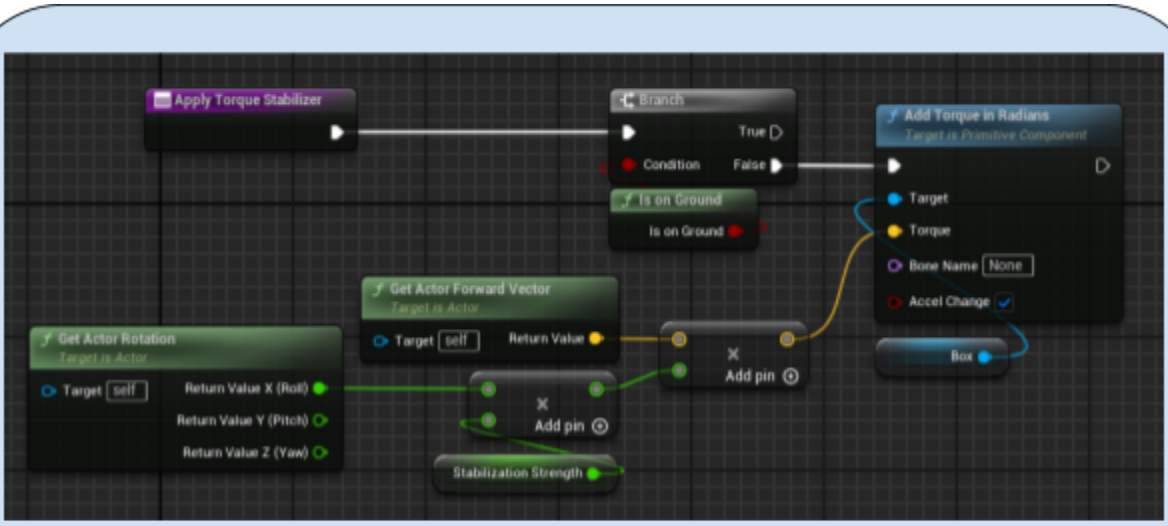
Adjusting the vehicle's linear and angular damping alone did not fully resolve the issues without compromising the car's responsiveness, so two stabilization systems were added on top of the existing Chaos simulation. Together, both systems maintain the arcade-like handling required for Tiny Transit while preserving the core behaviour of Unreal Engine's Chaos simulation. They provide predictable, physics interactions without modifying engine code or overriding the vehicle's code.

The Anti-Roll Bar checks which wheels are in contact with the ground and applies a counteracting force across the vehicle to reduce excessive roll. This force is calculated after the wheel acceleration forces have been applied, allowing it to complement rather than replace/completely remove the wacky physics.



Anti-Roll Bar Function

The Torque Stabilizer addresses rotational instability while the vehicle is airborne. When the car leaves the ground, Chaos can accumulate angular momentum from previous impacts or uneven surface contact. To counteract this, the stabilizer applies corrective torque in radians based on the vehicle's current roll angle, scaled by a stabilization strength value and directed along the vehicle's forward vector. This reduces unwanted spinning and flipping without interfering with the air-control of the vehicle.

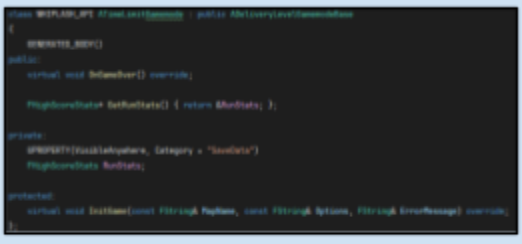


Torque Stabilizer Function

7.0 | MAPS & GAMEMODES

```
UCLASS()
1 derived blueprint class
class WHIPLASH_API ADeliveryLevelGamemodeBase : public AGameModeBase
{
    GENERATED_BODY()

public:
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "Modifiers")
    FGameplayTagContainer SelectedModifiers;  Unchanged in assets

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Settings")
    bool IsFreeRoam;

    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Time")
    int RemainingGameTime;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "Settings")
    float StartTime = 5.0f;  Unchanged in assets

    UPROPERTY(BlueprintAssignable, BlueprintCallable)
    FOnTimeUpdated OnTimeUpdatedDelegate;  7 blueprint usages

    FOnGameOver OnGameOverDelegate;

    FTimerHandle RemainingGameTimeHandle;

    virtual void BeginGame();
    virtual void DecrementRemainingGameTime();
    virtual void AddMoreRemainingGameTime(int AddedTime);

    UFUNCTION(BlueprintCallable)
    virtual void OnPauseGame() const;  1 blueprint usage
    UFUNCTION(BlueprintCallable)
    virtual void OnResumeGame() const;  2 blueprint usages

    virtual void OnGameOver();

    UFUNCTION(BlueprintCallable)
    virtual void QuitLevel();  2 blueprint usages

protected:
    UPROPERTY(VisibleAnywhere, Category = "Reference")
    UWhiplashGameInstance* GameInstance;

    virtual void BeginPlay() override;
    EMedalRank EvaluateScore(int FinalScore);

private:
    void ActivateMatchingModifierActors();
};
```

Base Gamemode Class



Time Limit Gamemode

There are 2 maps and 3 gamemodes that make up the initial prototype of Tiny Transit. The level select map allows the player to learn the ropes of the game through the on-screen dialogue tutorial, customize their car's appearance in the garage, and select a gamemode and difficulty to play. The three modes are Time Limit (set amount of time to get the highest score), Heat Mode (starting and completing deliveries adds to your maximum time limit), and Free Roam (explore the map without limits). Time Limit also has a hard mode, unlocked through earning a gold medal during any of the time limit runs. It adds obstacles that make it harder for the player to deliver packages and play the game, seen in 3.4.3 | Modifier Specific Hazards. The main level is where the game takes place, it is full of physics objects, delivery locations, and challenges that the player can interact with to earn money.

Each gamemode derives from the **DeliveryLevelGamemodeBase** AGameModeBase class, which has basic functionality for decrementing remaining time, beginning and ending the game, as well as evaluating score medals based on the player's results.

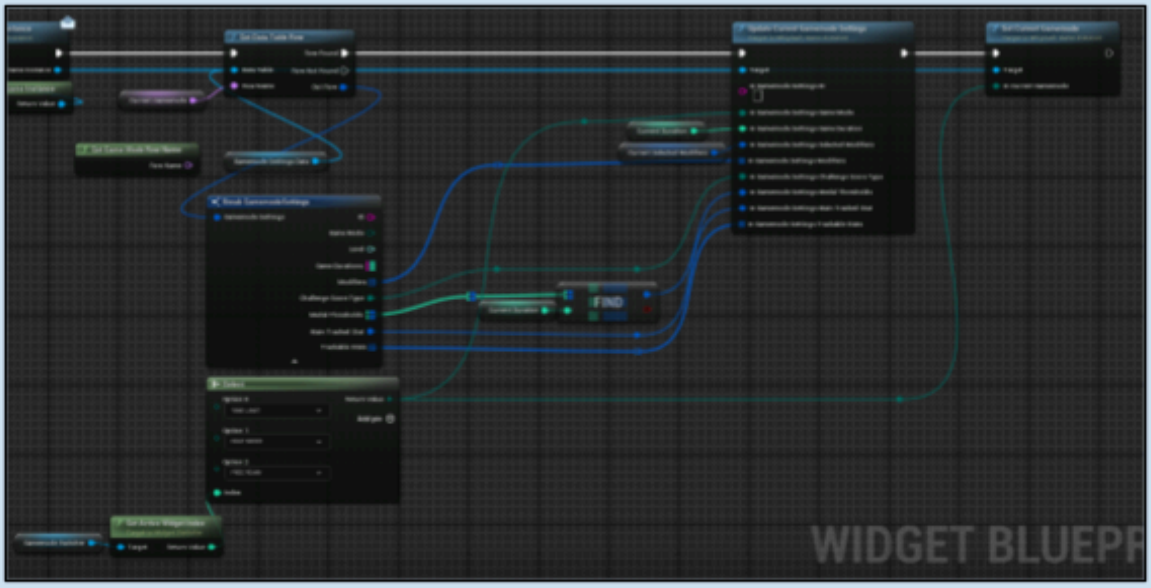
Due to Unreal Engine not allowing gamemode changes during runtime, there is only one gamemode class: Time Limit. Depending on what gamemode the player chooses in level select several functionalities may be enabled/disabled, such as no longer decrementing time if the player chooses Free Roam or the ability to add more time when picking up/delivering packages in Heat Mode.

Gamemode Data

Each gamemode has a **FGamemodeSettings** UDataAsset which designers can edit to adjust the available durations, difficulties (modifiers), medal thresholds, scoring type (highest or lowest value), tracked statistics and the main tracked statistic (used for scoring medals).

When a new gamemode is selected the UI updates the Game Instance's Gamemode Settings data asset to be equal to the new one.

Gamemode Data - C++



Gamemode Select UI Widget

8.0 | UI INTERFACE

With the User Interface in the game there are mainly 10 different User interface screens, 7 in the main menu and 3 in the main level.

Unreal Engine's Common UI plugin will be used when creating the UI in the game. Mainly for key prompts and input action inputs.

8.1 | UI LAYOUT ARCHITECTURE

Each Unreal Engine widget will be a balance between normal user widgets in UE5 and common UI widgets.

8.1.1 | MAIN MENU START SCREEN

The blueprint for the start screen will be a activatable common widget and will have the following layout hierarchy:

-Root (WidgetNameStartScreen)
–Overlay
	-Size Box (Logo Size Box)
		-Image (Logo Image)
	-Horizontal Box (Key Prompt Holder)
		-Size Box
			-Horizontal Box
				- Common UI Action Button
		-Spacer
		-Common Text (Key Prompt Name)
	-Button (PlayButton)

The display of the start screen will show the game title and a key prompt at the bottom to transition when pressed to the main menu selection screen

Figure 1 (Main Menu Start Screen)

8.1.2 | MAIN MENU SELECTION SCREEN

The selection screen blueprint will be a common activactble widget and will have the following hierarchy:

-Root (WidgetNameSelectionScreen)
-Grid Box
-Vertical Box
-Horizontal Box
-Spacer
-Vertical Box
-Spacer
-Size Box (Logo Size Box)
- Image (Logo Image)
-Spacer
-Vertical Box (Buttons Vertical Box)
- Button (Start Button)
- Button (Leaderboard Button)
- Button (Settings Button)
- Button (Quit Button)
-Size box (common UI input size box - Hidden
-Horizontal Box
-Common UI Action button (Back)
- Custom Widget - UserProfileMenu

The selection screen will have the logo of the game and 4 buttons the player can press. Start, Leaderboard, Settings and Quit…The user can also press a back button input and go back to the start screen.

Figure 1 (Main Menu Selection Screen)

8.1.3 | CREDITS

The credits menu is part of the main menu and displays the credits of the game, the player can return from the credits menu to selection screen

Figure 1 (Credits)

8.1.4 | SETTINGS

The settings user interface will consist of six parts. One main part that holds the other five parts.

The main part being the actual settings menu while the other parts being each setting category. The settings that will be in the game are (Controls, Graphics and Audio), Widget keybind input window and a Widget tab list to navigate the categories

The main settings blueprint will be a common activatable widget and will have the layout:

-Root (WidgetNameSettings
-Overlay
-Sizebox
- Common Border
-Vertical Box
-Spacer
-SettingsTabList - Custom Widget
-Horizontal Box (Tab horizontal Box)
-Spacer
-Common Activatable Widget Switcher
-Control Settings Widget
-Graphic Settings Widget
-Audio Settings Widget
-Size box (common UI input size box) - Hidden
- Horizontal Box
-Common UI Action button (Back)
-Common Bound Action Bar
-InputNewKeybind Widget

The Main settings menu will display a tab list and a settings window. The settings window will change depending which category in the tab list is selected.

If the control category is selected. The player can change their keybinds and selecting a keybind to change will make the InputNewKeybind Widget show up

Figure 1 (Settings Example 1)

Figure 2 (Settings Example 2)

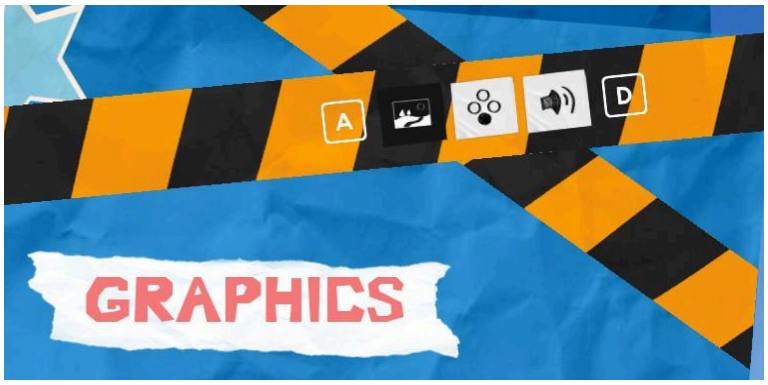
Figure 3 (Settings Input New Keybind)

8.1.4.1 | SETTINGS CATEGORY TABLIST

The settings tab will be a small part to the main settings widget and will be used to navigate and maintain attached categories. It will be a Common Tab List Widget Base and Its layout will be:

-Root
- -Overlay
  - -Vertical Box
    - -Horizontal Box (Tab horizontal box that will hold the categories)
    - -Horizontal Box
      - -CommonActionWidget (Left navigation Input)
      - -Custom Text Widget (Left Arrow “<”)
      - -Border
        - -Custom Text Widget (Settings Category Name)
      - -Custom Text Widget (Right Arrow “>”)
      - -CommonActionWidget (Right navigation Input)

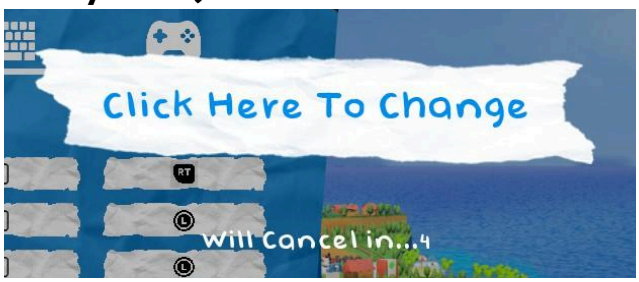
Figure 1 (TabList)



8.1.4.2 | INPUT NEW KEYBIND

The input new keybind widget will be a user widget and will have the layout:
-Root (WidgetNameInputNewKeybind)
- -Uniform Grid Panel
  - -Image (Background Image)
  - -Vertical Box
    - -Overlay
      - -Image (Background Input Image)
      - -Input Key Selector
    - -Custom Text Widget (prompt text)
    - -Custom Text Widget (cancel timer text)

Figure 1 (Input New Keybind)



8.1.4.3 | CONTROL SETTINGS

The control settings widget will attach to the main settings menu and will be a common activatable widget. This is used to list and change keybind settings...

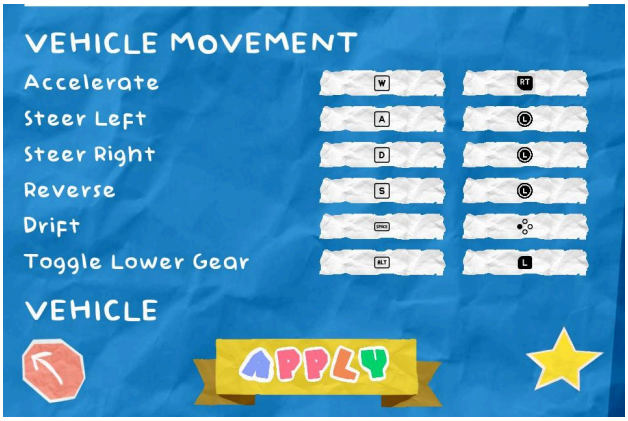
The layout:

-Root(WidgetNameControlSettings
	-Overlay
		-Sizebox
			-Vertical box
				-InputLogoBox
				-Spacer
			-CommonHierachicalScrollBox
				-Vertical Box (Keybind Categories)

This is the main base layout of the widget, but there is 2 other widgets that connect to the control widget when at runtime.

Category setting widget and **setting widget.**

(Control Settings)



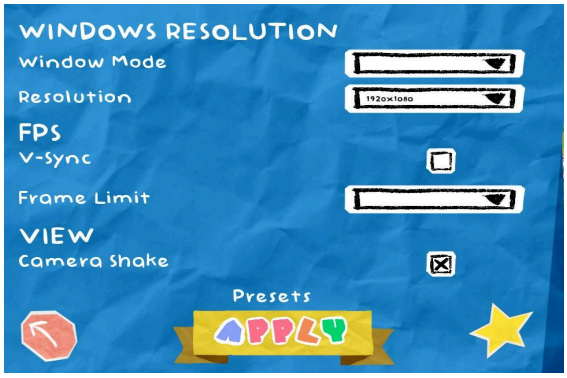
8.1.4.4 | GRAPHIC SETTINGS

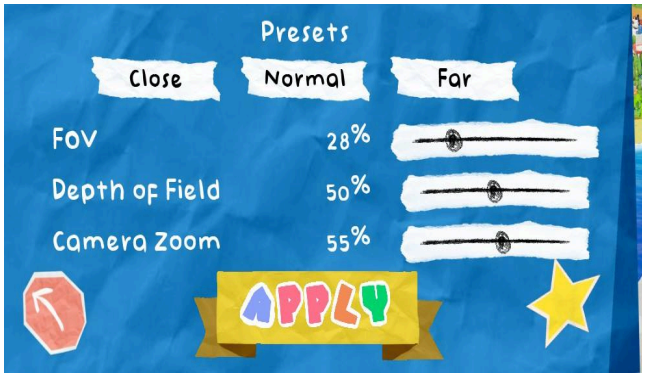
The graphic settings widget will attach to the main settings widget and will be a common activatable widget. The purpose of the widget is to list and change graphic settings. The layout:

With new graphic settings added to the ui, the layout will change. But as of now. Only four settings are listed.

-Root(WidgetNameGraphicSettings)
- -Overlay
- -Size box
- -Vertical Box
- -Spacer
- -CommonHierachicalScrollBox
- -Vertical Box (GraphicsBox)
- -Common Text (Category One Name “Resolution”)
- -Custom Widget - SettingsComboBox (Window Mode)
- -Custom Widget - SettingsComboBox (Resolution)
- -Common Text (Category Two Name “FPS”)
- -Custom Widget - SettingsCheckbox (Vsync)
- -Custom Widget - SettingsComboBox (fps limit)

Figure 1 & 2(Graphics Settings)





8.1.4.5 | AUDIO SETTINGS

The Audio settings widget will attach to the main settings widget and will be a common activatable widget. The purpose of the widget is to list and change the audio settings in the game. The layout:

With new audio settings added to the ui, the layout will change. But as of now. Only three settings are listed.

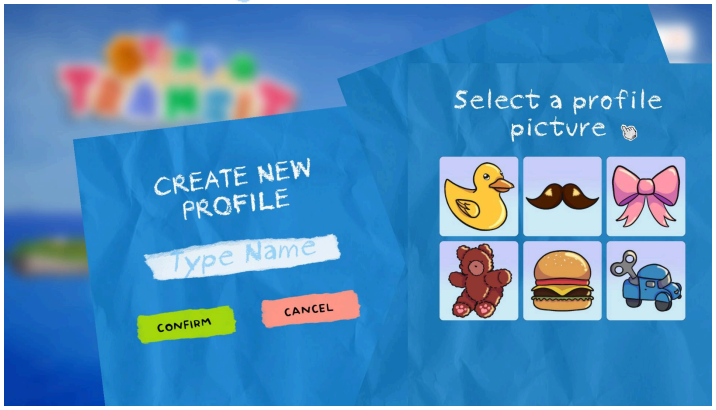
-Root(WidgetNameAudioSettings)
-Overlay
-Size box
-Vertical Box
-Spacer
-CommonHierachicalScrollBox
-Vertical Box (AudioBox)
-Custom Widget - SettingsSlider (Master)
-Custom Widget - SettingsSlider (Sound)
-Custom Widget - SettingsSlider (Music)

Figure 1 (Audio Settings)

8.1.5 | USER PROFILE MENU

The user profile menu will attach to the main menu selection screen. Its main purpose is to allow the player to create, delete and switch to existing profiles

Figure 1, 2, 3 & 4 (User Profile Menu)



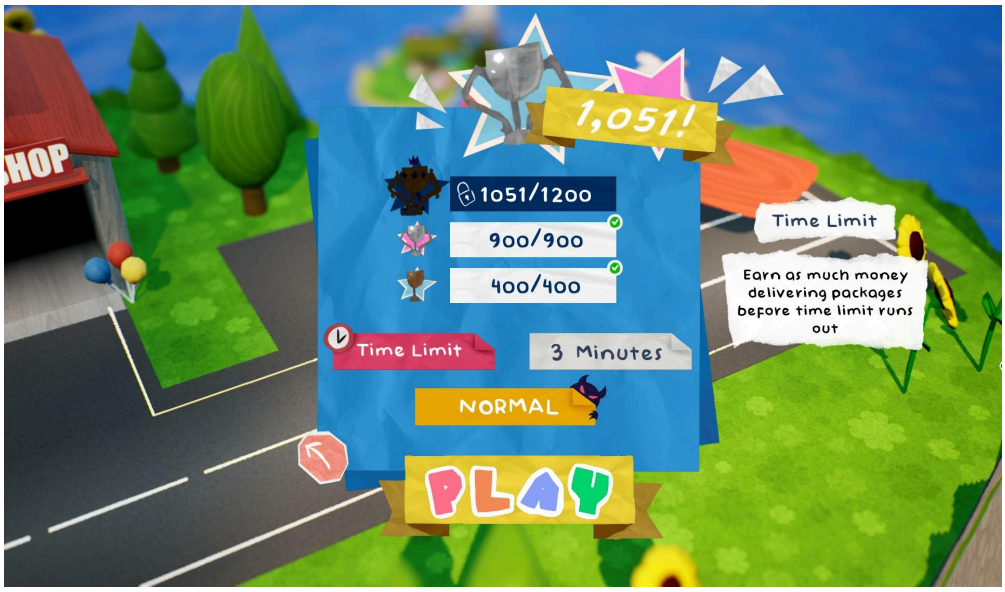
8.1.6 | UNLOCKABLE SHOP

The unlockable shop widget is part of the select level. Its main purpose is to list all the cosmetics for the player vehicle in the game. That the player can buy…The player can earn shop coins in every run. And each coin is $100 each earnt in a run.

Figure 1 (Unlockable Shop)

8.1.7 | CHOOSE GAMEMODE

Figure 1, 2 & 3 (Choose Gamemode)



8.1.8 | MAIN LEVEL HUD

The main UI in the game, it displays the information to the player. There are Five elements to the main Level HUD.

The timer and money counter, the circular weight scale, the minimap, the package objectives and the bonus money popup…The Layout of the main level hud:

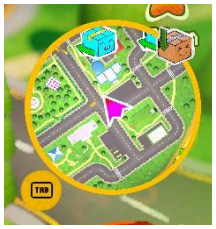
-Root
-Sizebox
-Overlay
-Overlay
-Vertical Box (Values)
- Size Box (Money Size Box)
- Border (MoneyBorder)
- Horizontal Box (MoneyBox)
- Image (Money Image)
- Text (Total Money)
- Size Box (Clock Size Box)
- Border (ClockBorder)
- Horizontal Box (ClockBox)
- Image (Clock Image)
- Text (Clock Money)
- Custom Widget - Minimap
- Custom Widget - Objectives
- Custom Widget - CircularWeightScale
- Size Box (Bonus Money Size Box)
- Overlay (Bonus Money Overlap)
- Image (Bonus Money Image)
- Text (Action Text)
- Text (Bonus money total)
- Custom Widget -Loading Screen
- Custom Widget - Leaderboard Victory Screen
- Custom Widget - Game Over Screen
- Custom Widget - Start Countdown

Figure 1 (Main Level Hud)

8.1.8.1 | TIMER AND MONEY COUNTER

The timer and money count is attached to the main level hud and rests in the top left corner of the player's screen…Its main job is to display the timer that's in the level and the total money earned in a run.

8.1.8.2 | MINIMAP



The Minimap widget is a part of the main level hud and rests at the top right of the player's screen. The main role of the minimap is to display to the user where they are on the map and the directions to the package they have active. Multiple packages that are selected will display on the minimap.

8.1.8.3 | PACKAGE AND MODIFIERS

Package And Modifiers, originally called phone, is a widget that is part of the main level hud. It rests right underneath the minimap on the right. The main purpose of this widget is to display to the player the amount of packages being held, and modifiers that are enabled

The player can cycle through the packages, drop active packages, or set a new active package.

8.1.8.4 | BONUS MONEY POPUP

Bonus money popups are a small attachment to the main level hud. It can be displayed anywhere around the hud. It is set to invisible by default…When the player does something else like running over furniture, landmarks. The bonus money widget will popup for a few seconds, displaying to the player, they earned bonus money

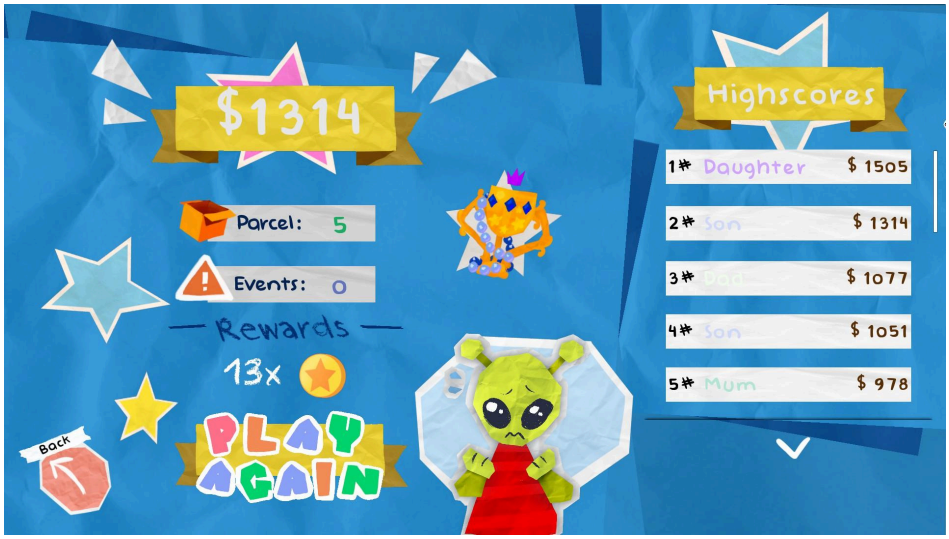
8.1.8.5 | ISLAND MAP

The Island Map widget is part of the main level hud and its main purpose is to display the pick up and drop off locations for packages…The map also displays challenge gates and dialogue quests. The map also reflects the time in the game.

8.1.9 | VICTORY SCREEN LEADERBOARD

Victory screen shows up when a run is complete. Showing the stats the player has earned…The player can go back to the main Menu or try another run again

Figure 1 (Leaderboard)



8.1.10 | PAUSE MENU

The pause menu shows up when the player pauses the game by pressing the pause keybind...The pause menu shows up in the main level and in the select level

Figure 1 (Pause Menu)

8.1.10 | LOADING SCREEN

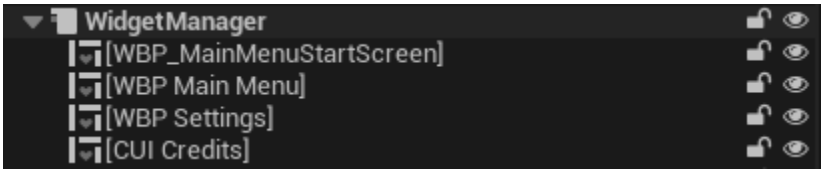
The loading screen shows up when a level is opened and its main purpose is to show something on screen while background assets / map loads.

Figure 1 (Pause Menu)

8.2 | UI INTERACTION - UNDER THE HOOD

8.2.1 | MAIN MENU INTERACTION

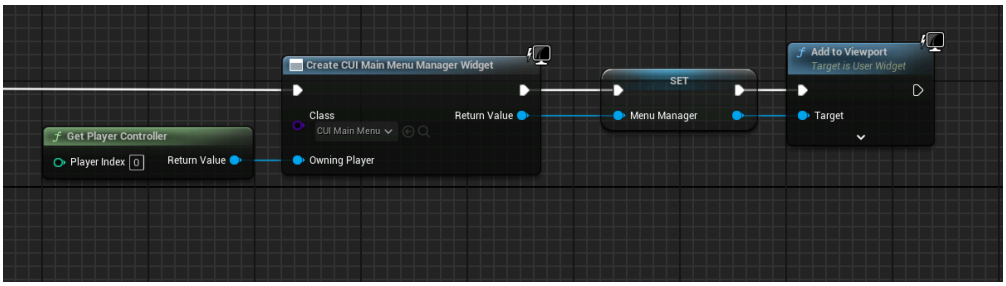
All the menus in the main menu (Start Screen, Selection Screen, Credits and Settings) are held in a manager widget, a common user widget... The manager widget uses a child called Common Activatable Widget Switcher. The child holds all the widgets that can be activated.



Only one widget can be active on the switcher and all the others deactivated and collapsed.

This is good since it eliminates the need to create and add a lot of widgets to the viewport in the hud every time creating a lot of draw calls.

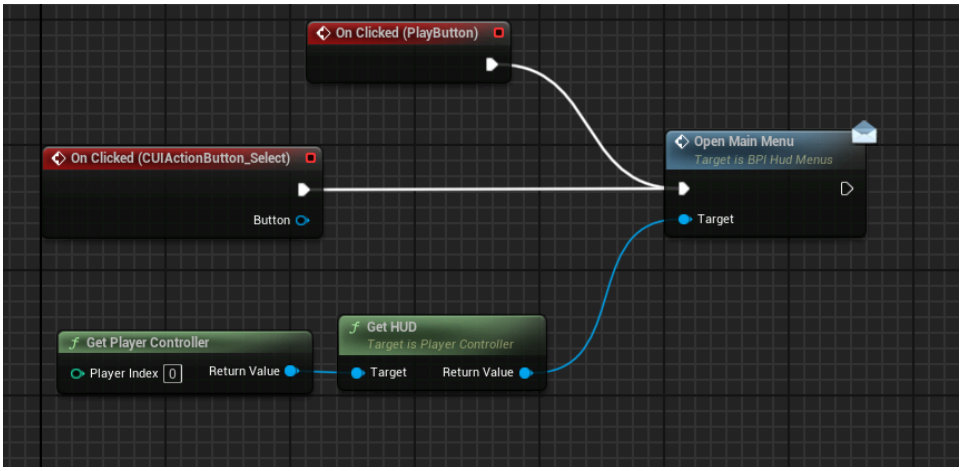
With this, only one widget can be created and added to viewport while the hud communicates with the widget manager to activate a widget it needs activated.

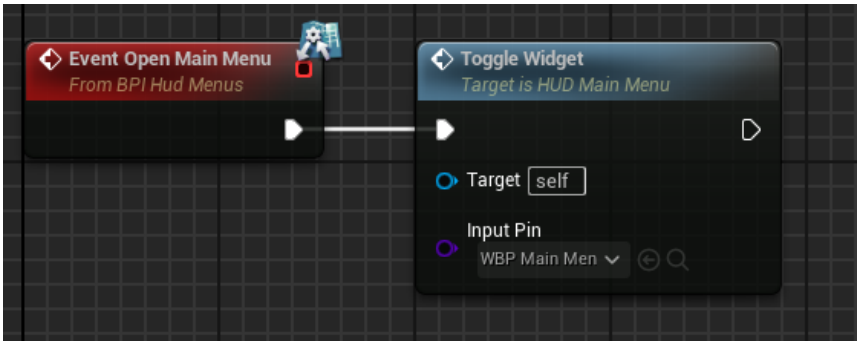


8.2.1.1 | START SCREEN INTERACTION

The start screen only has a button that can be clicked or an input listener that if pressed triggers the main menu manager to switch the active widget to the selection screen.

These does this by sending a signal to the hud by using interfaces to activate the selection screen.

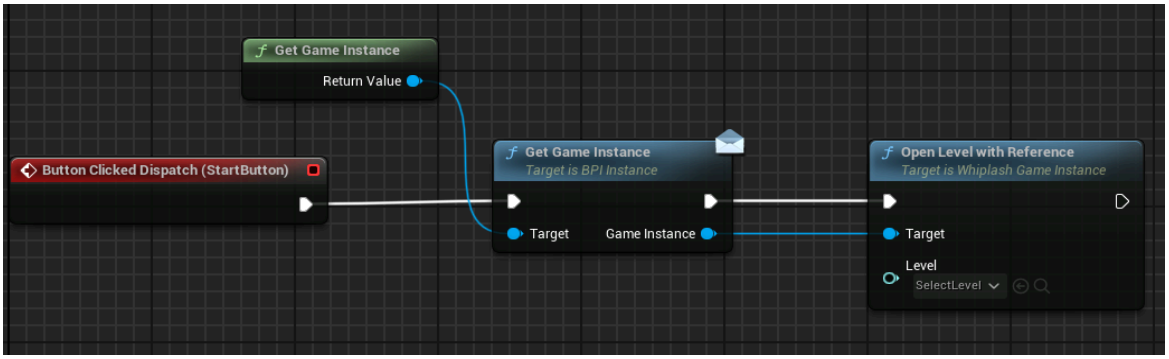




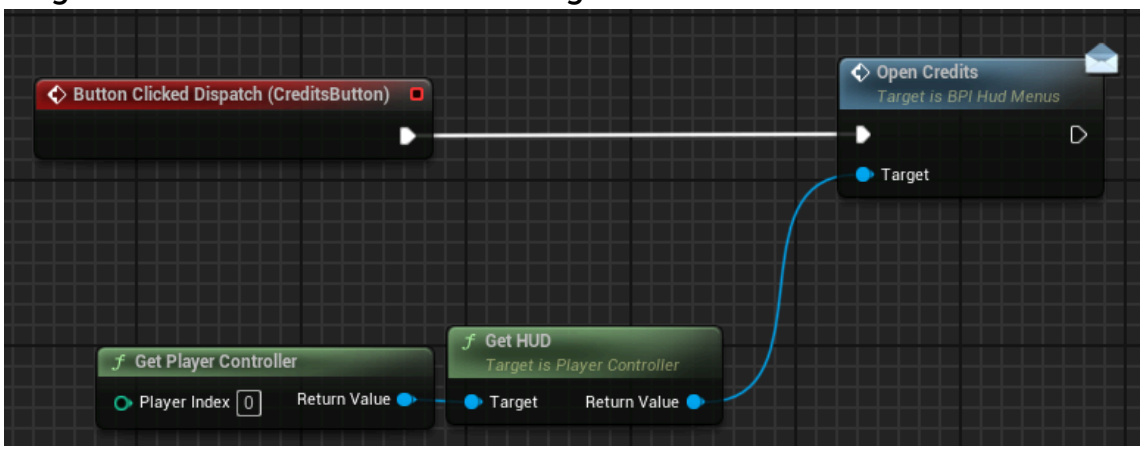
8.2.1.2 | SELECTION SCREEN INTERACTION

The selection screen hosts four buttons. (Start, Credits, Settings and Quit)

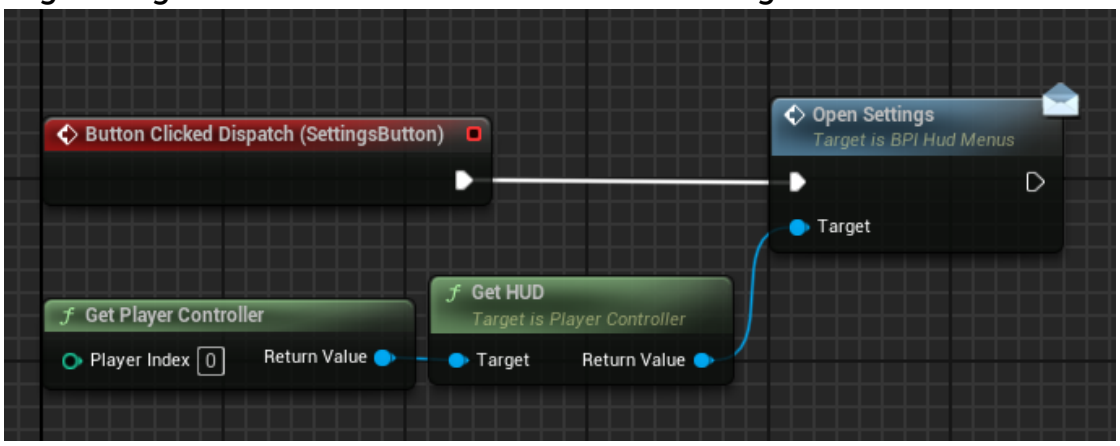
Pressing start opens the select level map



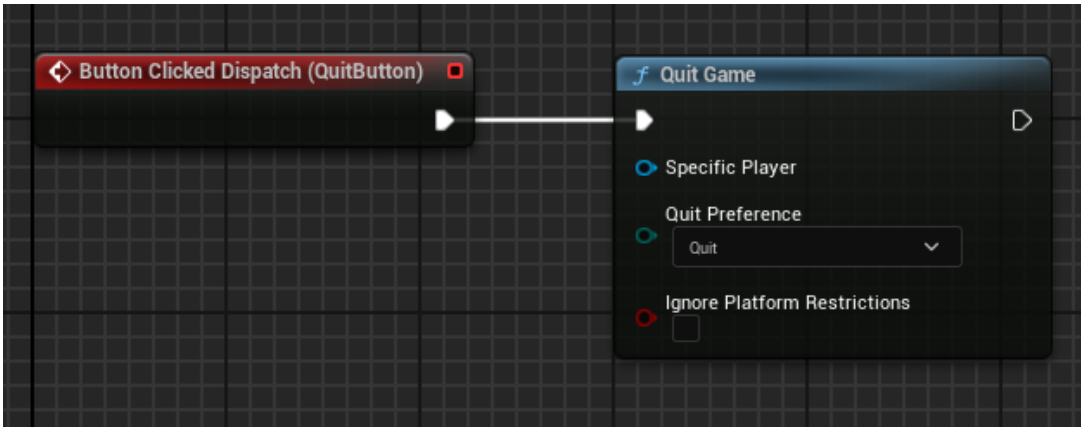
Pressing Credits tells the main menu manager to active the credits menu



Pressing Settings tells the main menu to active the settings menu



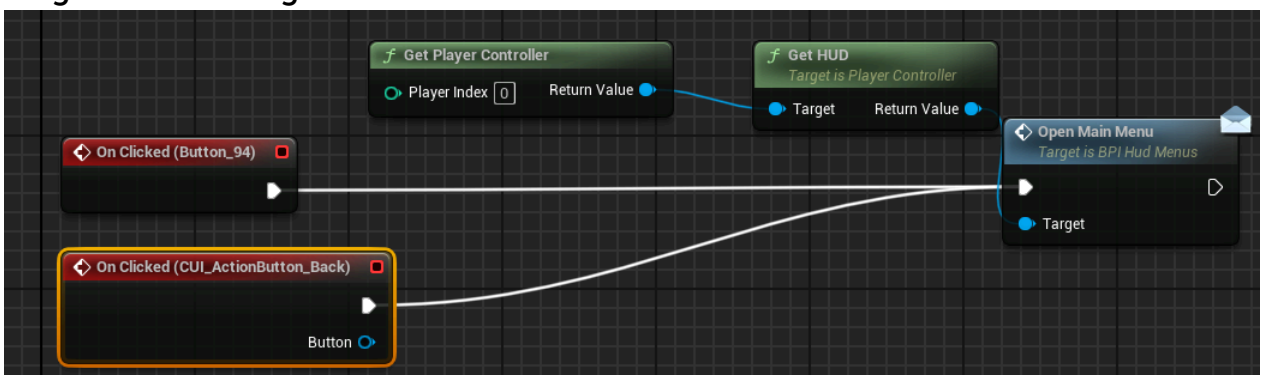
Pressing Quit, quits the game



The Selection Screen also hosts a profile widget. Then when clicked hosts options to either switch profiles, delete or create new profiles.

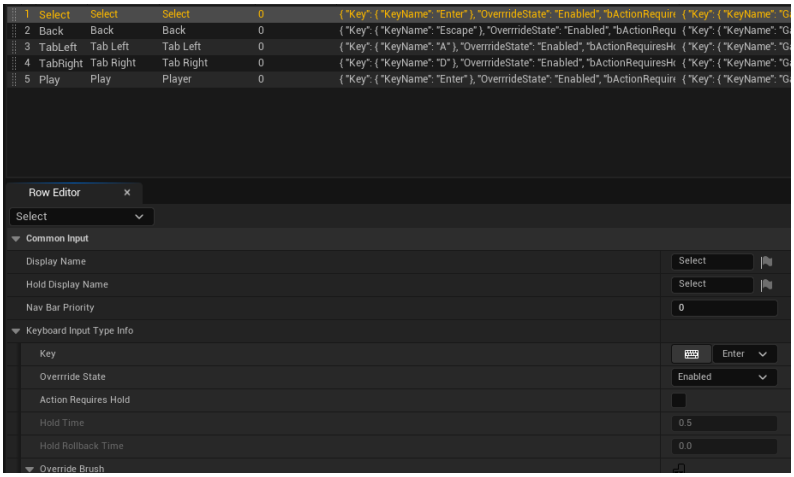
8.2.1.3 | CREDITS MENU

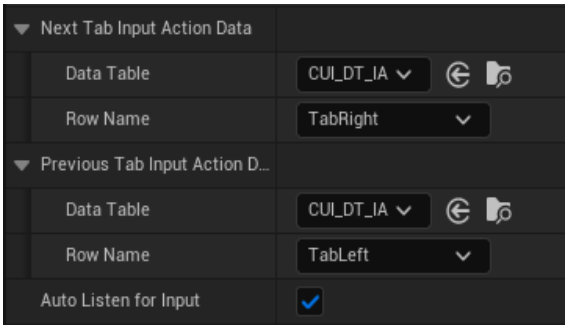
The credits menu has only one button and pressing that button sends a signal to the widget menu manager to activate the selection screen to return back.



8.2.1.4 | SETTINGS MENU CATEGORY SWITCHER

The settings menu has three categories, (Graphics, Controls and Audio). To switch between the categories. The settings menu has a child, CommonTabList. Part of the Common UI plugin it is used with Common UI input to switch between tabs.





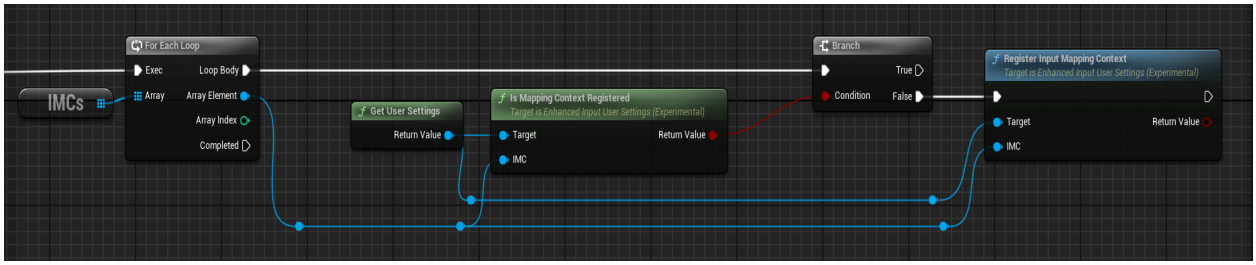
If one of the keys selected in the common UI generic actions. It will automatically switch depending on what's on tab left or right.

8.2.1.5 | CONTROL SETTINGS

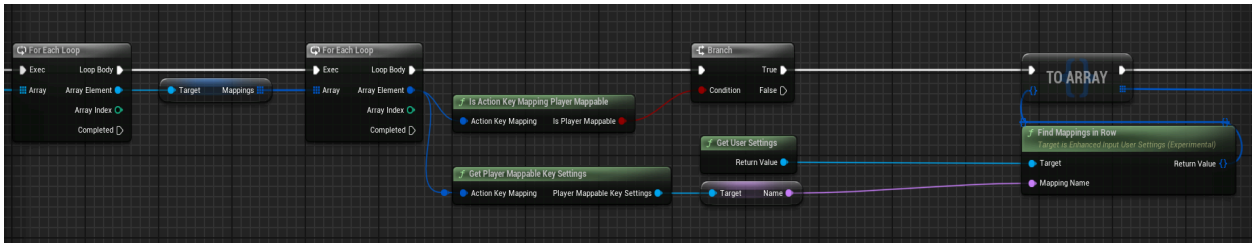
Control settings displays the key mappings in the input actions that have “Player Mappable Key Settings’ turned on in the user settings section.

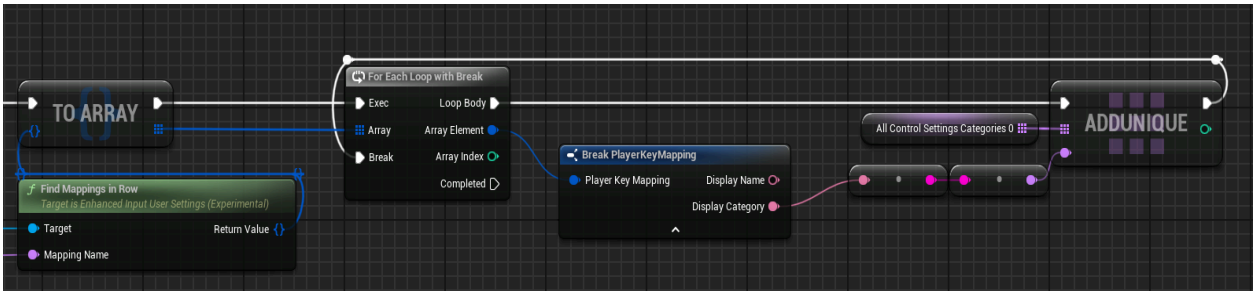
With the setting enabled, each input action will need a category and name.

With every input action having the setting turned on and configured. All that’s left is to register the input mapping contexts that hosts the input actions if they are not already registered.



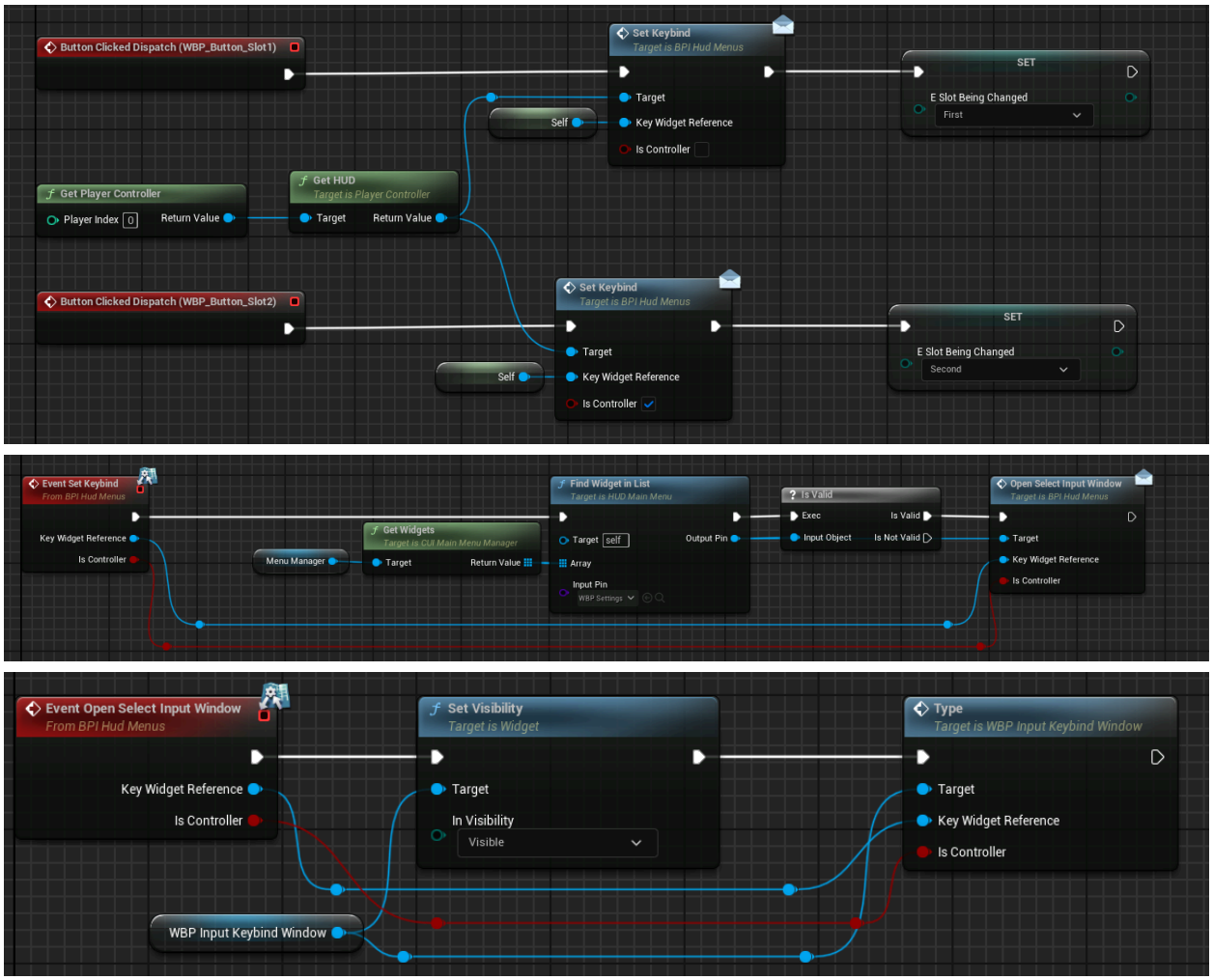
The settings menu will then grab the all the registered mappings from the IMC and start adding the categories and settings name to the menu



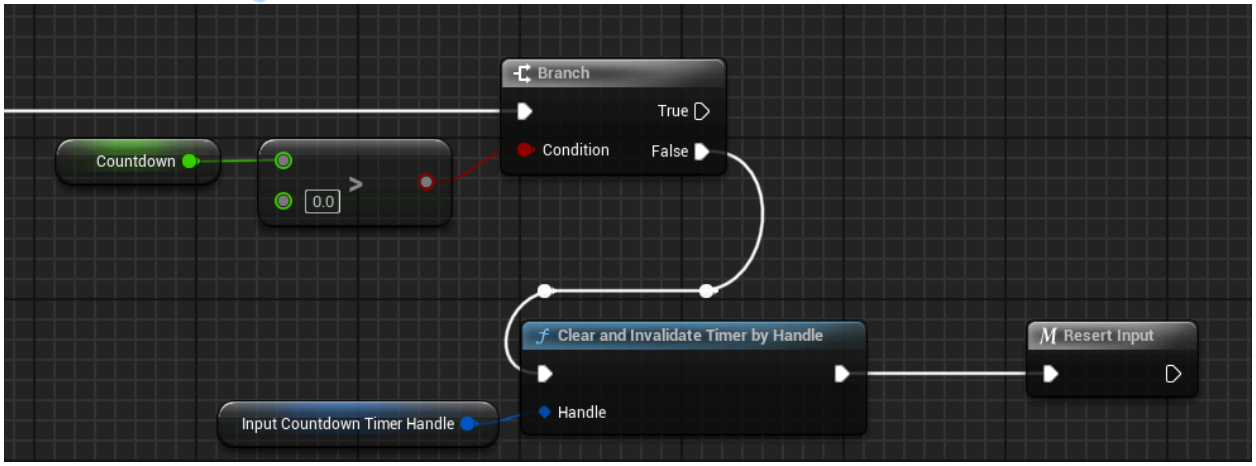


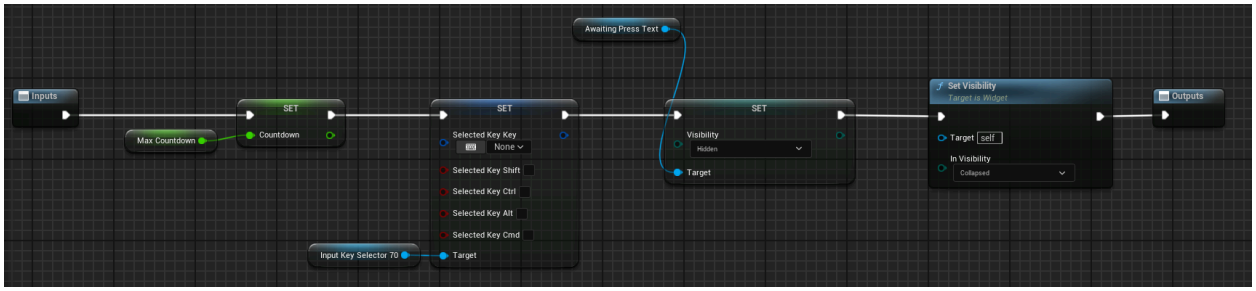
The controls menu will be able to have rebindable keys in the menu. For this to function. Slot 1 and slot 2 in settings will have a button that can be clicked.

When clicked, it sends a signal to hud to let the settings menu know to shop the rebinding widget on screen. While doing this. It will send the key and slot that is being rebinded.

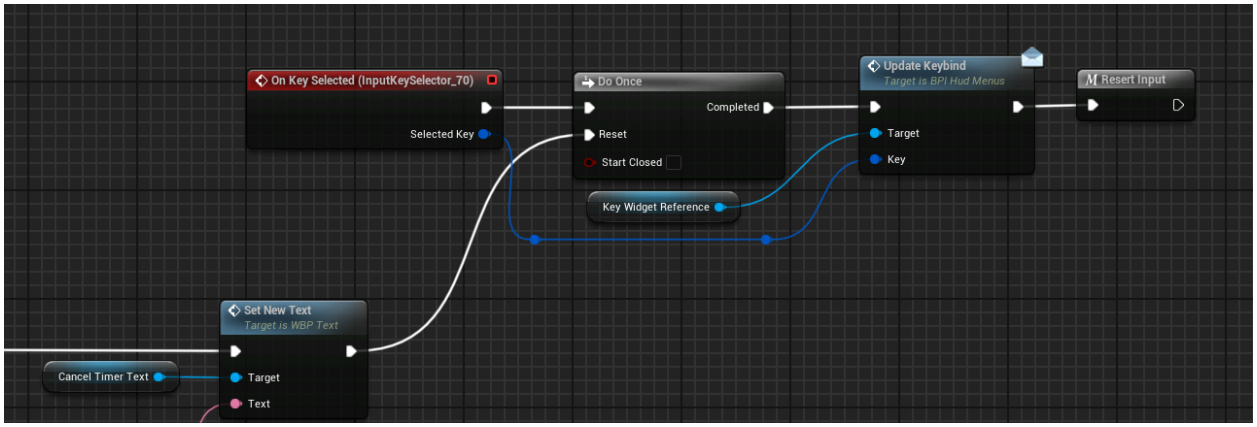


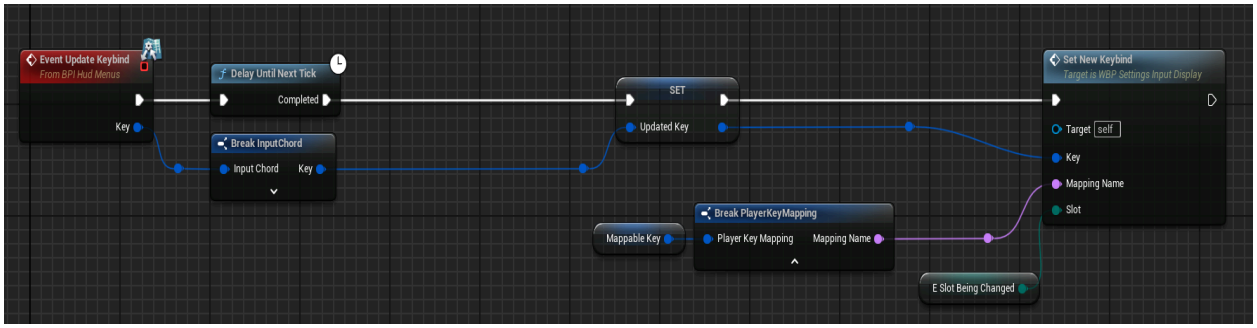
With the rebinding widget open. There will be a countdown that display for the player to put the new key before the countdown hits 0 and cancels rebinding and collapses the widget.





If the player presses a new key. The rebinding widget closes and a signal gets sent to the slot and setting that needs to be changed to be overwritten.





8.2.1.6 | GRAPHIC SETTINGS

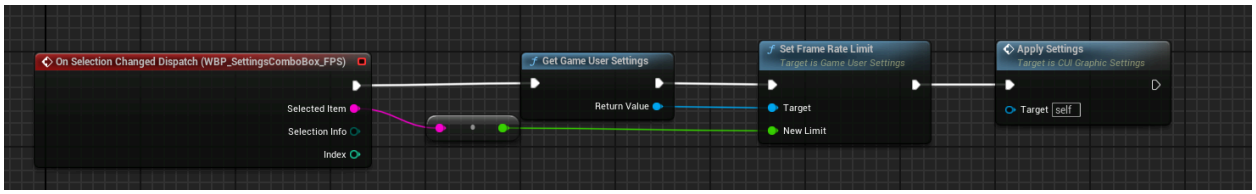
The Graphic settings will have seven graphic settings with a view preset option.

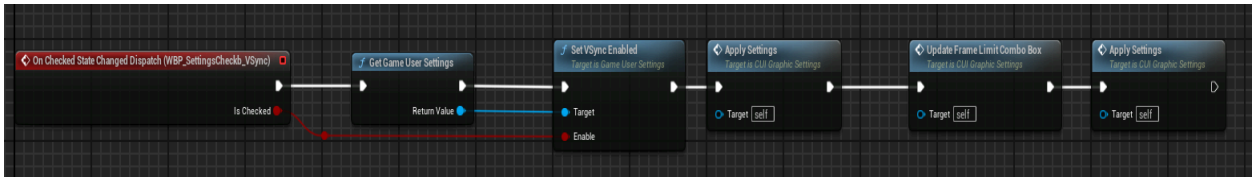
The graphics settings that can be changed and displayed

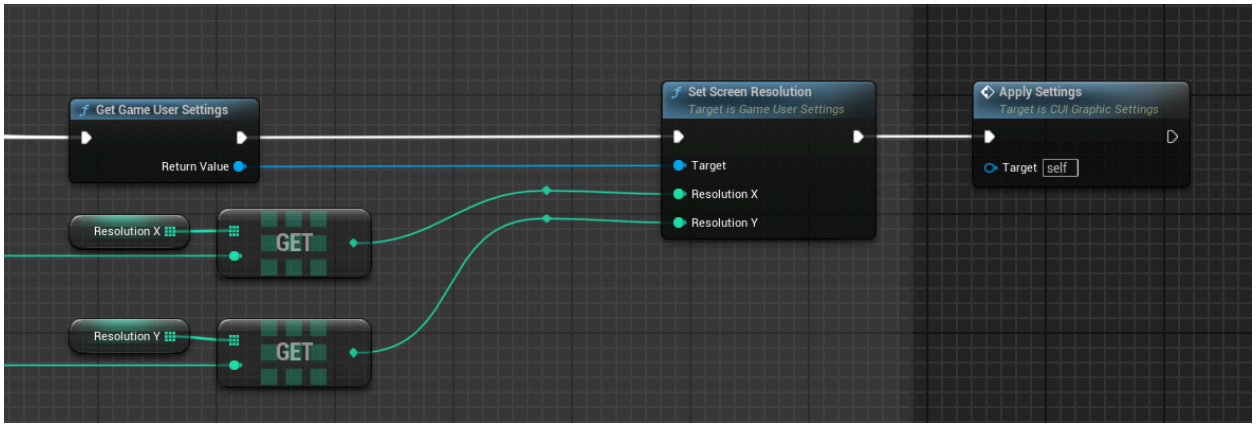
- Window Mode
- Resolution
- V-Sync
- Frame Limit
- Camera Shake
- Camera FOV
- Camera Depth of Field
- Camera Zoom

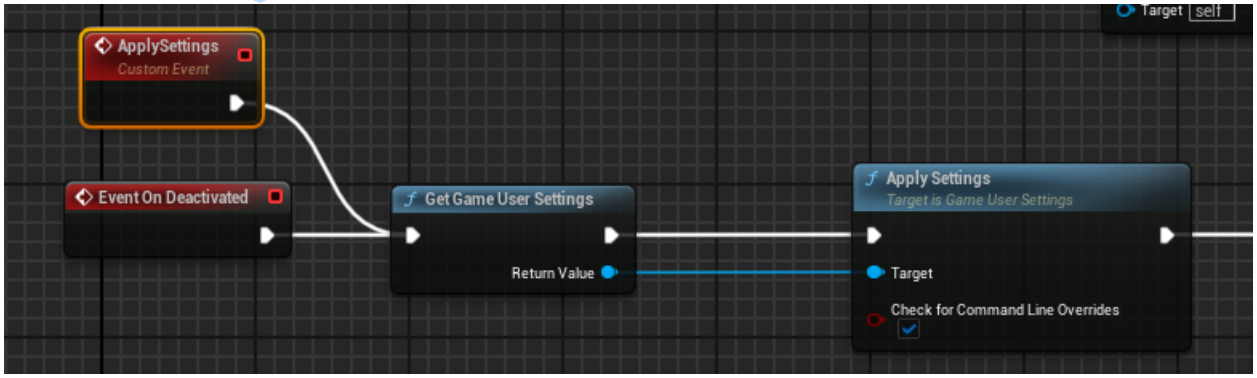
The view preset options will have had a defined value for the camera settings. Close, Normal and far.

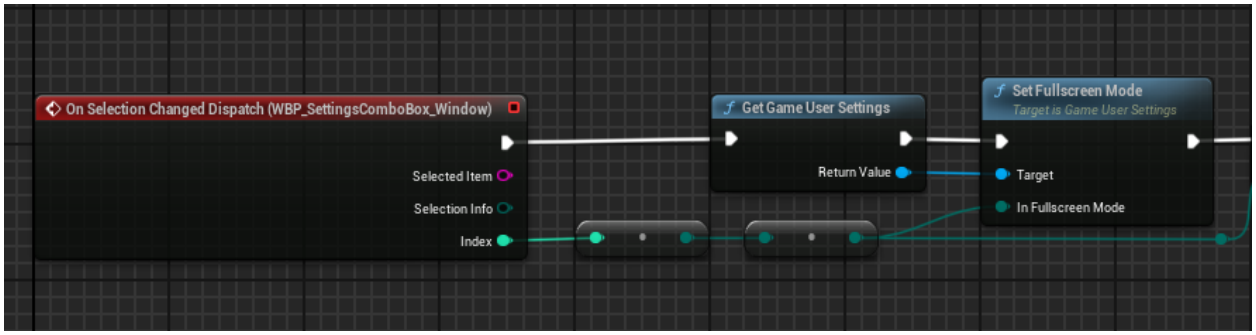
For the generic graph settings option. Using unreal engine's default game user settings to apply the setting is best standard and it will automatically save the settings to file/



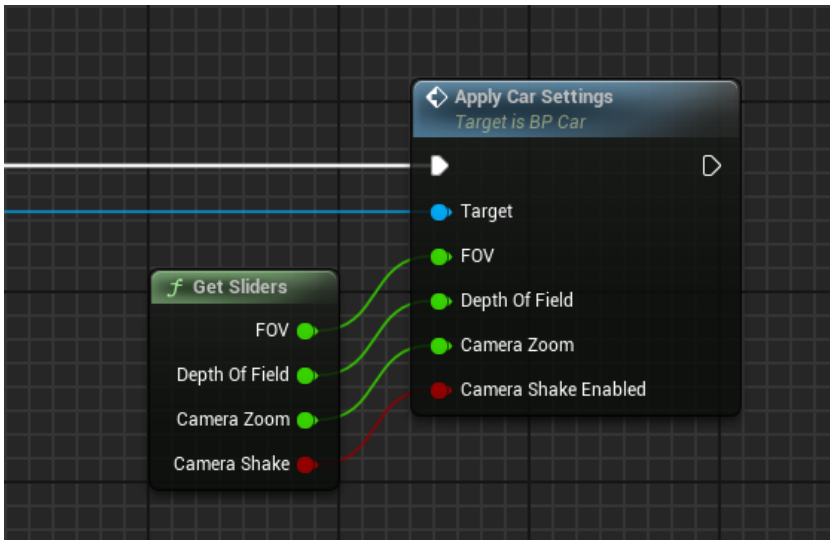








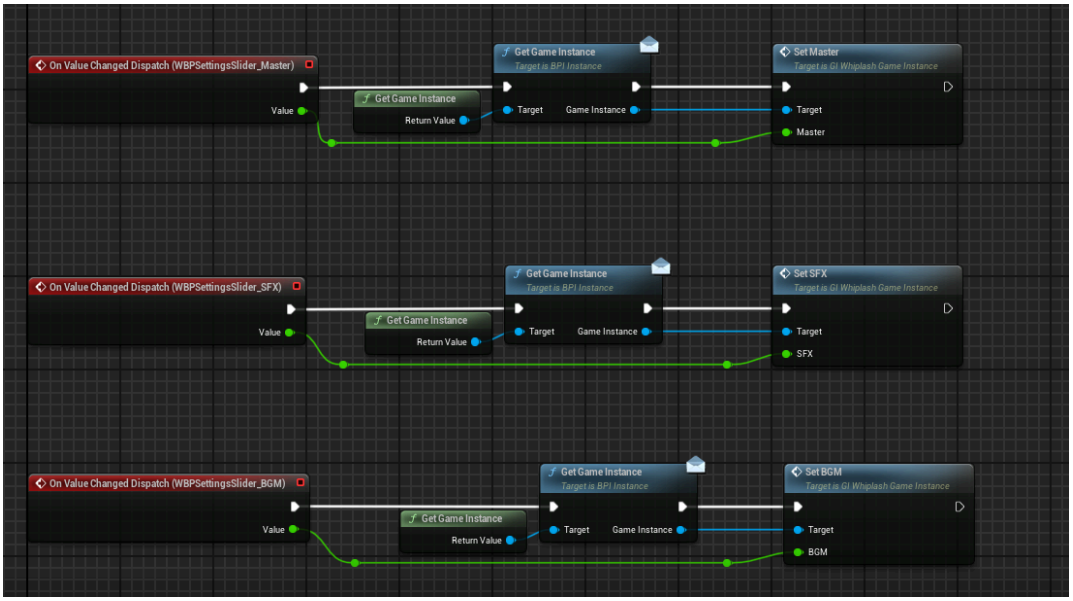
For the view settings, it will be applied to the player car as the settings effect the camera.

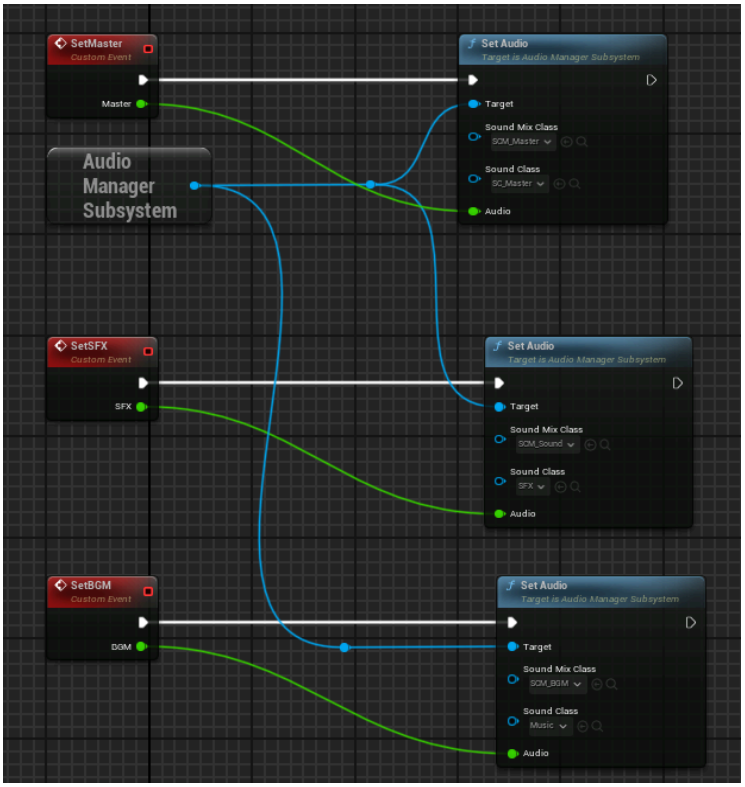


8.2.1.5 | AUDIO SETTINGS

The audio settings in the menu will be master, background music and sound effects.

Changing the sliders will send a signal to the game instance to apply the new volume to the sound class





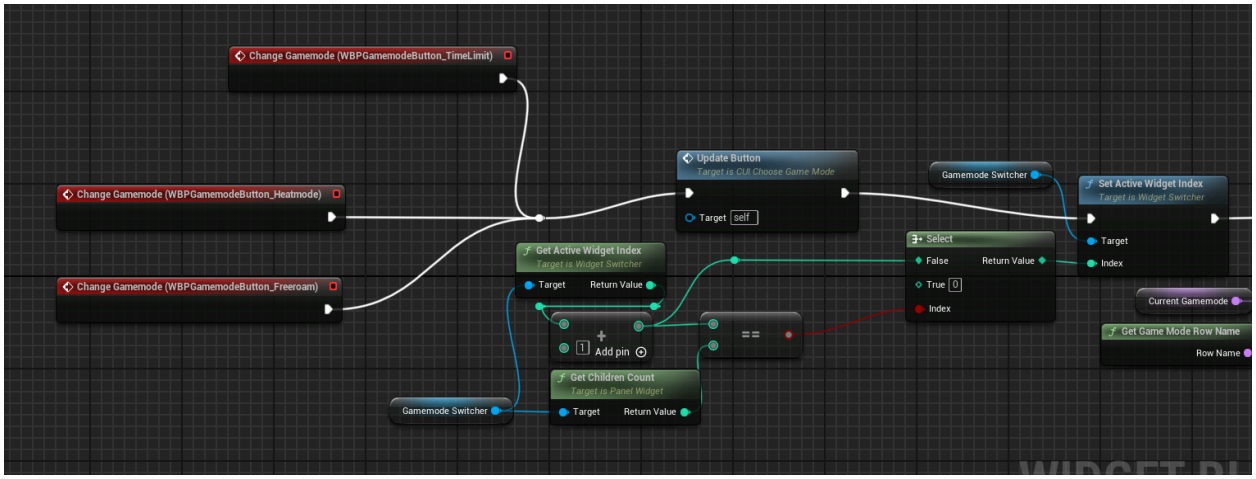
8.2.2 | CHOOSE GAMEMODE INTERACTION

The choose gamemode widget has three primary interactions. One is to display the medals earnt for the gamemode and duration that is showing.

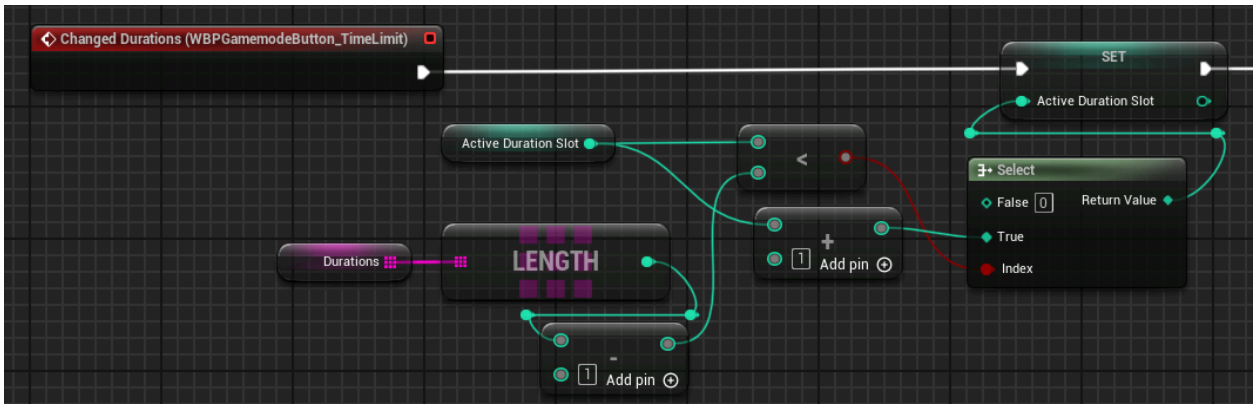
Two is to display the gamemode and durations.

And three is to select between normal or hard mode if unlocked.

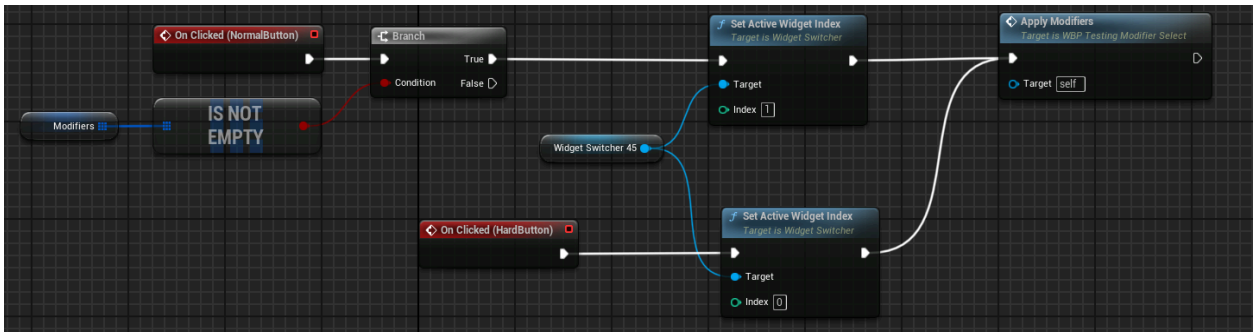
When clicking on the gamemode, it cycles to the next available gamemode.



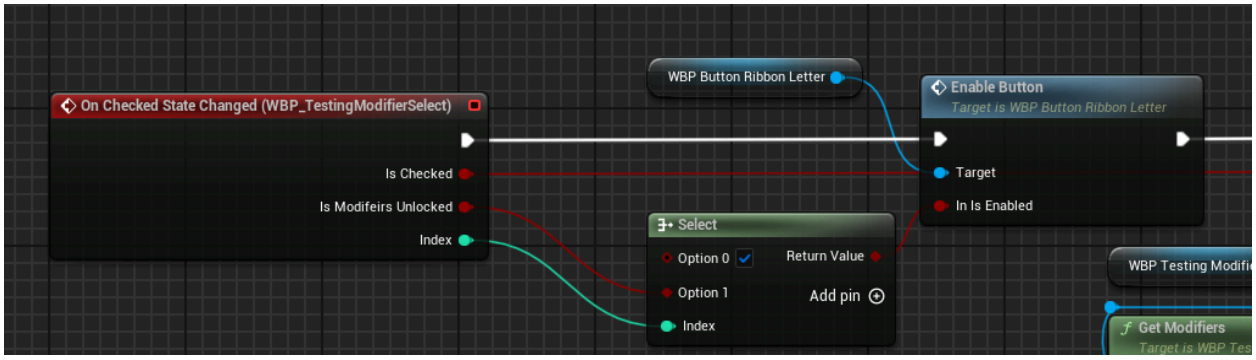
And when clicking on the duration button when available. It will cycle to the next available duration



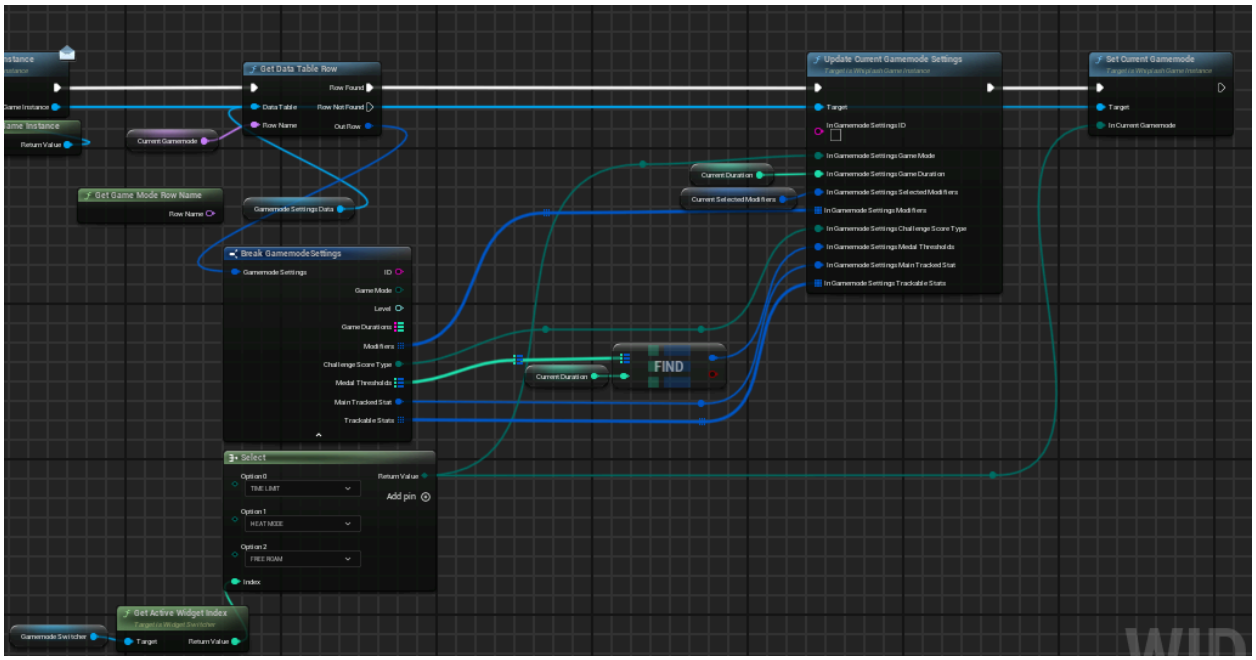
When clicking on the normal button, it will cycle to the hard mode button. And vice versa.



Depending what is shown on the widget. The press play button will be disabled or enabled.



Pressing start will get the gamemode shown, duration shown and difficulty shown and send it to the gamemode settings.

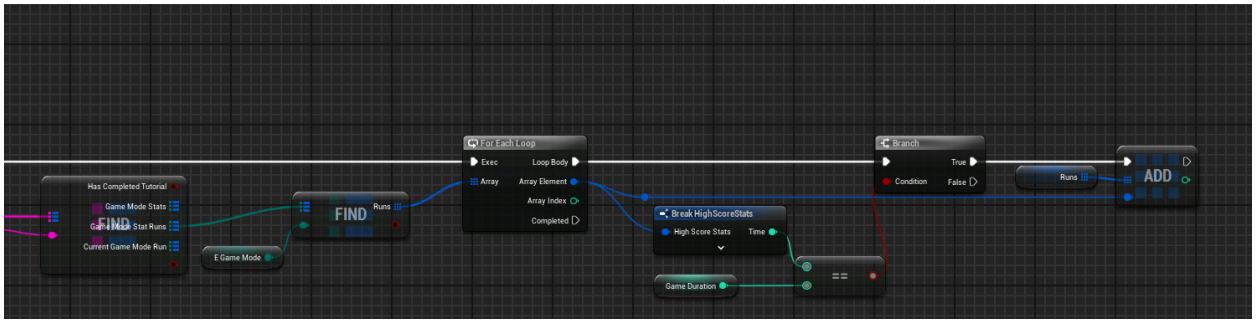


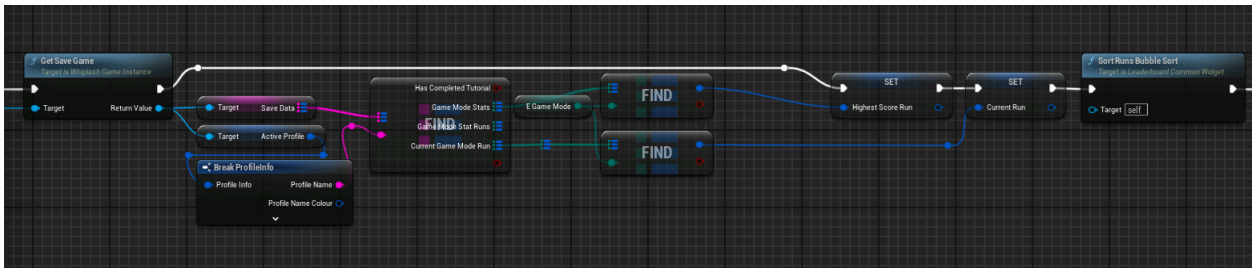
8.2.3 | LEADERBOARD INTERACTION

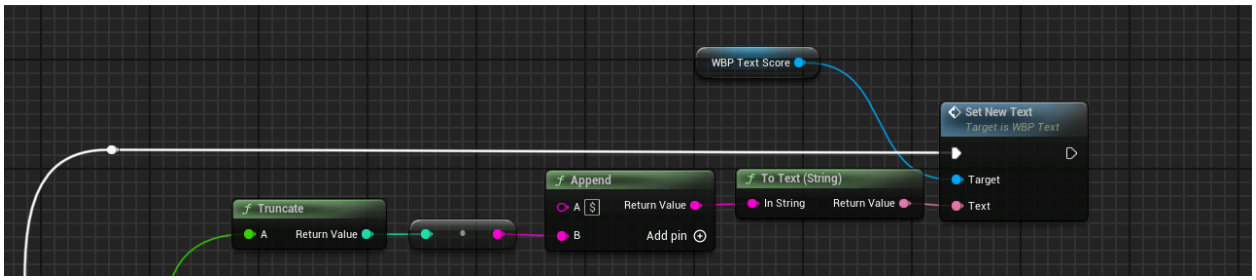
The leaderboard will display the stats of the current gamemode and run...Gets the current profile selected and added to the leaderboard ladder.

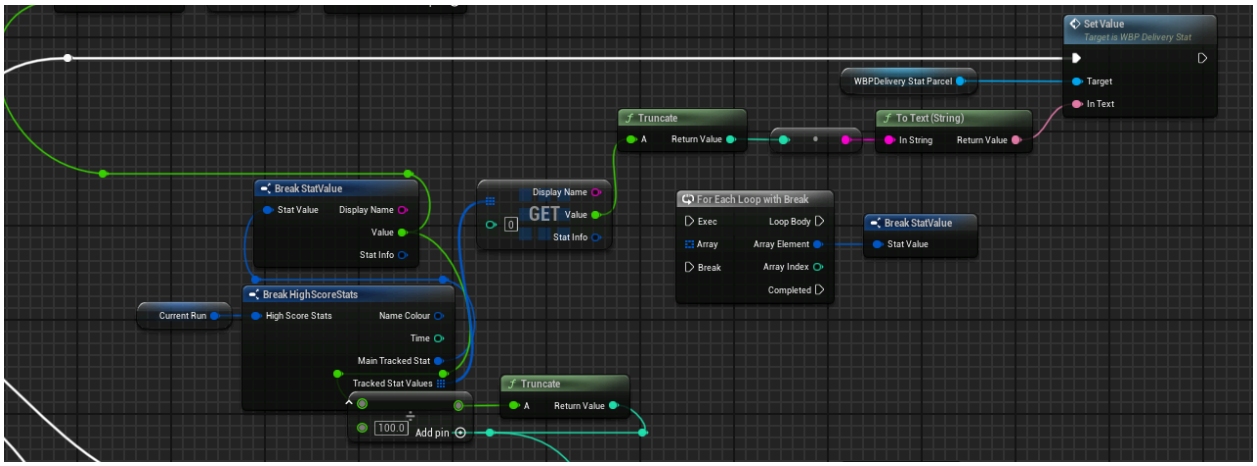
The way it gets its stats is from the save data in the save game class. Each Sava data lists the current run, highest run and multiple runs with different profiles.

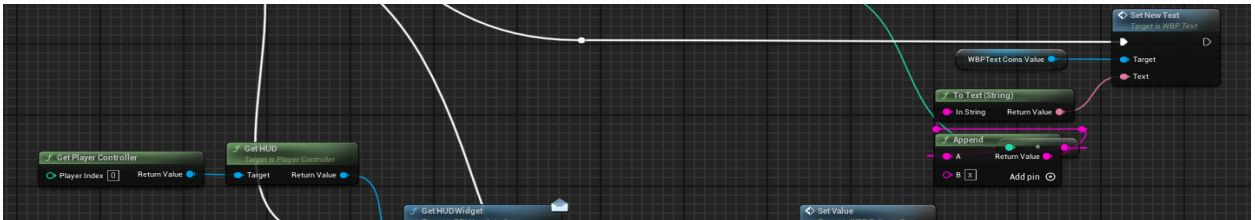
Each run data is a map. That the key is the gamemode.



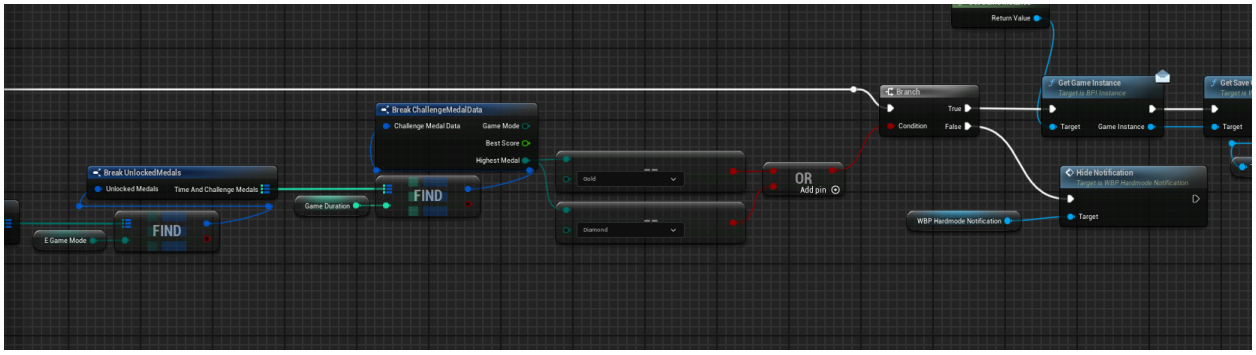


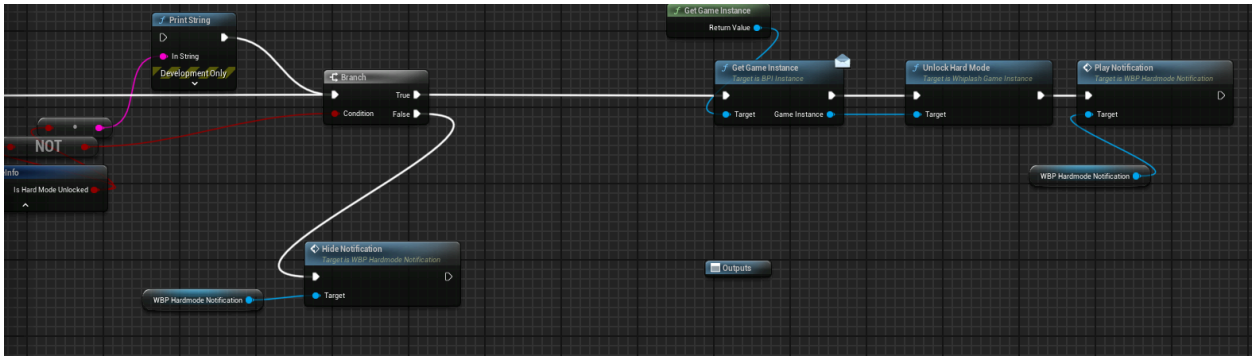






If the player has scored gold. The unlocked hard mode widget will show for a few seconds.

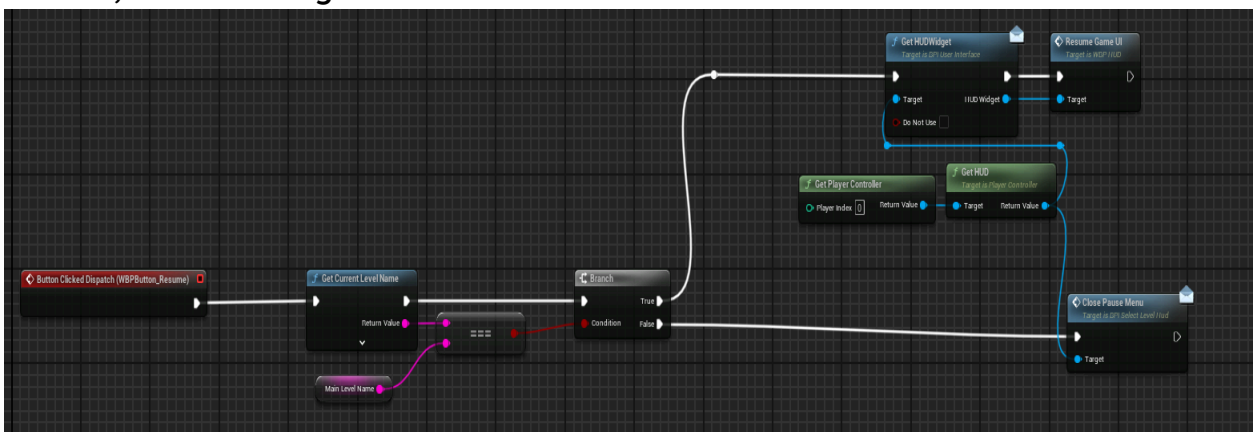


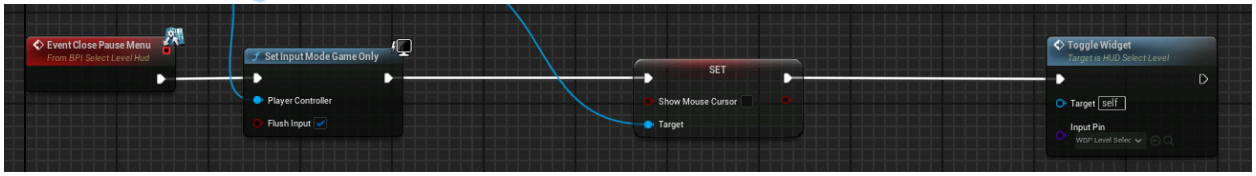


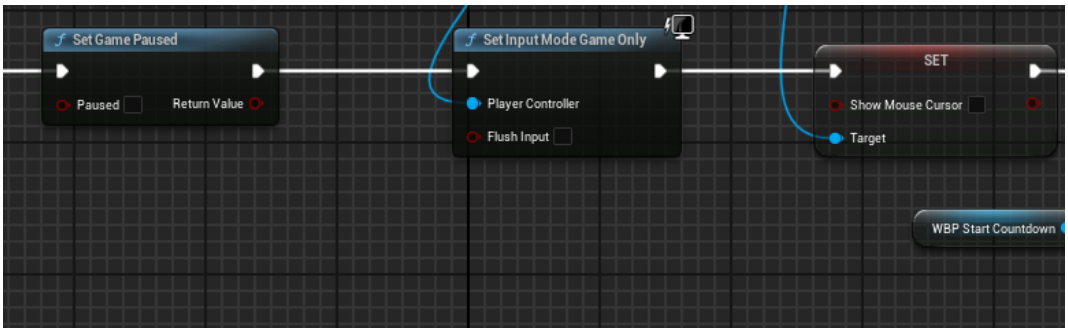
8.2.4 | PAUSE MENU INTERACTION

The pause menu has for buttons (Resume, Settings, quit to select level and quit to main menu)

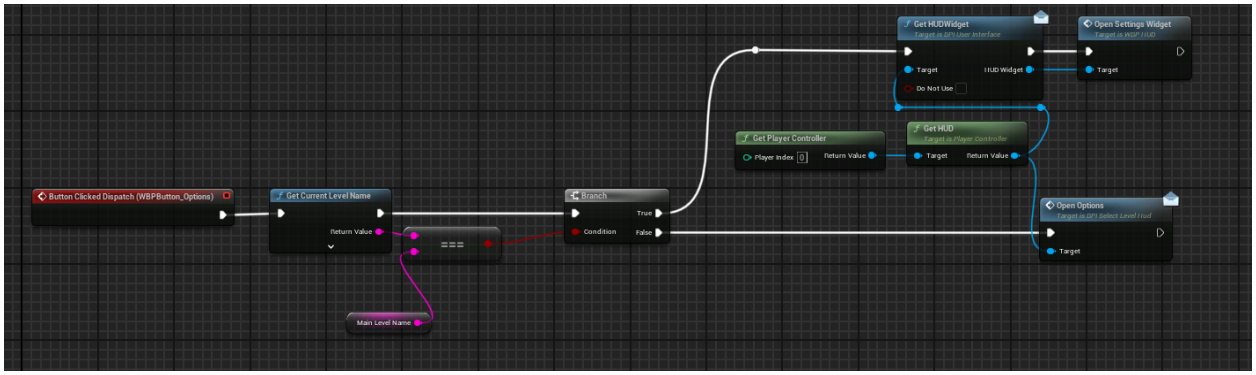
Resume, resumes the game when clicked. Or closes the menu when in the select level

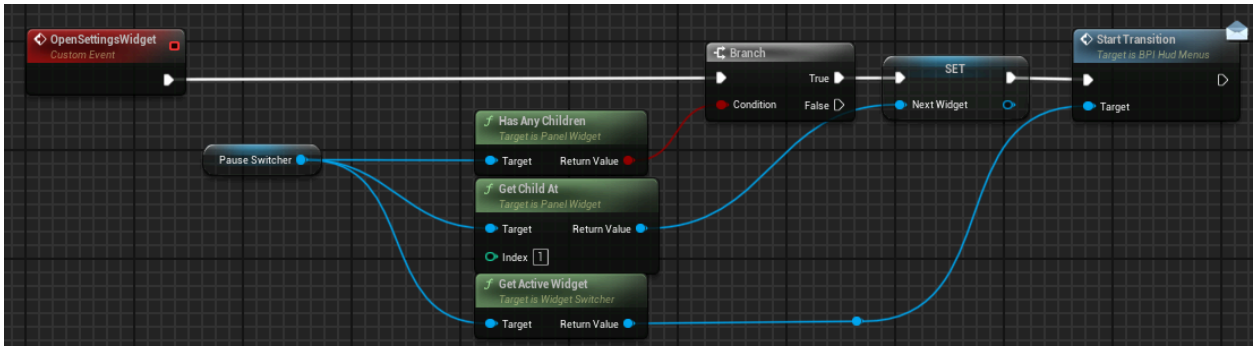




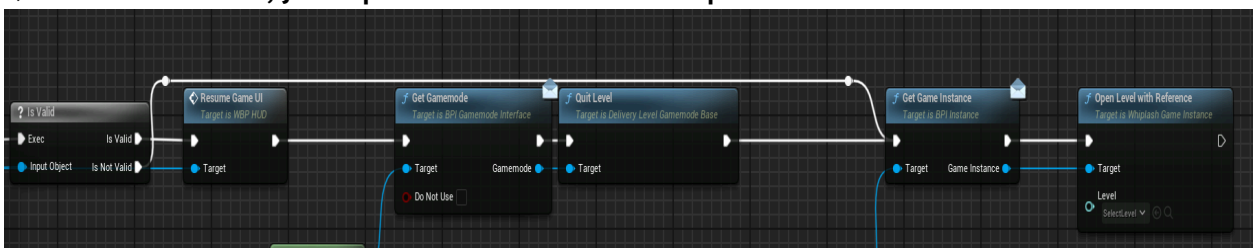


Settings, just opens the settings menu, like how in the main menu

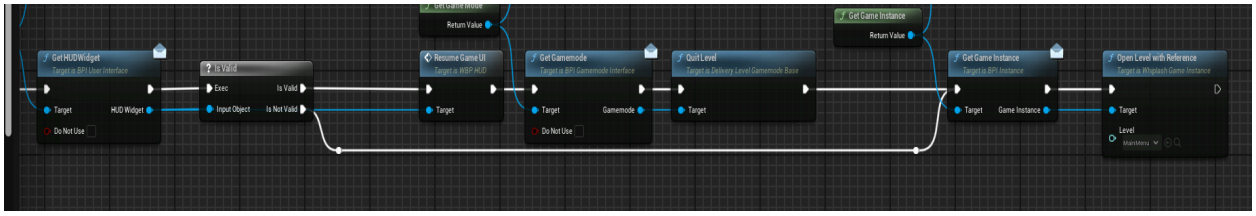




Quit to select level, just opens the select level map



Quit to main menu opens the main menu map



8.2.5 | LOADING SCREEN INTERACTION

In the game instance. There are two events that handle the loading screen. “PreLoadMap” which handles what happens before the map gets opened and “PostLoadMapWith”. Which handles the aftermath of the map loading.

During PreLoadMap. The loading screen will show on screen.

–GameInstance–

FUNCTION PreLoadMap
    ShowLoadingScreenTransitionIn()

FUNCTION END

During PostLoadMapWith. The loading screen does its transition out animation and the widget teardown.

–GameInstance–

FUNCTION PostLoadMapWith
    ShowLoadingScreenTransitionOut()

FUNCTION END

8.2.6 | UNLOCKABLE SHOP

With the unlockable shop. There will be six categories and the tab changing will work the same way as the tab list in the settings menu

Each item will have these values:

–Shop Item–

ENUM UnlockableCategory
CarPaint,
CarDetails,
Hats,
Windshield,
Front Bumper
END ENUM

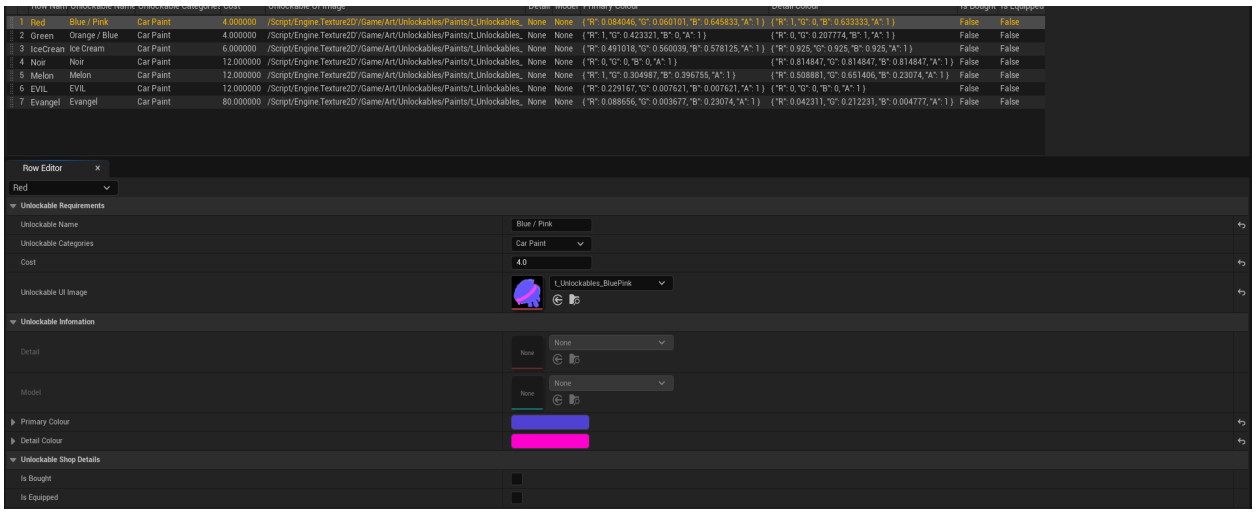
STRUCT ItemInformation

UnlockableName
UnlockableCategory
Cost
UnlockableUIImage

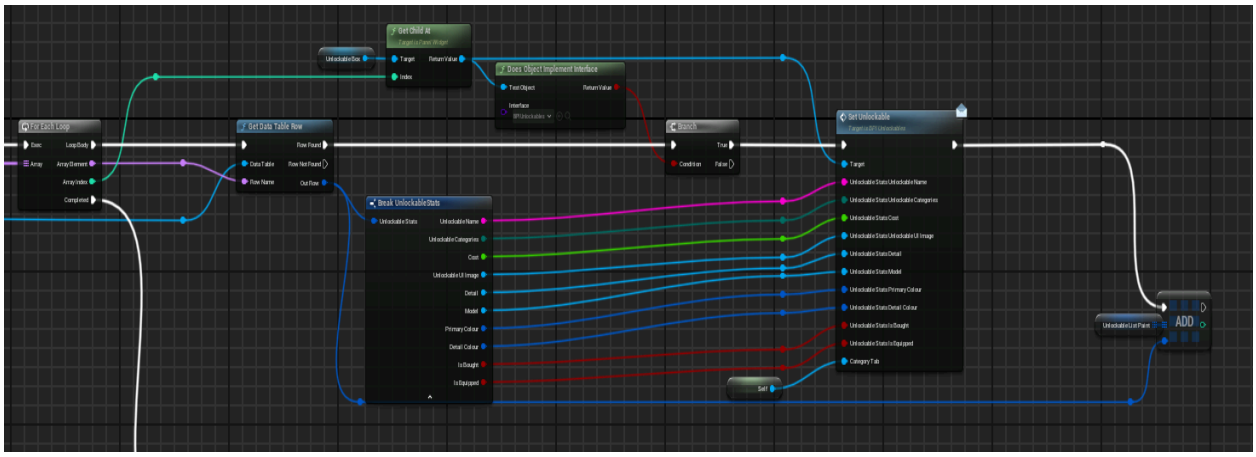
Detail
Model
PrimaryColour
DetailColour

IsBought
IsEquipped

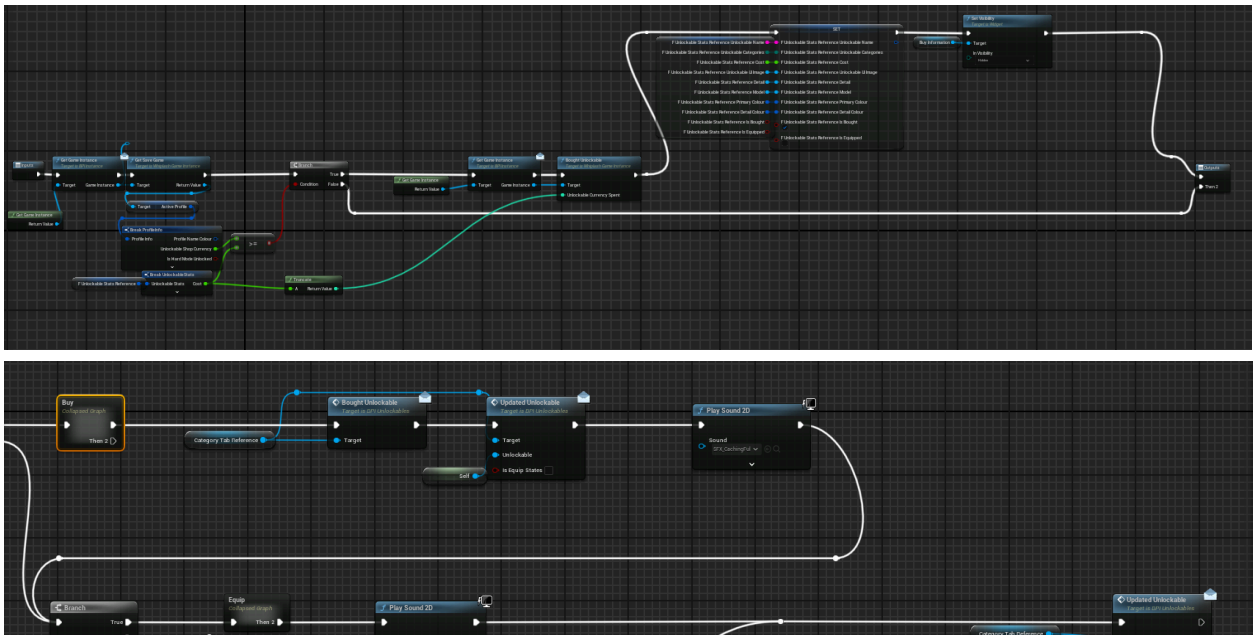
Each category will have a datatable of these values to store each item for each category.



For each category widget. It will get the information from its datatable category and fill out the contents of its grid, making it a dynamic system for designers to easily add items to the game.

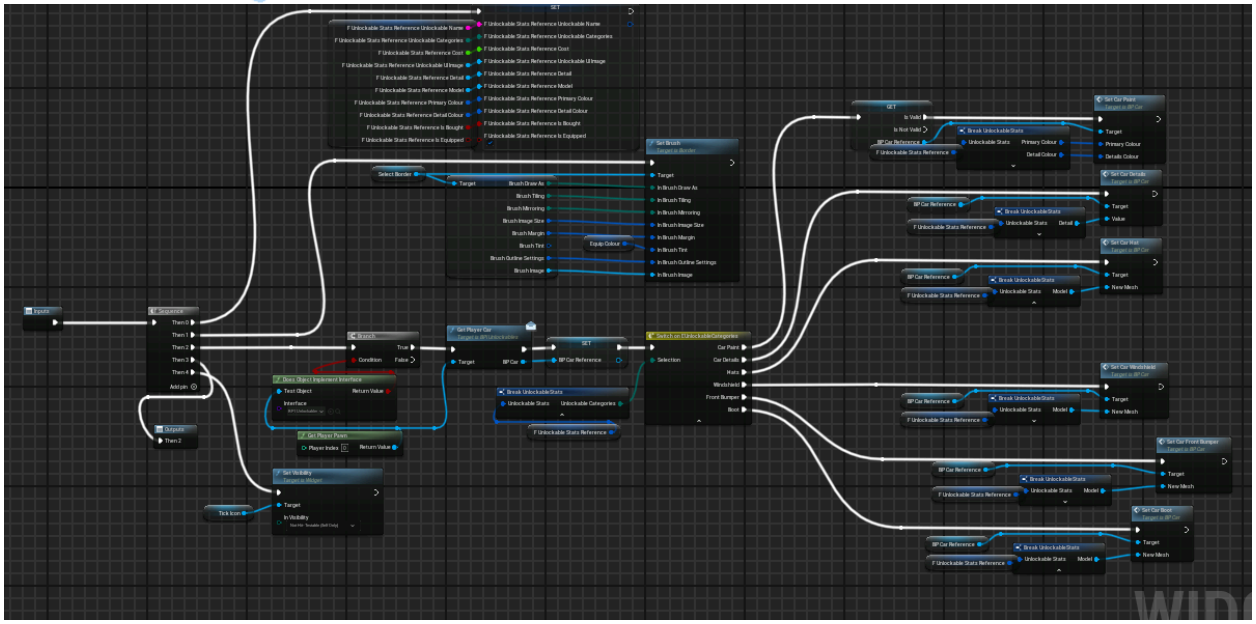


When the player buys an item, it will get the current currency the player has and calculate it between the cost and currency to see if the player can buy the item. If the player can buy the item. It will auto equip the item.

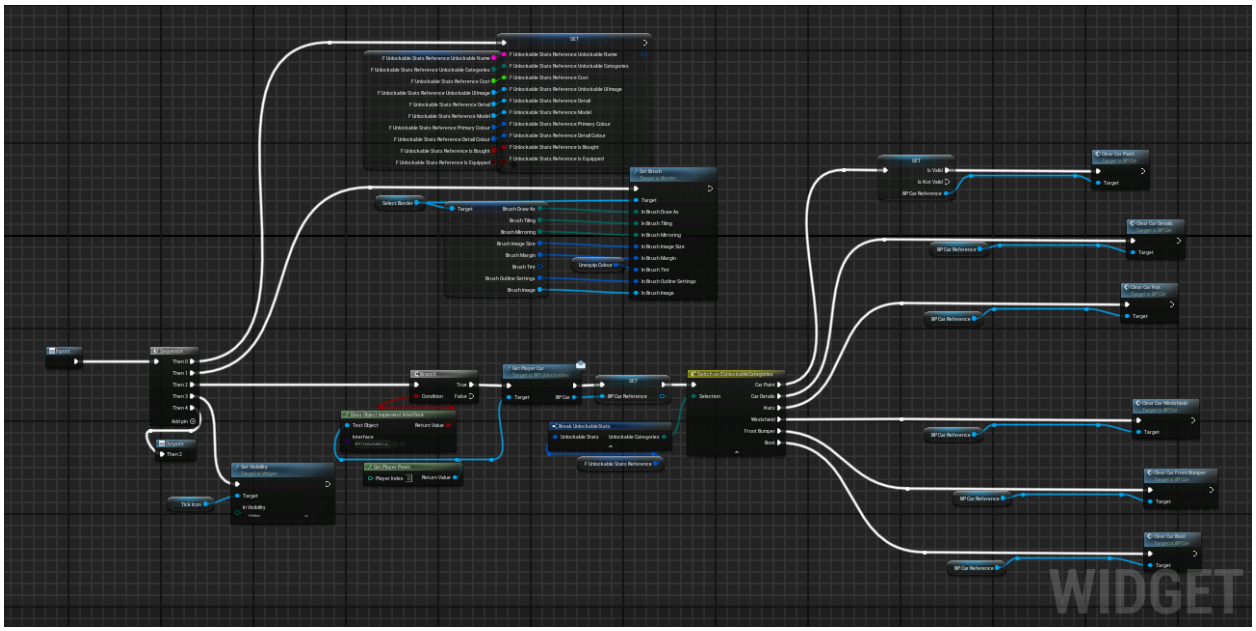


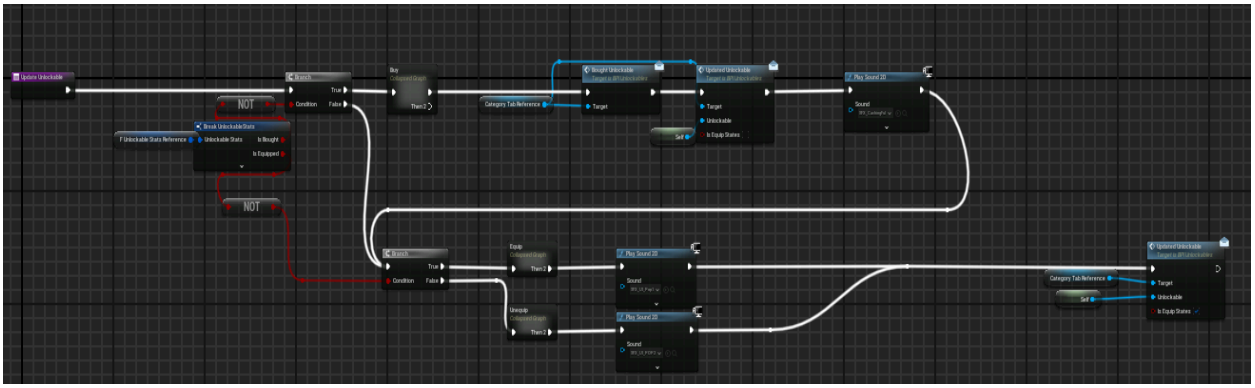
If the player has already bought the item. The item will have two states. Equipped and unequipped.

If in the unequipped state, the player when clicking on the item will equip the item.



If the item is in the equipped state, then click on the item will unequip it from the player





9.0 | AUDIO

CREDITS: https://freesfx.co.uk/Default.aspx

Name	Link(s)	Purpose/Uses
Tire Drift Skid	1	Loops while the player is drifting.
Car Driving	1 2	Loops while the player is driving. Variation 1 is also used for the plane engine idling in level select.
Max Drift Boost	1	Played when the maximum drift boost is reached.
Drift Boost Activated	1	Played when drift boost is activated.
Car Honks	1 2 3 4	Played when running into AI vehicles.
Pickup Location	1	Played when picking up a delivery.
Tree Thud	1	Played when running into any physics object without an overridden sound.
3, 2, 1 Countdown	1	Played at the start of the main game.
Game Over Whistle	1	Played at the end of the main game.
Nature Ambience	1	Loops in the background of the main game.
Waterfall Ambience	1	Loops near waterfalls.
Beach Ambience	1	Loops in the background of the beach area.
Electric Spark	1	Played on collision with a street light.
UFO	1 2	Variant 1 plays when the alien is lifting the player and variant 2 loops while the alien is moving.
Water Splash	1 2 3	Played on collision with bodies of water (other than the ocean).

Car Crash	1 2 3	Played on collision between player and AI vehicles.
Leaves Rustling	1	Played on collision with trees.
Vegetable Rustling	1	Played on collision with cauliflower, broccoli, and sunflowers.
Popcorn Pop	1	Played on collision with corn.
Carrot Pop	1 2 3	Played on collision with carrots.
Chicken	1	Played on collision with a chicken.
Deer	1	Played on collision with a deer.
Horse	1	Played on collision with a horse.
Penguin	1	Played on collision with a penguin.
Bird	1	Loops near birds.
Dog	1 2 3	Played on collision with a dog.
Seagull	1 2 3 4	Played on collision with a seagull.
Duck	1 2	Played on collision with a duck.
Campfire	1	Loops on the campfire.
Chain Fence	1	Played on collision with a chain fence.
Plastic	1	Played on collision with plastic materials.
Fan	1 2	Loops near fan objects.
Magnet	1	Loops near magnet objects.
Explosion	1	Played on collision with mines and magnets.

	2 3	
Bomb Fuse	1	Loops while bombs are inactive (during hard mode bomb rain).
Tire/Rubber	1	Played on collision with tires/rubber.
Car Falling Impact - Road	1 2	Played when the player falls onto the road.
Car Falling Impact - Grass	1 2 3	Played when the player falls onto grass.
Car Falling Impact - Sand	1 2	Played when the player falls onto sand.
Car Driving Terrain - Road	1	Loops while the player drives on the road.
Car Driving Terrain - Grass	1	Loops while the player drives on grass.
Car Driving Terrain - Sand	1	Loops while the player drives on sand. **Sound Name:** Pour Sand into Metal Bucket
Car In Air Wind	1 2	Played while the player is in the air.

10.0 | ASSET LIST

Asset list is held in a different document that will be linked:

https://docs.google.com/spreadsheets/d/1LG_t05Hq0rxWyyVr2AjaaP1O4Aw1YsdunT36vzVGleU/edit?gid=0#gid=0

11.0 | Technical Risks

11.1 | Not Practical UI Design

User Interface design and functionality plays an important role when it comes to performance for a game. Not having the User Interface consistent and clean will lead to bad draw calls and cost performance…Sticking to Unreal Engine's best practices for the layout of the hud and functionality will mitigates the risk

11.2 | Too Many AI Vehicles on the Map

With the game having AI vehicles that drive around the map. Too many loaded at once can cost performance loss and stress the player's computer…Having an object pool in place for the AI Vehicles mitigates the risk

11.3 | Levels Having Too Many Assets

With different levels and gamemodes being in the game. There is a risk that the amount of assets loaded in a level could decrease the performance on lower end PCs… To mitigate this. Multiple techniques that are associated with asset loading and gpu instancing will be used.

13.0 | TDD FEEDBACK

Version	Feedback	Date
v1.0	**JOSH:** **- 3.1. | Movement/Vehicle Physics:** Add more technical detail, equations, psuedocode, etc. Also screenshot the blueprint components being talked about. For example look at the packages section and apply the same layout to movement/vehicle physics. **- 4.0. | Graphics:** Add graphics settings, texture sizes, and any other graphics related information. **- 6.0. | Physics:** Bit more in depth about the actual physics settings being used. **- 7.0. | Maps & Gamemodes:** Screenshots, how the user transitions between them.	5/12/25