ALPHA\_RCRacing / RCRacingPawn

Architecture/Design Document

**Table of Contents**

1 Introduction.. 3

2 Design Goals.. 3

3 System Behavior.. 3

4 Logical View... 3

4.1 High-Level Design (Architecture of the Entire system)… 4

4.2 Mid-Level Design of the RCRacingPawn Module... 5

4.3 Detailed Class Design of the RCRacingPawn Module… 6

5 Process View of the RCRacingPawn Module... 7

5.1 Process View of the ABowlingBall\_PowerUp Submodule… 9

5.2 Process View of the AFirework\_PowerUp Submodule… 12

5.3 Process View of the AFreeze\_PowerUp Submodule… 15

5.3 Process View of the ATrap\_PowerUp Submodule… 16

6 Use Case View... 17

Change History

**Version:** 0.1

**Modifier:** Antoine Plouffe

**Date:** 03 / 14 / 2022

**Description of Change:** Module Design Document started.

**Version:** 0.2

**Modifier:** Antoine Plouffe

**Date:** 03 / 16 / 2022

**Description of Change:** Logical and Process View updated.

**Version:** 0.3

**Modifier:** Antoine Plouffe

**Date:** 03 / 18 / 2022

**Description of Change:**

1. **Introduction**

This document describes the architecture and design for ALPHA\_RCRacing, a game being developed by Inertial Sketch. ALPHA\_RCRacing is a Multiplayer RC Car Game where player compete against each other in a competitive racing track filled with jumps, turns and Power Ups.

The purpose of this document is to describe the architecture and design of the RCRacingPawn Module application in a way that addresses the interests and concerns of all major stakeholders. For this application the major stakeholders are:

* Developers;
* Project Manager.

1. **Design Goals**

The design priorities for the RCRacingPawn system are:

* The design should be dynamic enough to interact with different kind of power ups;
* The design should feel natural to the player;
* The design should allow the Designers to modify key variables within the Engine;
* The design should let the player reset the car if it is stuck.

1. **System Behavior**

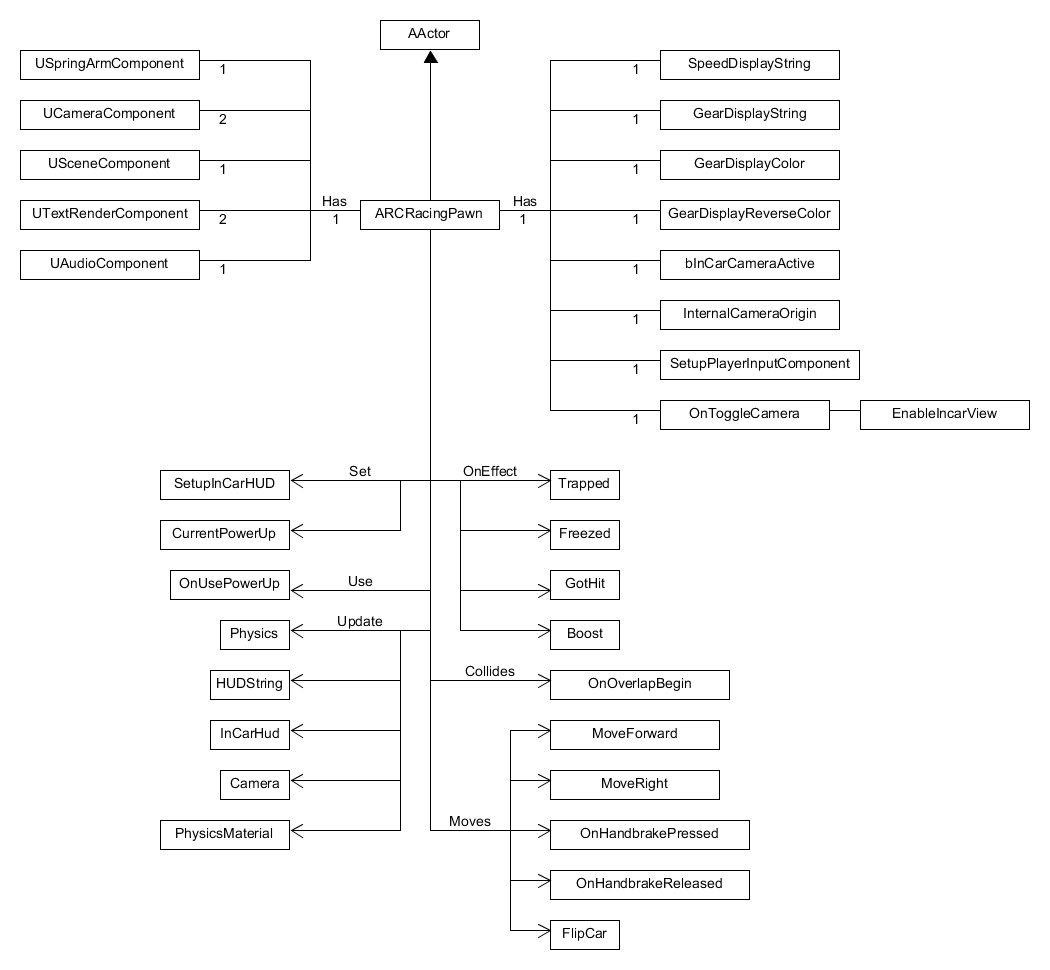
The RCRacingPawn module is built from a single RCRacingPawn and is linked to multiple class system like the UI/Menu, PowerUp, Boost and RCRacingWheels. This configuration will allow a centralization of the different system to create a cohesive game.

1. **Logical View**

The logical view describes the main shared components, attributes and switches of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed from a macro perspective and progressively goes to a micro perspective to view the detailed sequences and components.

* 1. **High-Level Design (Architecture of the Entire system)**
* RacingPawn System is the main system and handles the car movements, current power up and various effects.
* PowerUp System knows its mesh and collision sphere as well as its cooldown and despawn timer. Its children have unique effects.
* UI System is used to indicate crucial information like the player’s current power up.
* Menu System handles the selection of car, map and handles the race creation.
* InputComponents system handles…
  1. **Mid-Level Design of the RacingPawn Module**



* 1. **Detailed Class Design of the RacingPawn Module**

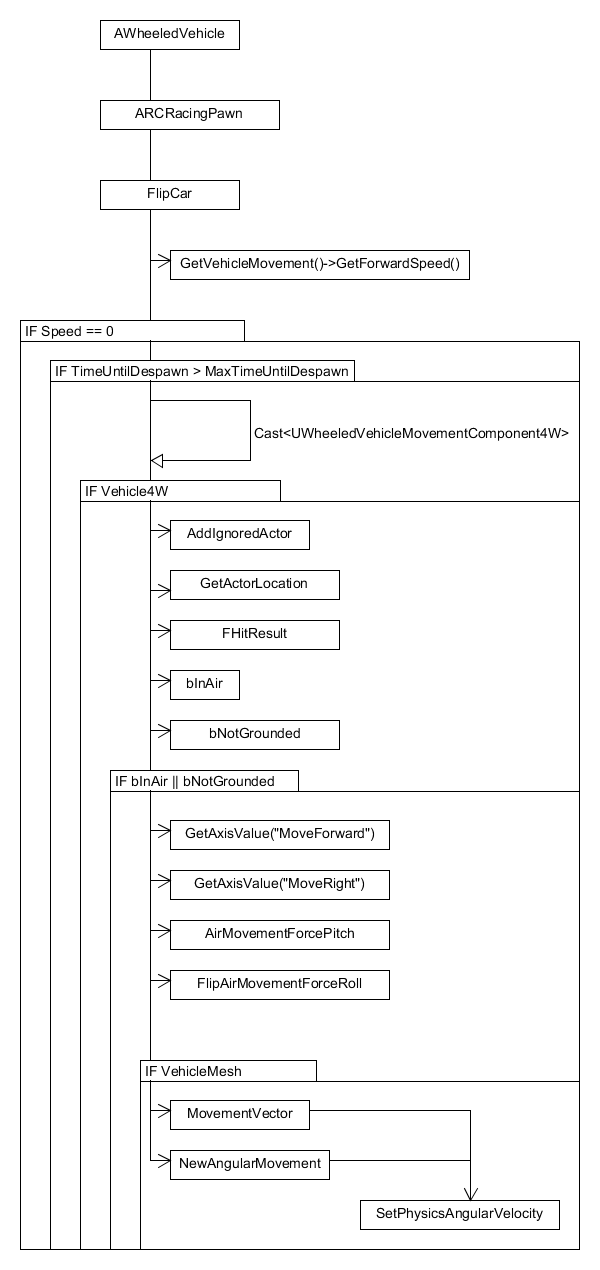
1. **Process View of the RacingPawn Module**

The operation related to the RacingPawn are dependant of factors like isPicked and isUsed. Some power ups are also dependant on their cooldown to activate. The power up isn’t active until it gets assigned when PowerUp OnOverlapBegin is triggered.

**ONOVERLAP**

**TICK**

**FLIPCAR**



The FlipCar sequence goal is to prevent the player getting their vehicle upside down and/or stuck. The function first looks to see if the car is moving and if the vehicle has a MovementComponent4W. Upon succeeding both checks, CollisionQueryParams will ignore this actor, registers this actor’s location and create two FVector:

1. TraceStart: where the LineTrace starts in the Z Axis;
2. TraceEnd: where the LineTrace ends in the Z Axis.

And two booleans:

1. bInAir uses LineTraceSingleByChannel to detect if the player is currently in the air;
2. bNotGrounded uses a DotProduct to detect if the player’s up vector is parallel to an UpVector. If the DotProduct result is under 0.1f, then the vehicle isn’t upside down.

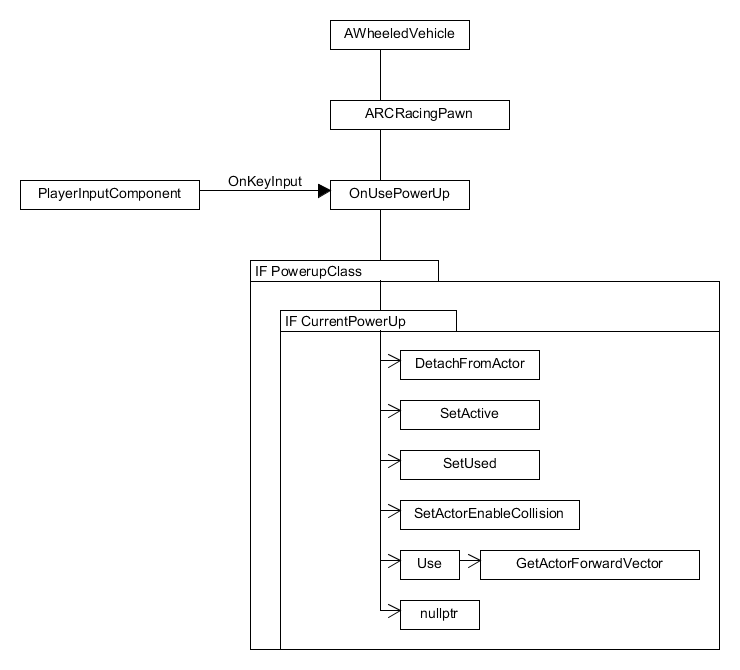
If any of the two Boolean return true, it will store two local variables with the axis values of the MoveForward’s input and MoveRight’s input (A value between 1 and -1). It will also store the pitch and roll force that we want to apply.

If a primitive component of type mesh is set on the vehicle, it will calculate a MovementVector by injecting:

1. RightInput and Roll Force;
2. ForwardInput and Pitch force.

Afterwards, it will calculate NewAngularMovement by getting the actor rotation and rotating it by MovementVector. Then the vehicle mesh physics linear velocity is set to NewAngularMovement to flip the car until it is grounded.

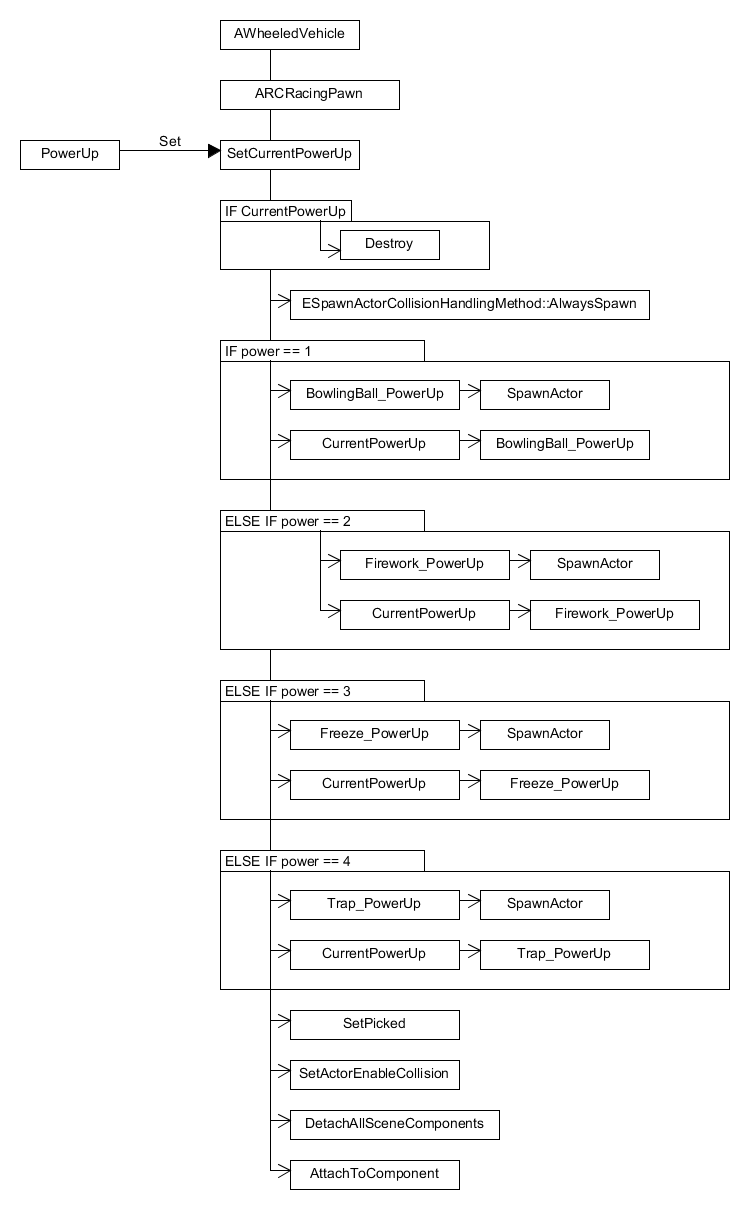
**ONUSEPOWERUP**

****

The OnUsePowerUp sequence is dependant of the player’s input (Space bar). Its main goal is to switch a few settings and is to call the virtual Use function of the PowerUp class. First, it looks to see if the car has a PowerUp class and if it has a CurrentPowerUp active. Secondly and upon succeeding both checks, a series a function calls will be made:

1. The CurrentPowerUp will detach itself from the vehicle;
2. The CurrentPowerUp will be set to active;
3. The CurrentPowerUp will be set to Used;
4. The CurrentPowerUp collision’s sphere will be enable;
5. The CurrentPowerUp call its function Use by giving it the actor forward vector;
6. The CurrentPowerUp will be set to nullptr.

**SETCURRENTPOWERUP**

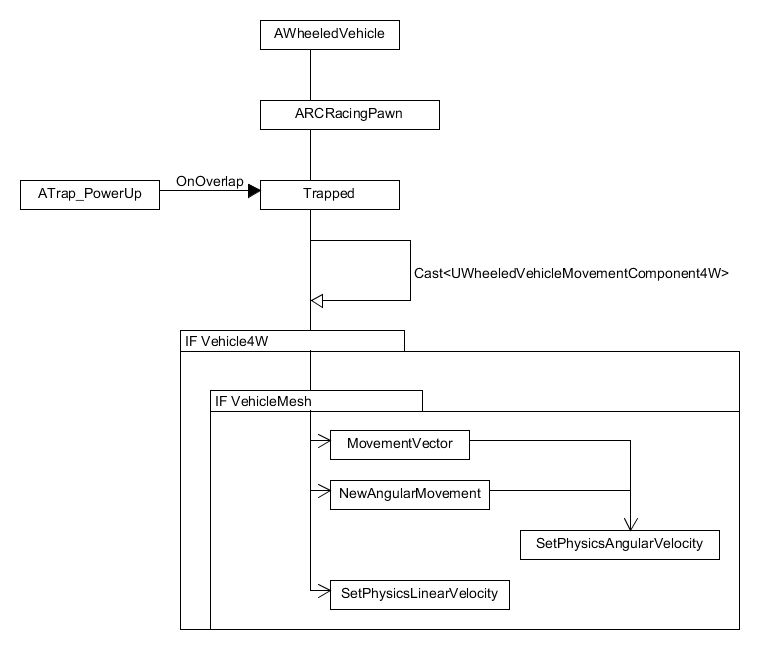


The SetCurrentPowerUp sequence is being called when the player’s overlaps with a power up who’s isPicked Boolean returns false. As the function names suggest, its main goal is to set the player’s current power up to the randomly chosen one inside the PowerUp class.

First, if CurrentPowerUp isn’t nullptr, CurrentPowerUp will destroys itself. Then depending on the result of the random number generation (1 to 4) done inside PowerUp, the function will spawn the corresponding PowerUp and set the CurrentPowerUp to that newly spawned PowerUp. Finally, a series a function calls will be made:

1. The CurrentPowerUp will be set to Picked;
2. The CurrentPowerUp collision’s sphere will be disable;
3. The CurrentPowerUp will detach itself from the vehicle;
4. The CurrentPowerUp will attach itself on the vehicle socket called BowlingBallSocket.

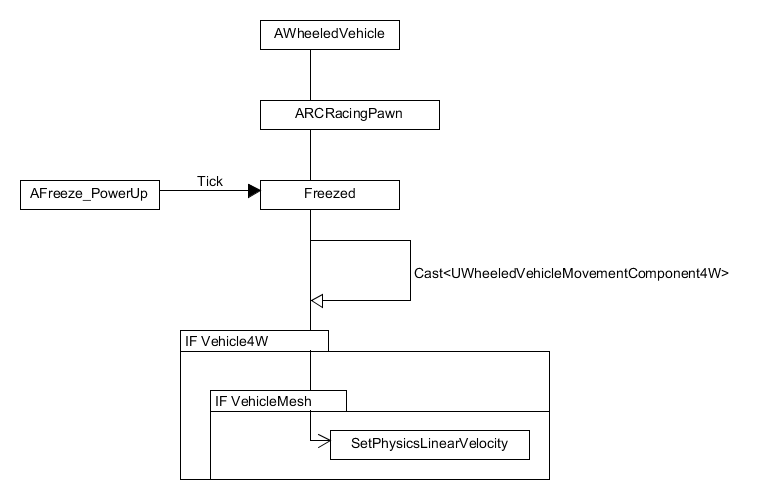
**TRAPPED**



The Trapped sequence starts when the player’s overlaps with the collision sphere of a ATrap\_PowerUp. The main goal if this function is to create a wow moment when the player overlaps a trap by moving upwards and rotating their vehicle mesh.

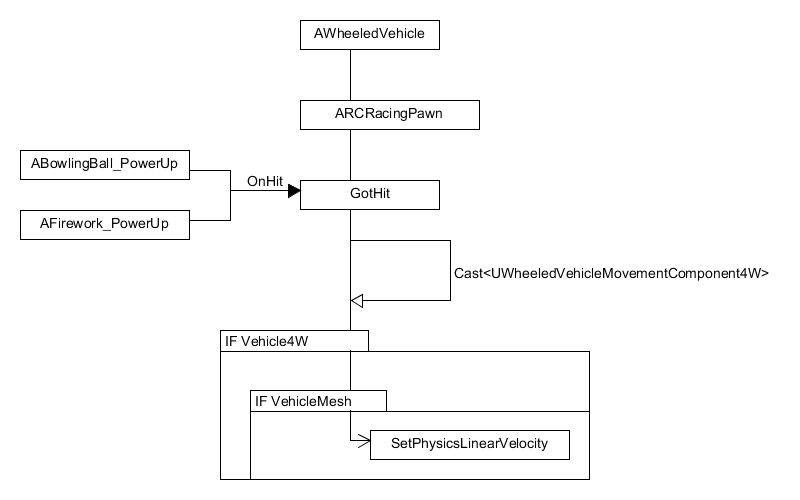
The function first looks to see if the vehicle has a MovementComponent4W and a primitive component of type mesh. Upon succeeding both checks, it will calculate a MovementVector by adding AirMovementForceRoll to the X Axis and NewAngularMovement by getting the actor rotation and rotating it by MovementVector. Then the vehicle mesh physics linear velocity is set to 1000.0f in the Z axis to simulate a jump and the vehicle physics angular velocity is set to that NewAngularMovement variable.

**FREEZED**



The Freezed sequence starts if the player’s overlaps with the blast radius of a AFreeze\_PowerUp. The main goal if this function is to apply the freeze effect on the vehicle mesh. The function first looks to see if the vehicle has a MovementComponent4W and a primitive component of type mesh. Upon succeeding both checks, it will reduce vehicle physical linear velocity to zero.

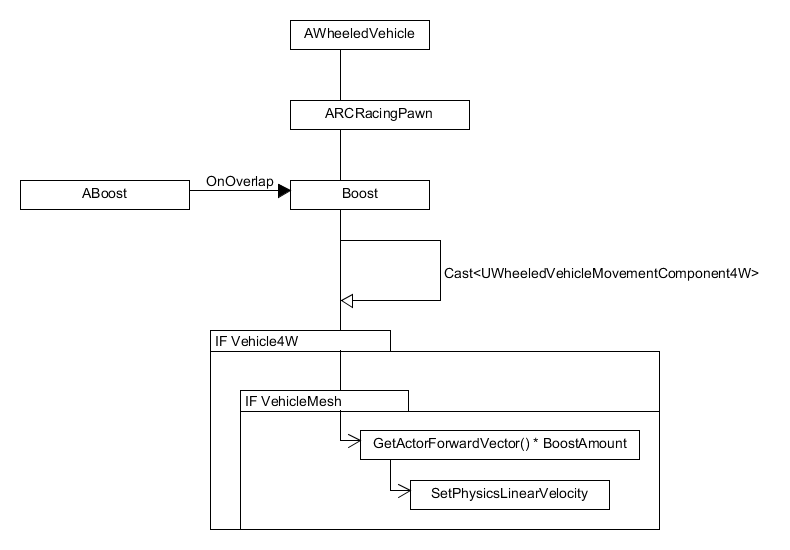
**GOTHIT**



The GotHit sequence starts when the player’s hit with the collision sphere of either the ABowlingBall\_PowerUp or the AFirework\_PowerUp. The main goal if this function is to temporarily paralyzed the player’s vehicle OnHit.

Upon succeeding both checks, it will set the vehicle mesh physics linear velocity on the Z axis to 500.0f and will briefly stop moving.

**BOOST**

****

The Boost sequence starts when the player’s overlaps with the collision sphere of ABoost. The main goal if this function is to temporarily boost the player’s vehicle speed.

Upon succeeding both checks, it will create a local variable of type FVector. Then, it will set its value to the ForwardVector of the actor times its BoostAmount. Finally, it will add to the vehicle mesh physics linear velocity this boost force.

1. **Use Case View**

For this release, all power ups indicators are displayed via debug messages. In normal gameplay, the type of power up will be indicated by an icon.

The PowerUp has multiple variables related to its design – all editable either within code or inside the Unreal Editor.

* isPicked: This switch indicates if the power up has been picked up. Default: false.
* isUsed: This switch indicates if the power up has been used. Default: false.
* m\_MaxCooldown: This timer indicates the maximum cooldown of the power up. Default: 2.0f.
* MaxTimeUntilDespawn: This timer indicates the maximum time before a projectile is despawn. This timer is used on both AFirework\_PowerUp and ABowlingBall\_PowerUp. Default: 5.0f.
* RandomPowerUp: This integer indicates the value associated with the latest given power up to the player. Default: nullptr;

The PowerUp is dragged into the scene where needed and will interact OnOverlap with a vehicle.

**ONOVERLAP**

When a power up overlaps with a vehicle, it will selfcheck if it was picked before. If not, it will continue and cast the OtherActor to ARCRacingPawn. If succeeded, the power up will generate a random FMath::RandRange between 1 and 4 corresponding to a power up and sets the player’s current power up to that random one and destroy itself.

1 = ABowlingBall\_PowerUp

2 = AFirework\_PowerUp

3 = AFreeze\_PowerUp

4 = ATrap\_PowerUp