ALPHA\_RCRacing / PowerUp

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 PowerUp Module... 5

4.3 Detailed Class Design of the PowerUp Module… 6

5 Process View of the PowerUp 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 / 15 / 2022

**Description of Change:** Sequences descriptions added.

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 PowerUp 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 Power Ups system are:

* The design should be dynamic enough to create different kind of power ups;
* The design should have a Parent/Child type structure;
* The design should allow the Designers to have access to key member variables within the Engine;
* The design should reward the player for using correctly and on time their power ups.

1. **System Behavior**

The PowerUp module is built from a single PowerUp that the following power ups inherit from:

* BowlingBall\_PowerUp;
* Firework\_PowerUp;
* Freeze\_PowerUp;
* Trap\_PowerUp.

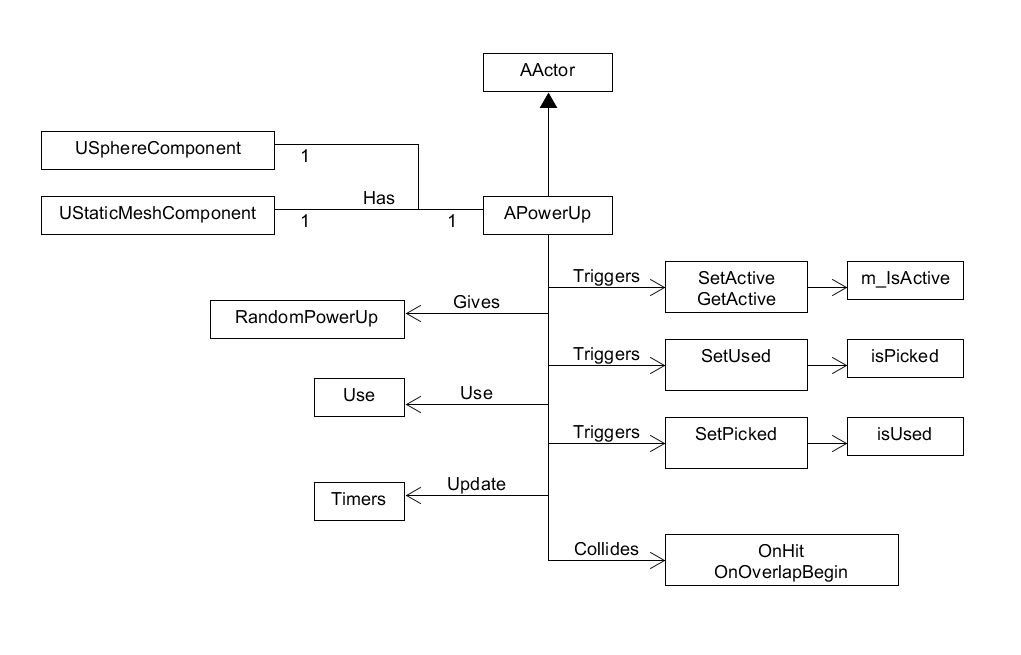
This configuration will allow all power ups to gain access to their shared attributes and methods like the USphereComponent, the UStaticMeshComponent, the cooldown and despawn timer, the OnHit and OnOverlapBegin functions, and finally the Use function.

1. **Logical View**

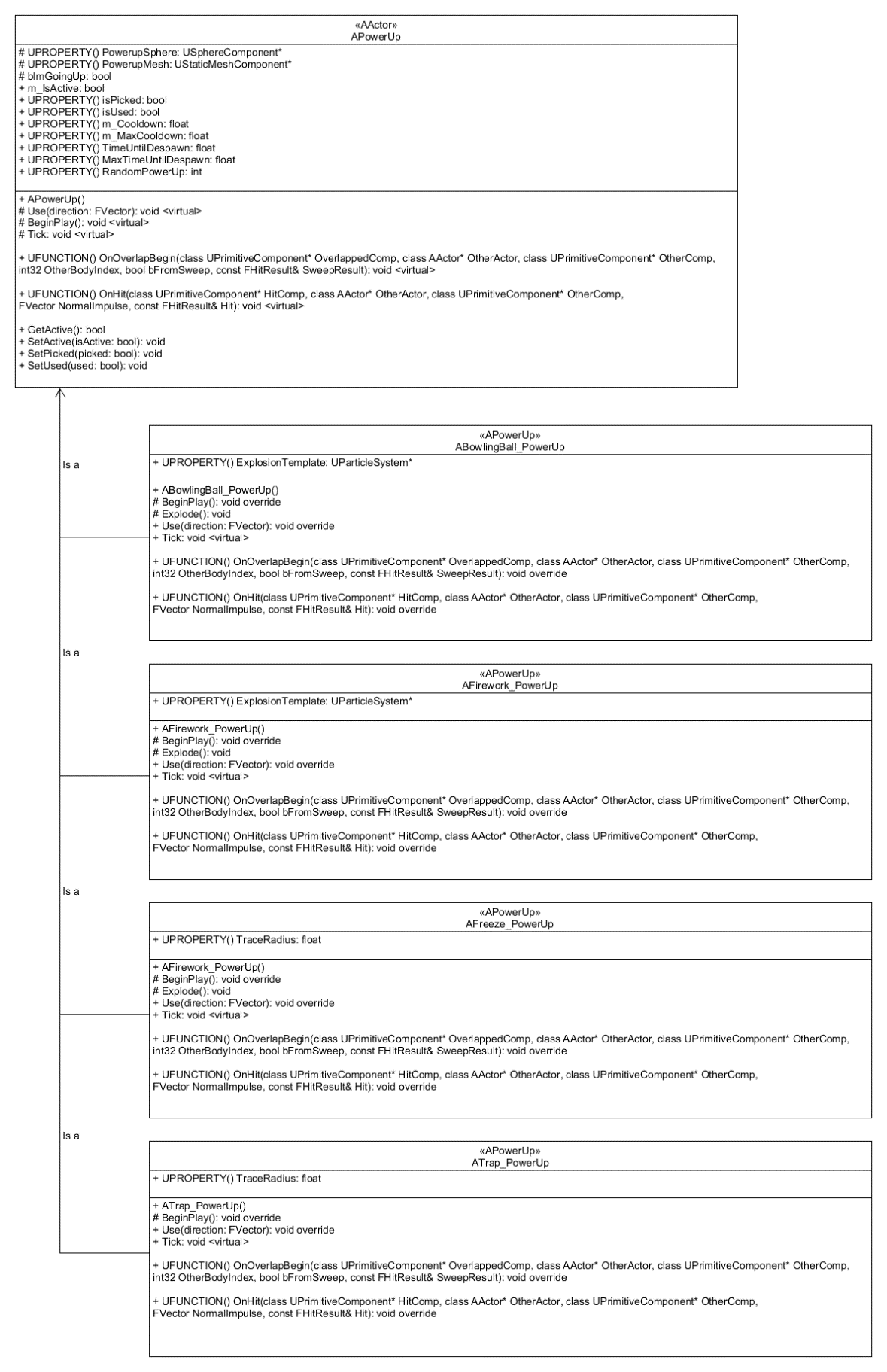
The logical view describes the main shared components, attributes and switches of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

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.
  1. **Mid-Level Design of the PowerUp Module**



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

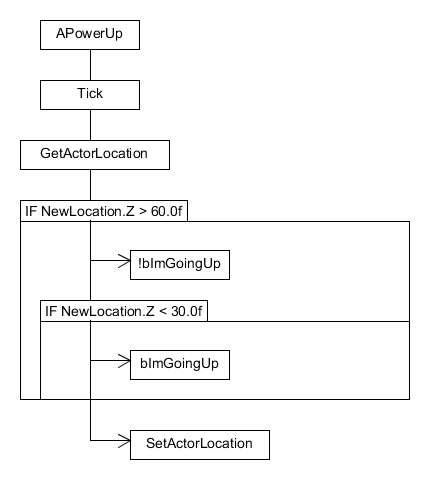


1. **Process View of the PowerUp Module**

The operation related to the PowerUp 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.

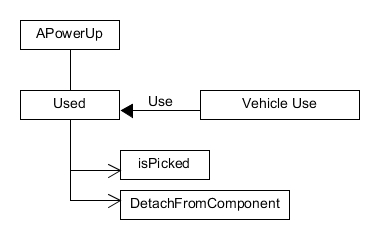
In our current release, PowerUps affects all vehicles, including the initiator for testing purposes.

**TICK**



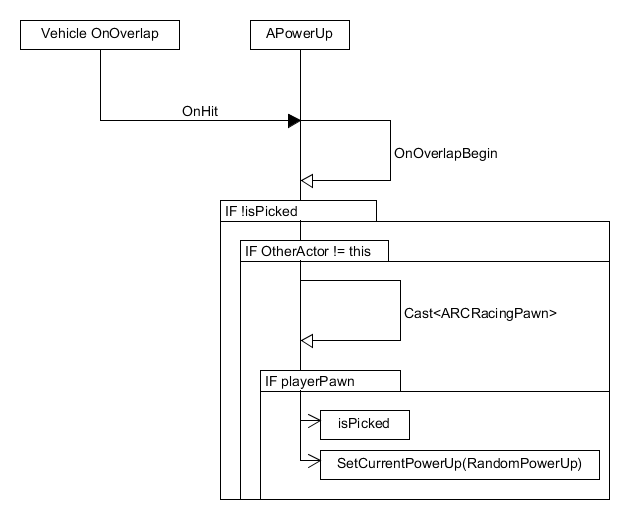
The tick sequence starts on Play. It takes the location of the action and moves it on the Z axis up to a certain point. Doing above (60.0f) or under (30.0f) the threshold will turn the switch accordingly. This is to create a dynamic and moving actor.

**USE**

****

The Use sequence starts when the player presses the use key space bar which calls the PowerUp’s Use function. Doing so will set the isPicked switch off and will detach the actor from the vehicle mesh.

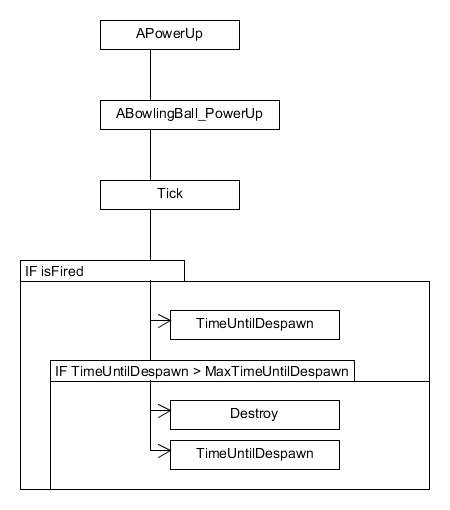
**ONOVERLAP**

****

The OnOverlap sequence starts when the player overlaps the power up collision sphere. When it begins, it first check if it has already been picked up by another player. If not, it will safe check if the OtherActor is, in fact, a ARCRacingPawn. If so, it will switch isPicked accordingly and set the player’s current power up to a random one.

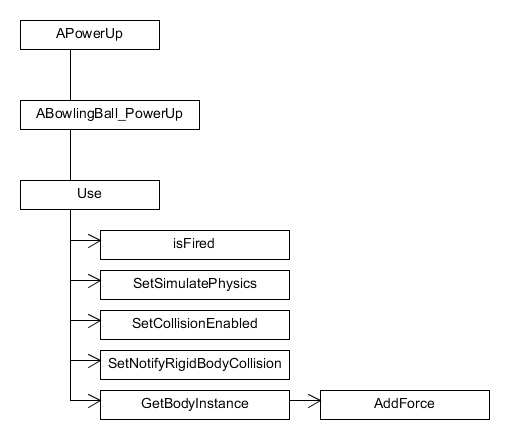
* 1. **Process View of the ABowlingBall\_PowerUp Submodule**

**TICK**

****

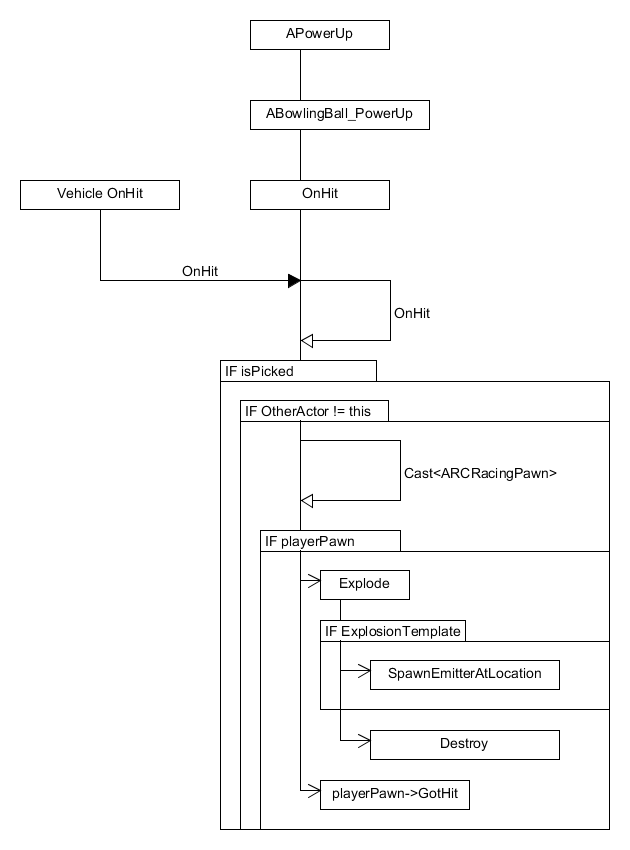
The tick sequence of ABowlingBall\_PowerUp starts when the game begins but the sequence only triggers if the power up was fired. Upon firing, TimeUntilDespawn is updated by DeltaTime. If TimeUntilDespawn goes over the threshold MaxTimeUntilDespawn (Value of 5.0f), it will destroy itself and reset TimeUntilDespawn to zero. This sequence is here only to make sure the power up will be destroy if it doesn’t hit a vehicle within 5 seconds.

**USE**

****

The Use sequence of ABowlingBall\_PowerUp starts when ARCRacingPawn uses its current power up. The goal if this sequence is to reenable different options that were initially disabled, such as physics and collisions, then add force to the actor. Upon use, the switch isFired is set to true (which triggers the Tick function). The collision is set to collisionEnabled::QueryAndPhysics and RigidBody is set to true. Finally, we add force to the sphere going in the opposite direction from the vehicle. The force is set at a 1000 multiplied by the mass of the sphere.

**ONHIT**

****

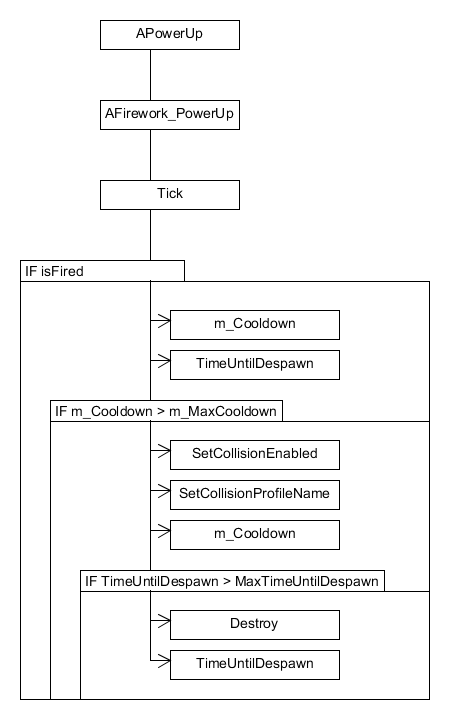
The OnHit sequence of ABowlingBall\_PowerUp starts when the power up collision sphere hits a vehicle. When it begins, it first checks if it already has been picked by another player. If not, it will safe check if the OtherActor is, in fact, a ARCRacingPawn.

If so, the function Explode will be called. First, it will safe check if ExplosionTemplate is declared. If so, it will spawn the ExplosionTemplate on the actor location then it will destroy itself.

Then, it will call player’s GotHit function, setting the vehicle mesh physics linear velocity on the Z axis to 500.0f and will briefly stop moving.

* 1. **Process View of the AFirework\_PowerUp Submodule**

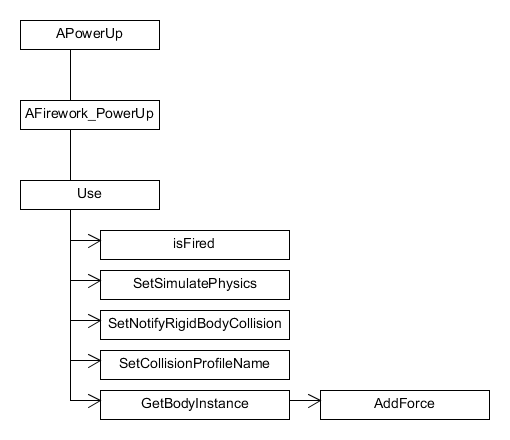
**TICK**

****

The Tick sequence of AFirework\_PowerUp starts when the game begins but the sequence only triggers if the power up was fired. Upon firing, the cooldown and despawn timers are updated by DeltaTime. To ensure the power up won’t collide with the emitter (vehicle pawn), its collision and profile name won’t be set until m\_Cooldown goes over the threshold MaxCooldown (Value of 2.0f). If so, the collision is set to collisionEnabled::QueryAndPhysics and its profile name is set to BlockAllDynamic. Finally, the cooldown is reset to zero.

At the same time, TimeUntilDespawn is updated by DeltaTime. If TimeUntilDespawn goes over the threshold MaxTimeUntilDespawn (Value of 5.0f), it will destroy itself and reset TimeUntilDespawn to zero. This sequence is here only to make sure the power up will be destroy if it doesn’t hit a vehicle within 5 seconds.

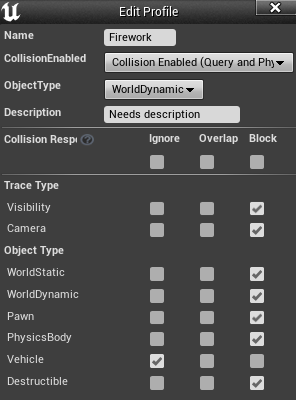
**USE**

****

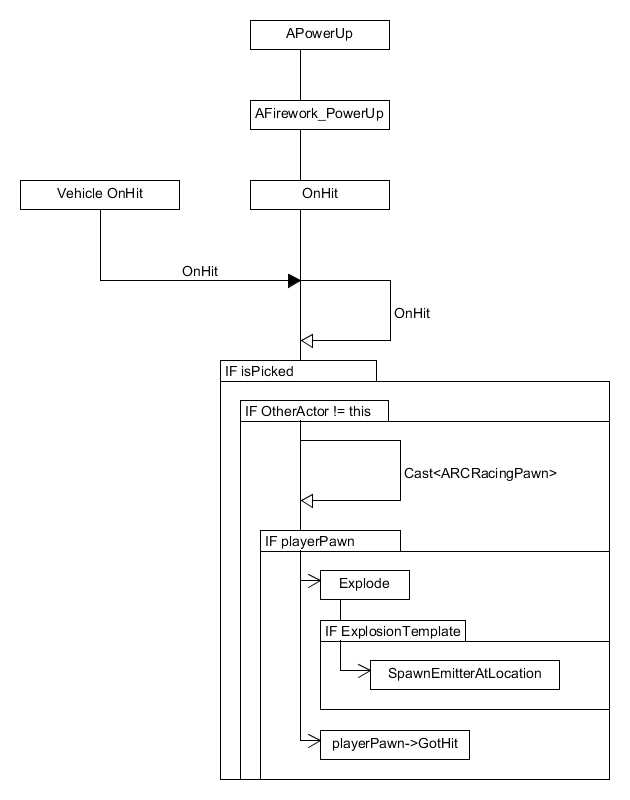
The Use sequence of AFirework\_PowerUp starts when ARCRacingPawn uses its current power up. The goal if this sequence is to reenable different options that were initially disabled, such as physics, collisions and profile name, then add force to the actor.

Upon use, the switch isFired is set to true (which triggers the Tick function). Simulate physics and rigid body are set to true and the collision profile name is set to Firework, so the power up can ignore the vehicle. Finally, we add force to the sphere going in the same direction as the vehicle. The force is set at a 50000 multiplied by the mass of the sphere.

The following image represent the firework collision profile name.



**ONHIT**

****

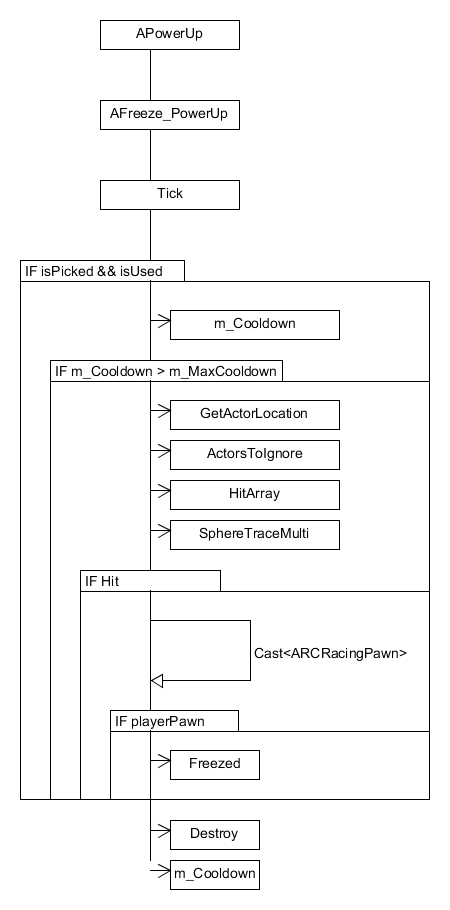
The OnHit sequence of AFirework\_PowerUp starts when the power up collision sphere hits a vehicle. When it begins, it first checks if it already has been picked by another player. If not, it will safe check if the OtherActor is, in fact, a ARCRacingPawn.

If so, the function Explode will be called. First, it will safe check if ExplosionTemplate is declared. If so, it will spawn the ExplosionTemplate on the actor location then it will destroy itself.

Then, it will call player’s GotHit function, setting the vehicle mesh physics linear velocity on the Z axis to 500.0f and will briefly stop moving.

**5.3 Process View of the AFreeze\_PowerUp Submodule**

**TICK**

****

The Tick sequence of AFreeze\_PowerUp starts when the game begins but the sequence only triggers if the power up was picked and used. The overall goal of this sequence is to detect the actors and pawns in a radius around the power up by using SphereTraceMulti then apply the freeze effect when it hits the right target.

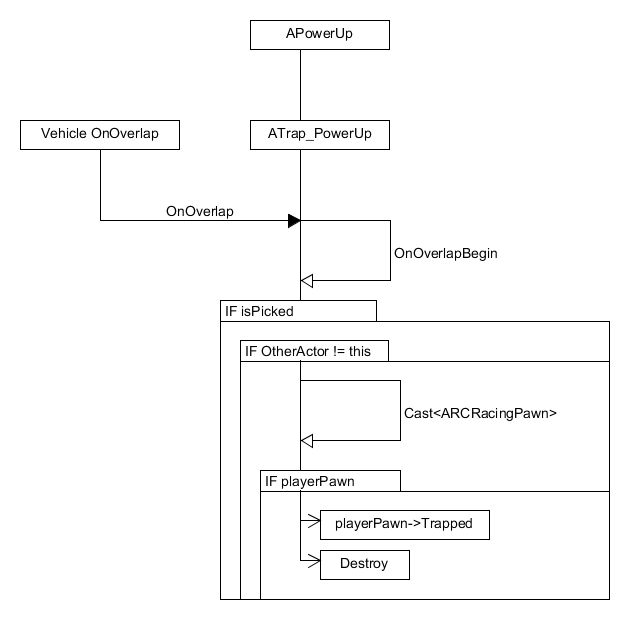
Upon succeeding both checks, the cooldown timer is updated by DeltaTime. When m\_Cooldown goes over the threshold MaxCooldown (Value of 2.0f), it registers the actor location and create two TArrays:

1. AActor\*: used as an ignore list for the SphereTraceMulti.
2. FHitResult\*: used to get the hit result from SphereTraceMulti.

SphereTraceMulti will return a boolean. If the power up hits something, it will loop the hit array and will safe check if the hit result is, in fact, a ARCRacingPawn. If so, the function will call the players’s freezed function, reducing its overall physical linear velocity to zero. Finally, the power up will destroy itself and reset the cooldown value to zero.

* 1. **Process View of the ATrap\_PowerUp Submodule**

**ONOVERLAP**

****

The Overlap sequence of ATrap\_PowerUp starts when the power up collision sphere overlaps a vehicle. 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.

When it begins, it first checks if it already has been picked by another player. If not, it will safe check if the OtherActor is, in fact, a ARCRacingPawn. If so, it will call the player’s Trapped function.

If a primitive component of type mesh is set on the vehicle, 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. Finally, the power up will destroy itself.

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