**CAR COMBAT GAME**

Lucas Krespi dos Santos

Julian Escobar Echeverri

George Brown College

CNR – 12515: Game Production 3

Instructor: Daniel Posner

December 4, 2020

Contents

[1. INTRODUCTION 3](#_Toc58007796)

[1.1 Requirements 3](#_Toc58007797)

[2. GAME ELEMENTS 3](#_Toc58007798)

[2.1 Turrets 3](#_Toc58007799)

[2.2 Hazards 4](#_Toc58007800)

[2.2.1 Recharge 4](#_Toc58007801)

[2.2.2 Hurt 5](#_Toc58007802)

[2.2.3 Hole. 5](#_Toc58007803)

[2.2.4 Boost 6](#_Toc58007804)

[2.3 Player 6](#_Toc58007805)

[2.4 Map 7](#_Toc58007806)

[2.5 UI. 8](#_Toc58007807)

[2.6 How to play. 8](#_Toc58007808)

[3. Vehicle Code C++ 9](#_Toc58007809)

[3.1 VehiclePawn.h 9](#_Toc58007810)

[3.2 VehiclePawn.cpp 12](#_Toc58007811)

[3.3 Code Explanation. 17](#_Toc58007812)

Figure Table

[Figure 1-Enemy Turret 3](#_Toc52280879)

[Figure 2 - Slow Hazard 4](#_Toc52280880)

[Figure 3 - Damage Hazard 4](#_Toc52280881)

[Figure 4 - Kill Hazard 4](#_Toc52280882)

[Figure 5 - Boost 5](#_Toc52280883)

[Figure 6 - player car 5](#_Toc52280884)

[Figure 7 - Map Prototype 6](#_Toc52280885)

# INTRODUCTION

This Game is a single player car combat game, where the player is on a dungeon like level and has to destroy enemy turrets in order to win the game, there is a total of 12 turrets placed on the map, this turret will shoot projectiles to try to destroy the player car. The car is going to have one gun to shoot back at the enemies. The level will have some boosts and hazards, the hazards cause damage, slow or both depending on the type, and the boosts will push the car forward fast.

## Requirements

If you want to Run the game in Unreal you need to do these requirements before hand:

-Install FMOD plugin in unreal.

-Install FMOD in your computer.

-Add the FMOD events contained in the folder into your Master Bank inside FMOD.

If you do not have these requirements the game is going to work but without the Car’s Engine sound.

Also, if you just want to play the game, you do not need these requirements, these are just for unreal, so just go to the ShippingPackage folder and open the .Exe file.

# GAME ELEMENTS

## Turrets

Enemy turrets will shoot projectiles that deal damage to the player, the projectiles are shoot every 5-10 seconds and only when the player is in Range (2000 Unreal units), also the turrets are only going to Aim the player when it is inside their Range. To destroy a Turret, they need to be hit by 3 bullets.



Figure 1-Enemy Turret

## Hazards

We have 3 types if hazards, the first one causes a slow on the player, the second one deals damage, and the third destroys the player, they are just a tile on de ground with a different light on top to differ the type.

### 2.2.1 Recharge

Recharge is going to add 1 bullet every 0.5 sec to the player when the player is on top of it, this can be cancelled if the player is hit by a bullet or exits the Recharge zone.

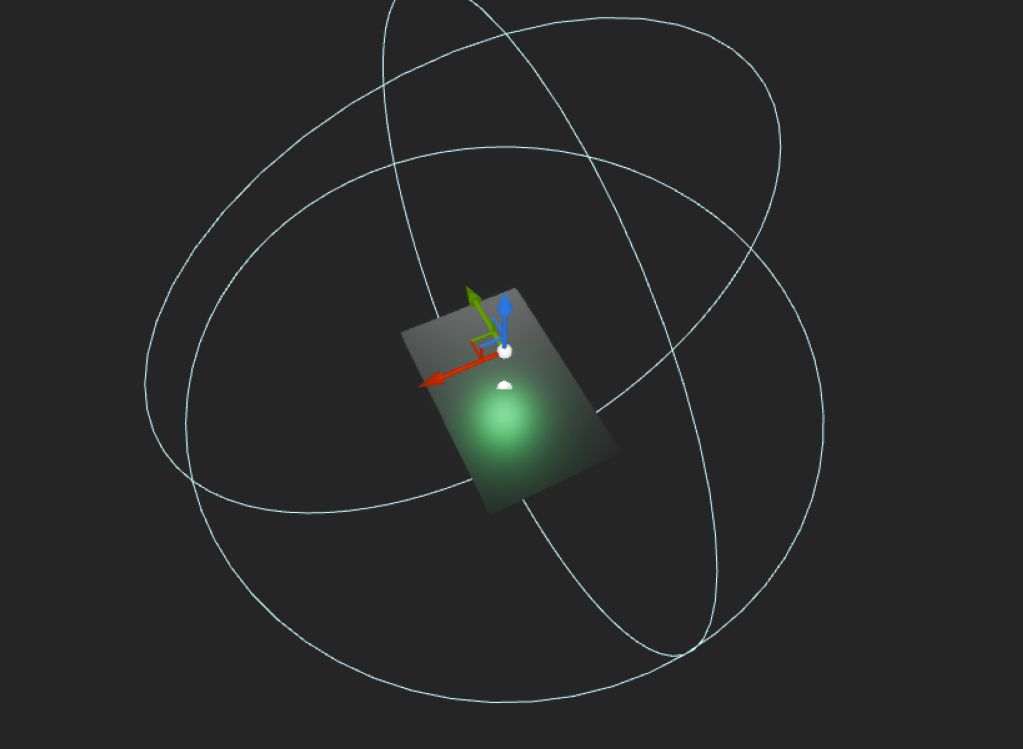


Figure 2 - Slow Hazard

### 2.2.2 Hurt

Hurt is going to apply 5 damage to the player, reducing the health by 5.

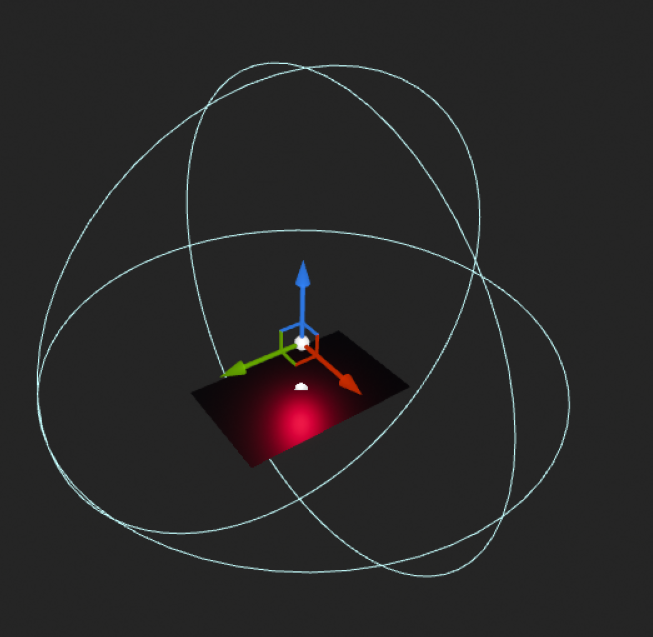


Figure 3 - Damage Hazard

### 2.2.3 Hole.

Hole is going to proc the Game Over screen when the player is on top of it.

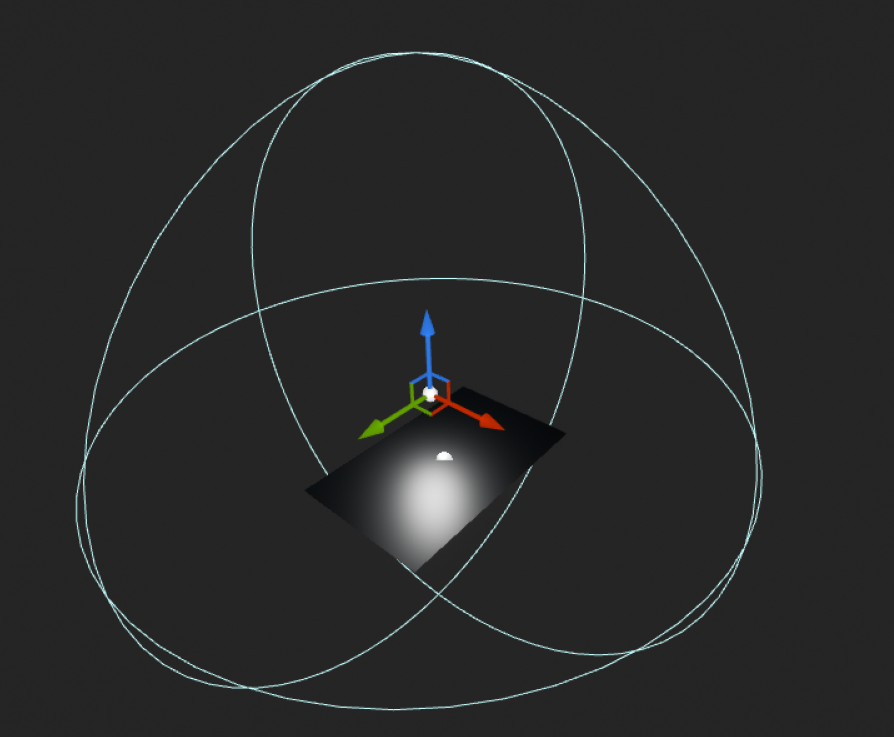


Figure 4 - Kill Hazard

### 2.2.4 Boost

Boost will be placed just before the ramps and will increase the car speed by twice the current speed.

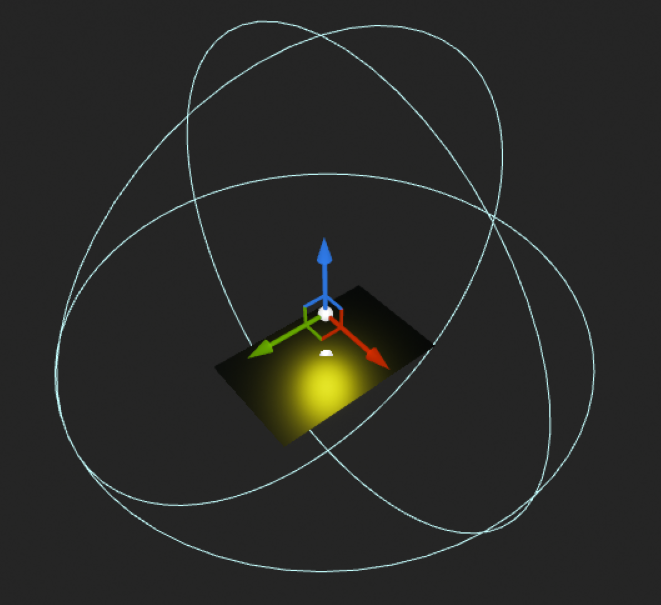


Figure 5 - Boost

## Player

For now, the player car is the unreal default car model. But in the future, it is going to be changed for and asset with guns, that can shoot, and a life bar. The car can move with WASD keys and is affected by the hazards and boosts.



Figure 6 - player car

## Map

The Map will be a dungeon like with all the elements listed above, where the player can move between the edges, every other asset will be stationary.

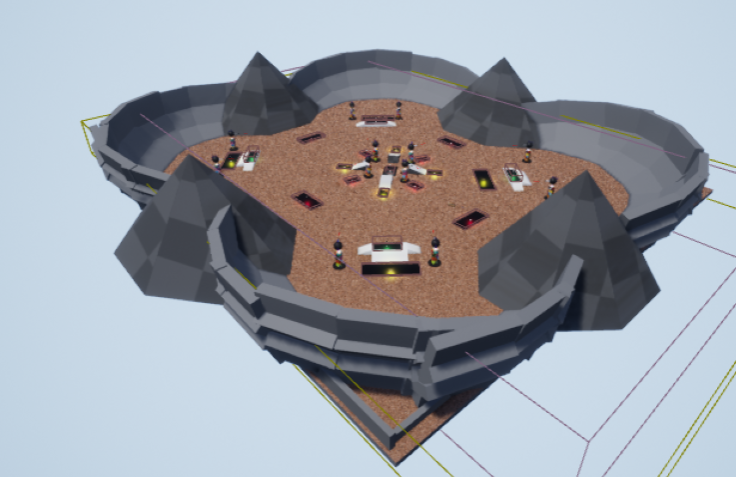


Figure 7 - Map Prototype

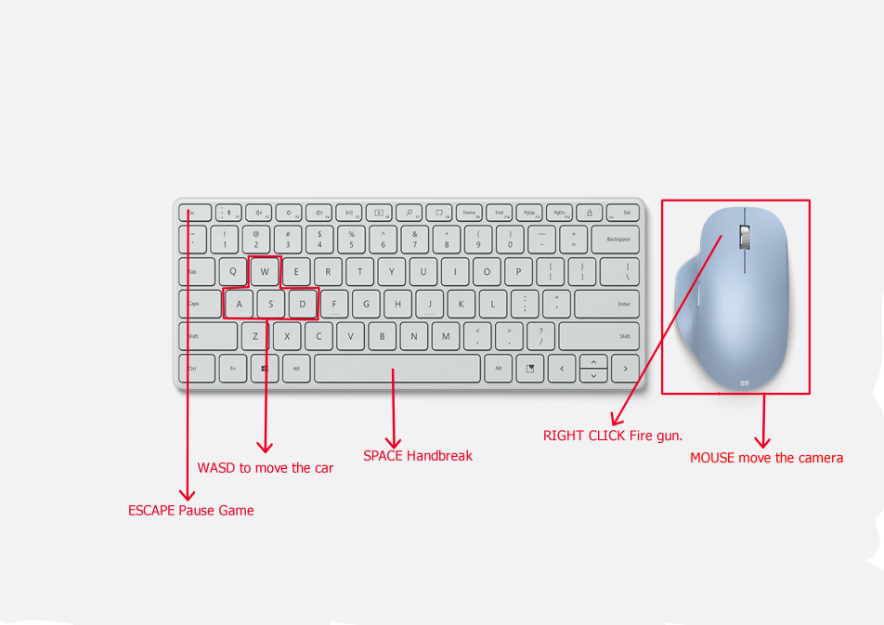
## UI.

The UI is really simple, it has a bar on the top that represent how much health the player has, it has a timer bellow this bar that show how many seconds has passed since we started the game, it also has 3 labels on the left side that shows the amount of turrets online (alive), the current velocity and the amount of bullets that we posses in the moment respectively.



## How to play.

This image can be found inside the game in the Main Menu -> How to play.



# Vehicle Code C++

## VehiclePawn.h

#pragma once

#include "CoreMinimal.h"

#include "WheeledVehicle.h"

#include "GameFramework/SpringArmComponent.h"

#include "VehiclePawn.generated.h"

/\*\*

\*

\*/

UCLASS()

class GAME\_PRODUCTION\_01\_API AVehiclePawn : public AWheeledVehicle

{

GENERATED\_BODY()

public:

AVehiclePawn();

virtual void Tick(float DeltaSeconds) override;

virtual void SetupPlayerInputComponent(UInputComponent\* PlayerInputComponent) override;

//throttle/steering

void ApplyThrottle(float val);

void ApplySteering(float val);

//Look around

void LookUp(float val);

void LookRight(float val);

//handbrake

void OnHandBrakePressed();

void OnHandBrakeReleased();

//Update in air Physics

void UpdateInAirControl(float DeltaTime);

protected:

//Spring arm that will offset the camera

UPROPERTY(Category = "InAirMovement", EditAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))

float AirMovement;

UPROPERTY(Category = "InAirMovement", EditAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))

float GroundMovement;

//Camera component that will be our viewpoint

/\*UPROPERTY(Category = Camera, EditDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

class UCameraComponent\* Camera;\*/

};

## VehiclePawn.cpp

// Fill out your copyright notice in the Description page of Project Settings.

#include "VehiclePawn.h"

#include "GameFramework/SpringArmComponent.h"

#include "Components/SkeletalMeshComponent.h"

#include "Camera/CameraComponent.h"

#include "Components/InputComponent.h"

#include "WheeledVehicleMovementComponent4W.h"

static const FName NAME\_SteerInput("MoveRight");

static const FName NAME\_ThrottleInput("MoveForward");

AVehiclePawn::AVehiclePawn()

{

//just gonna use some Unity functions that already are defined in term of vehicles. we can play with the if we want but in this case we are just going to use them.

UWheeledVehicleMovementComponent4W\* Vehicle4W = CastChecked<UWheeledVehicleMovementComponent4W>(GetVehicleMovement());

//Adjust the tire loading

Vehicle4W->MinNormalizedTireLoad = 0.0f;

Vehicle4W->MinNormalizedTireLoadFiltered = 0.2f;

Vehicle4W->MaxNormalizedTireLoad = 2.0f;

Vehicle4W->MaxNormalizedTireLoadFiltered = 2.0f;

//Torque Setup

Vehicle4W->MaxEngineRPM = 5700.f;

Vehicle4W->EngineSetup.TorqueCurve.GetRichCurve()->Reset();

Vehicle4W->EngineSetup.TorqueCurve.GetRichCurve()->AddKey(0.f, 400.0f);

Vehicle4W->EngineSetup.TorqueCurve.GetRichCurve()->AddKey(1890.0f, 500.0f);

Vehicle4W->EngineSetup.TorqueCurve.GetRichCurve()->AddKey(5730.0f, 400.0f);

//Adjust the steering

Vehicle4W->SteeringCurve.GetRichCurve()->Reset();

Vehicle4W->SteeringCurve.GetRichCurve()->AddKey(0.0f, 1.0f);

Vehicle4W->SteeringCurve.GetRichCurve()->AddKey(40.0f, 0.7f);

Vehicle4W->SteeringCurve.GetRichCurve()->AddKey(120.0f, 0.6f);

Vehicle4W->DifferentialSetup.DifferentialType = EVehicleDifferential4W::LimitedSlip\_4W;

Vehicle4W->DifferentialSetup.FrontRearSplit = 0.65;

//Automatic gearbox

Vehicle4W->TransmissionSetup.bUseGearAutoBox = true;

Vehicle4W->TransmissionSetup.GearSwitchTime = 0.15f;

Vehicle4W->TransmissionSetup.GearAutoBoxLatency = 1.0f;

GroundMovement = 6.f;

AirMovement = 1.f;

//Create a spring arm component for our chase camera

/\*SpringArm = CreateDefaultSubobject<USpringArmComponent>(TEXT("SpringArm"));

SpringArm->SetupAttachment(RootComponent);

SpringArm->TargetArmLength = 250.f;

SpringArm->bUsePawnControlRotation = true;\*/

//Create the chase camera component, also avoid the camera going thru walls

/\*Camera = CreateDefaultSubobject<UCameraComponent>(TEXT("ChaseCamera"));

Camera->SetupAttachment(SpringArm, USpringArmComponent::SocketName);

Camera->FieldOfView = 90.f;\*/

}

void AVehiclePawn::Tick(float DeltaSeconds)

{

Super::Tick(DeltaSeconds);

UpdateInAirControl(DeltaSeconds);

}

void AVehiclePawn::SetupPlayerInputComponent(UInputComponent\* PlayerInputComponent)

{

Super::SetupPlayerInputComponent(PlayerInputComponent);

//Axis

PlayerInputComponent->BindAxis(NAME\_ThrottleInput, this, &AVehiclePawn::ApplyThrottle);

PlayerInputComponent->BindAxis(NAME\_SteerInput, this, &AVehiclePawn::ApplySteering);

//PlayerInputComponent->BindAxis("LookUp", this, &AVehiclePawn::LookUp);

//PlayerInputComponent->BindAxis("LookRight", this, &AVehiclePawn::LookRight);

////Actions

//PlayerInputComponent->BindAction("HandBreak", IE\_Pressed, this, &AVehiclePawn::OnHandBrakePressed);

//PlayerInputComponent->BindAction("HandBreak", IE\_Released, this, &AVehiclePawn::OnHandBrakeReleased);

}

void AVehiclePawn::ApplyThrottle(float val)

{

GetVehicleMovementComponent()->SetThrottleInput(val);

}

void AVehiclePawn::ApplySteering(float val)

{

GetVehicleMovementComponent()->SetSteeringInput(val);

}

void AVehiclePawn::LookUp(float val)

{

if (val != 0.0f)

{

AddControllerPitchInput(val);

}

}

void AVehiclePawn::LookRight(float val)

{

if (val != 0.0f)

{

AddControllerYawInput(val);

}

}

void AVehiclePawn::OnHandBrakePressed()

{

GetVehicleMovementComponent()->SetHandbrakeInput(true);

}

void AVehiclePawn::OnHandBrakeReleased()

{

GetVehicleMovementComponent()->SetHandbrakeInput(false);

}

void AVehiclePawn::UpdateInAirControl(float DeltaTime)

{

if(UWheeledVehicleMovementComponent4W\* Vehicle4W = CastChecked<UWheeledVehicleMovementComponent4W>(GetVehicleMovement()))

{

FCollisionQueryParams QueryParams;

QueryParams.AddIgnoredActor(this);

const FVector TraceStart = GetActorLocation() + FVector(0.f,0.f,50.f);

const FVector TraceEnd = GetActorLocation() - FVector(0.f,0.f,200.f);

FHitResult Hit;

//Check if the car is flipped on its side, and check if the car is in air

const bool bInAir = !GetWorld()->LineTraceSingleByChannel(Hit, TraceStart, TraceEnd, ECC\_Visibility, QueryParams);

const bool bNotGrounded = FVector::DotProduct(GetActorUpVector(), FVector::UpVector) < 0.1f;

//Only allow in air-movement if we are not on the ground, or are in the air

if(bInAir || bNotGrounded)

{

const float ForwardInput = InputComponent->GetAxisValue(NAME\_ThrottleInput);

const float RightInput = InputComponent->GetAxisValue(NAME\_SteerInput);

//If car is grounded allow player to roll the car over

float AirMovementForcePitch = AirMovement;

float AirMovementForceRoll = !bInAir && bNotGrounded ? GroundMovement : AirMovement;

if(UPrimitiveComponent\* VehicleMesh = Vehicle4W->UpdatedPrimitive)

{

const FVector MovementVector = FVector(RightInput \* -AirMovementForceRoll, ForwardInput \* AirMovementForcePitch, 0.f) \* DeltaTime \* 200.f;

const FVector NewAngularMovement = GetActorRotation().RotateVector(MovementVector);

VehicleMesh->SetPhysicsAngularVelocity(NewAngularMovement, true);

}

}

}

}

## Code Explanation.

We are going to explain the call in the order every function is called, so we begin saying that this VehiclePawn class has a Parent class called AWheeledVehicle which is an Unreal function created to facilitate the implementation of vehicles in c++. So, in this class we are going to take advantage of this class and use it to customize our vehicle.

The first thing that we do is the constructor, inside the constructor we are going to set all the parameters for our vehicle, starting with tire load, the setting the torque curve which is going to affect directly how the Car’s engine is going to behave, for example the acceleration and speed. After, we setup the steering curve which is going to restrict how much steering can I do depend on the current speed. Then we setup the gear box, configuring setting like the delay when a change gear or if the box is automatic or manual. And last, we initialize 2 variables that are related to my air movement.

The Tick function comes after the constructor and the only thing that has is a function called UpdateInAirControl, which is the one that controls how much force the car is going to have when is on the ground or in the air. We want to control this and differentiated because if the car is flipped, we want to flip it back to normal so we can still play.

After this we can Input functions like: ApplyThrottle, ApplySteering, LoopUp, LookRight and OnHandBreakPressed. These functions are going to be the ones who takes the input from the player and apply the values to the vehicle. How they work is they take the exact name of the variable input settled in the unreal project and take the value of it, then it takes that value and apply to the already created functions for the AWheeledVehicle and those function apply those values to the engine.