**DYNAMIC CAR RACING**

**A PROJECT REPORT**

***Submitted by***

**Dhruv Arora** (19MIM10014)

**Shrey Shrivastava** (19MIM10056)

**Asif Ali Sherani** (19MIM10104)

**Naman Arora** (19MIM10057)

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

*Specialization in*

***Artificial intelligence and machine learning***



**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

**VIT BHOPAL UNIVERSITY**

**KOTHRIKALAN, SEHORE**

**MADHYA PRADESH – 466114**

OCT 2020

**VIT BHOPAL UNIVERSITY, KOTHRIKALAN, SEHORE**

**MADHYA PRADESH – 466114**

**BONAFIDE CERTIFICATE**

Certified that this project report titled **“DYNAMIC CAR RACING”** the bonafide work of **Dhruv Arora, Shrey Shrivastava, Asif Ali Sherani and Naman Arora** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported here does not form part of any other project / research work on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**PROGRAM CHAIR PROJECT GUIDE**

Dr. V. Pandimurugan Dr. M. Ashwin

School of AI & ML division School of AI & ML division

VIT BHOPAL UNIVERSITY VIT BHOPAL UNIVERSITY

**ACKNOWLEDGEMENT**

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to Dr. V. Pandimurugan, Head of the Department, School of AI & ML division for much of his valuable support encouragement in carrying out this work.

I would like to thank my internal guide Dr. M. Ashwin for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of School of Computer Science and Engineering, who extended directly or indirectly all support.

Last, but not the least, I am deeply indebted to my patents who have been the greatest support while I worked day and night for the project to make it a success.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| 1 | Behavior tree modelling the search and grasp plan of a two-armed robot | 5 |
| 2 | Software workflow | 10 |
| 3 | Sprite Controller | 12 |
| 4 | Audio Circuitry in Sockit Board | 13 |
| 5 | Audio Controller Function Block Diagram | 13 |
| 6 | Player car | 26 |
| 7 | Non-playable character | 26 |
| 8 | Navigation mesh | 27 |
| 9 | Behavior tree 1 | 27 |
| 10 | Behavior tree 2 | 28 |
| 11 | Car model Animation | 28 |

**ABSTRACT**

This paper presents a car racing simulator game called Racer, in which the human player races a car against three game-controlled cars in a three-dimensional environment. The objective of the game is not to defeat the human player, but to provide the player with a challenging and enjoyable experience. To ensure that this objective can be accomplished, the game incorporates artificial intelligence (AI) techniques, which enable the cars to be controlled in a manner that mimics natural driving. The paper provides a brief history of AI techniques in games, presents the use of AI techniques in contemporary video games, and discusses the AI techniques that were implemented in the development of Racer. A comparison of the AI techniques implemented in the Unreal engine platform with traditional AI search techniques is also included in the discussion.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | List of Abbreviations  List of Figures  List of Tables  Abstract | iv  v  vi  vii |
| 1 | **INTRODUCTION**  1.1 Introduction  1.2 Motivation for the work  1.3 Objective of the work  1.4 Summary | 1-3  1  2  2  3 |
| 2 | **LITERATURE SURVEY**  2.1 Literature review  2.2 Idea of project  2.3 Behaviour tree  2.4 Summary | 4-6  4  4  4-5  6 |
| 3 | **FRAMEWORK**  3.1 Existing work  3.2 Limitations/disadvantages of existing work  3.3 Basic system requirements  3.4 Software requirements – Unreal Engine  3.5 Summary | 7-9  7  7-8  8  8-9  9 |
| 4 | **SYSTEM DESIGN AND IMPLEMENTATION**  4.1 Module description  4.1.1 Module 1 – Game logic module  4.1.2 Module 2 – Sprite control module  4.1.3 Module 3 – Audio module  4.2 Module work flow  4.3 Coding | 10-21  10-14  10-11  11-12  12-14  14  14-21 |
| 5 | **PERFORMANCE ANALYSIS**  5.1 Performance measures  5.2 Performance analysis | 22-23  22  22-23 |
| 6 | **FUTURE DEVELOPMENT PLANS AND CONCLUSION**  6.1 Limitations of the program  6.2 Future plans  6.3 Conclusion | 24-25  24  24  24-25 |
| 7 | **BASIC LOOK OF THE GAME**  7.1 Snapshots | 26-28  26-28 |

**Introduction**

**1.1 Introduction**

For this project, our group would like to implement a Car Racing Game. We try to implement a simple car game based on the basic design. The object of this game is to survive as long as possible, get to the High scores in the shortest possible time while avoiding the obstacles on the tracks and also to win the race against an AI bot by reaching the finish line first,

In the fast-growing field of software engineering and development and even more rapidly growing sector of game development the future is hard to predict. We are working with this game and as a part of our degree we choose this type of work for doing better with development cycle, development period, graphics, scripting, adopting new technology, animation.

In general, software project is a project focusing on creation of software. Consequently, Success can be measured by a taking look at the resulting software.

In a game project, the product is game. But and here comes the point: A game is much more than just its software. It has to provide content to become enjoyable. Just like a web server: Without content the server is useless, and the quality cannot be measured. This has an important effect on the

game project as a whole. The software part of the project is not the only one, and it must be considered in connection to all the other parts: The environment of the game, The story, characters, game plays, and the artwork.

The purpose of this document is to outline all aspects of the project creates. Over the following document I shall, explain the reasons for the game,

explain how the game works and give you our idea for the future of this game and others which will be made from it.

**1.2 Motivation for the work**

Games have become an integral part of everyday life for many people. A traditional game often presents a situation where “players engage in an artificial conflict, defined by rules and results in a quantifiable outcome”. Such artificial conflicts are often represented as a puzzle or a challenge, and having the puzzle solved or the challenge resolved provides a real-world purpose to the game players.

While this is true for other genres of games, racing games lacks in providing a new experience to the players. Racing games are fun in the beginning but as the time passes players tend to get used to the AI’s movements and which in turn makes the gameplay stale.

Motivation for this project is to develop a dynamic AI which gives a new experience to the player. This AI will emulate human game playing style which will look natural to the players playing the game and will post a challenge for the players making the game more comprehensive.

**1.3 Objective of the work**

Games have become an integral part of everyday life for many people. The artificial intelligence (AI) community has witnessed a similar transition from the “classical AI games” such as Samuel’s Checker Player and Waterman’s Poker Player to the contemporary AI techniques adopted in electronic games. The objective of the game is no longer a quantifiable outcome of beating the opponent in a checker or poker game. Instead, a contemporary game contains changing environments, multiple objectives, and dynamic aspects of the game that are revealed to the game player as the game unfolds. The objective is to offer to the human player an enjoyable experience through his or her interaction with the game, and this does not involve any specific quantifiable outcome.

The gameRacer is a contemporary video game, and its objective is to offer the player a challenging and enjoyable experience in a car race against a game-controlled car. Although the player’s goal within the game is to win the race against the game-controlled car, the AI techniques adopted in the game are primarily designed to give the player an enjoyable time racing his or her car. In other words, the objective of winning by either side is not given the highest priority.

The main objective of this game is that the AI controlled car would adapt the skill level of the human controlled car and increase of decrease its difficulty level accordingly.

**1.4 Summary**

As the software industry progresses at full pace bringing us closer and closer to software of the future which is revolutionizing the way we interact with our surroundings. The gaming is not just a computer enthusiast’s dream anymore. Gaming on computers has become an integral part of our lives and an important part of gaming is the game-play experience which is heavily impacted by the implementation of artificial intelligence. The AI implementation in the games for the past decade has plateaued due to same implementation in games of similar genre games. What we are trying to achieve is to use a different approach to AI implementation which if implemented successfully has the ability to revolutionize the racing game industry.

**Literature Survey**

**2.1 Literature Review**

There has been substantial research work that focuses on developing AI-controlled components of game systems, which can approximate or emulate human game playing styles.

These components are often referred to as “bots.” The motivation for developing the “bots” is that a human player’s enjoyment in the gaming experience will be higher if he or she can be led to believe that the opponents in the game are other human players.

Similar AI-controlled components or “bots” also appear in the early video game of Pac Man; the computer-controlled ghosts in the maze are able to move towards the player-controlled characters because the former incorporated some path finding algorithms.

Other AI algorithms that were adopted by games include finite state machines, fuzzy state machines, and decision trees. Decision trees are used to represent the decision-making process involved in games like Checkers or Chess.

Similarly, in driving and racing games, human players expect to control their cars by making small adjustments to the car’s direction while driving, just like how they would control a car in real life.

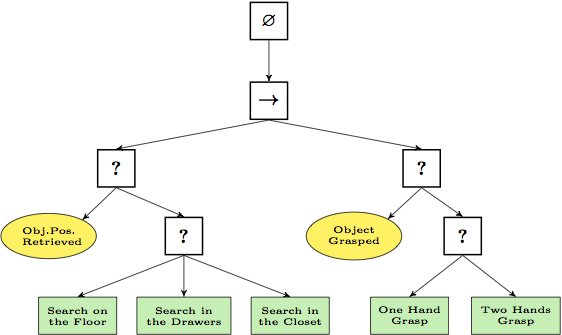
**2.2 Idea of the Project**

Idea of the project is to build a dynamic AI which will provide human like gameplay. This AI will adapt to the opponent’s play style making the game more challenging and strategic. This dynamic AI gives the player a natural experience with countless possibilities to explore making the gameplay more erratic.

**2.3 Behavior Tree**

A behaviour tree is a mathematical model of plan execution used in computer science, robotics, control systems and video games. They describe switching between a finite set of tasks in a modular fashion. Their strength comes from their ability to create very complex tasks composed of simple tasks, without worrying how the simple tasks are implemented. Behaviour trees present some similarities to hierarchical state machines with the key difference that the main building block of a behaviour is a task rather than a state. Its ease of human understanding makes behaviour trees less error prone and very popular in the game developer community. Behaviour trees have been shown to generalize several other control architectures. Mathematically, they are directed acyclic graphs.

Behavior trees originate from the computer game industry as a powerful tool to model the behavior of non-player characters (NPCs). They have been extensively used in high-profile video games such as Halo, Bioshock, and Spore.



*Figure 1*

A behavior tree is graphically represented as a directed tree in which the nodes are classified as root, control flow nodes, or execution nodes (tasks). For each pair of connected nodes, the outgoing node is called parent and the incoming node is called child. The root has no parents and exactly one child, the control flow nodes have one parent and at least one child, and the execution nodes have one parent and no children. Graphically, the children of a control flow node are placed below it, ordered from left to right.

The execution of a behavior tree starts from the root which sends ticks with a certain frequency to its child. A tick is an enabling signal that allows the execution of a child. When the execution of a node in the behavior tree is allowed, it returns to the parent a status running if its execution has not finished yet, success if it has achieved its goal, or failure otherwise.

**2.4 Summary**

These components are often referred to as “bots.” The motivation for developing the “bots” is that a human player’s enjoyment in the gaming experience will be higher if he or she can be led to believe that the opponents in the game are other human players. Idea of the project is to build a dynamic AI which will provide human like gameplay which will look natural to the opponent player. A behaviour tree is a mathematical model of plan execution used in computer science, robotics, control systems and video games.

**FRAMEWORK**

**3.1 Existing work**

Today there are numerous racing games that uses the technology of artificial intelligence in the racing bots in the game.

Traditionally, hard-coded methods such as state machines, rule bases systems, and scripting are used for AI in commercial games. The problem with these methods is that they are static, can be hard to expand, and time is needed to hand tune parameters.

All AI bots are all the same due to whichthe real time players gets bored playing the same game again and again. As such it seems that the bots are not interactive with the players.

Some of the racing games available nowadays are Need for speed (NFS), Forza horizon, real racing etc.

In addition to decision trees, other AI techniques are also used for building bots so that they would mimic or emulate human behaviour.

In driving and racing games, human players expect to control their cars by making small adjustments to the car’s direction while driving, just like how they would control a car in real life.

**3.2 Limitations/disadvantages of existing work**

Some of the limitations/disadvantages of existing car racing games are: -

1. Low-fidelity simulators may evoke unrealistic driving behaviour and therefore produce invalid research outcomes. Simulator fidelity is known to affect user opinion. Participants may become demotivated by a limited-fidelity simulator and prefer a real vehicle instead (or a more costly high-fidelity simulator for that matter). Interestingly, while safety is often cited as an advantage of driving simulation, sometimes this same feature is interpreted as a disadvantage.
2. A growing body of evidence indicates that driving-simulator measures are predictive for on-the-road driving performance. However, only a few studies have investigated whether skills learned in a driving simulator transfer to the road.
3. Simulator sickness symptoms may undermine training effectiveness and negatively affect the usability of simulators. This is a serious concern, but fortunately, useful technological and procedural guidelines are available to alleviate it. Research shows that simulator sickness is less of a problem for young drivers.

**3.3 Basic system requirements**

* + Recommended Hardware
    - * Processor Intel q6600 or AMD Fx-4350 or higher
      * Memory 8GB or higher
      * Video Card DirectX version 10.0 or higher
  + Recommended Software
    - * Operating System windows 7/8/8.1 or 10
      * DirectX version 10.0 or higher installed
  + Developing software
    - * Visual studio 2017 or higher with unreal engine 4

**3.4 Software requirements – Unreal Engine**

Unreal Engine (currently released as Unreal Engine 4) is a popular and widely-used game engine developed by Epic Games.

It is used in many modern AAA games like Epic’s own battle royale shooter Fortnite or other hit games like Psyonix’s ‘Rocket League’.

It allows for development across multiple platforms from PC to consoles such as the PS4, Xbox One, and Nintendo Switch. This is part of the reason it is so widely used, due to its flexibility to work between these different platforms.

More experienced programmers can use the C++ language to make their own scripts that run in the game engine. More hobbyist developers can utilise its very powerful blueprints which are basically prefabricated blocks of code you can add to your objects to make interactions.

It also has powerful material and animation tools for artists which allow you to make complex scenes fast. Setting up some of these features can seem daunting at first, but several examples are provided where you can just change the parameters until you create what you’re looking for.

**3.5 Summary**

Traditionally, hard-coded methods such as state machines, rule bases systems, and scripting are used for AI in commercial games. The problem with these methods is that they are static, can be hard to expand, and time is needed to hand tune parameters. As such it seems that the bots are not interactive with the players. Low-fidelity simulators are one of the main limitations of car racing games that exist in today’s world.

Unreal engine is a popular and widely-used game engine developed by Epic Games. It allows for development across multiple platforms from PC to consoles such as the PS4, Xbox One, and Nintendo Switch. This is part of the reason it is so widely used, due to its flexibility to work between these different platforms. It also has powerful material and animation tools for artists which allow you to make complex scenes fast.

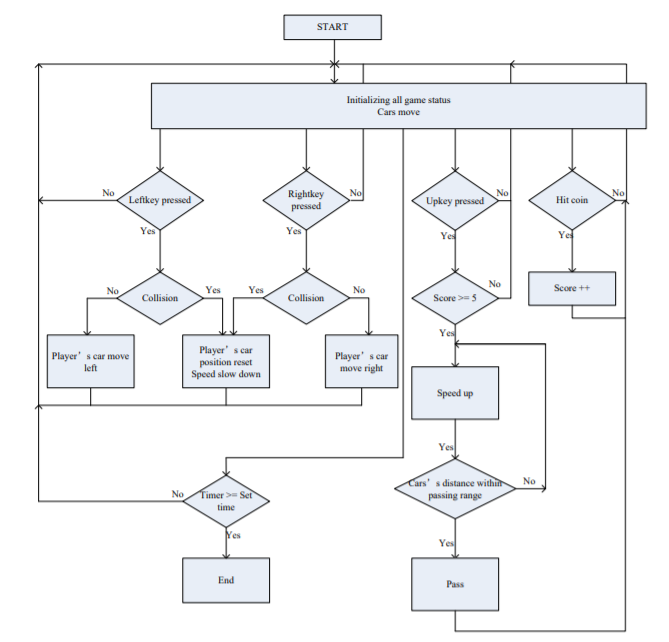
**SYSTEM DESIGN AND IMPLEMENTATION**

**4.1 Module description**

For this Car Racing Game, we would like to accomplish a video game imitating the existing game with a projective view. The theme of our game is to compete with an opponent that is controlled by computer in a racing tournament, the player’s goal is to get to the destination as soon as possible while trying to avoid bumping into the other car or road object, the final score will be posted according to the finishing position, numbers of bumps and the time.

This game consists of three major modules. First part is the Game Logic Generator which calculate the logic of this game, such as to detect bumps to obstacle, speed control based on keyboard input, opponents control and road generation, and this module is based on software. The second part is the Screen Rendering Module, we adopt the Sprite Graphics technique to decompose the display screen into 7 layers, which will be explained in details in later section. The last part is the Audio Module which generate the proper sound under the control of game logic.

**4.1.1 Module 1 – Game logic module**



*Figure 2*

When the game starts, a traffic light will change from red to green signalling the start of the game and all cars will start going forward. The other two opponent cars are set to have higher speeds than the player’s car and one of them is the fastest. The detection of the USB keyboard is in a separate thread and the system will be detecting if a key is pressed all the time. When the left key is pressed, the player’s car will be moving towards left. When the right key is pressed, the player’s car will be moving towards right. If the player’s car collides with the road or other cars, its position will be reset to the middle of the road and its speed will be slowed down. If the player’s car hits a coin, the score will be added up. When the score is larger than or equal to 5 and the up key is pressed, the player’s car will speed up. If the distance between the player’s car and the opponent cars is within the passing range and the player’s car is still speeding up, the player’s car will be passing the opponent cars. If the player’s car is not speeding up and the distance is within passing range, the player’s car will be passed by the opponent cars.

**4.1.2 Module 2 – Sprite control module**

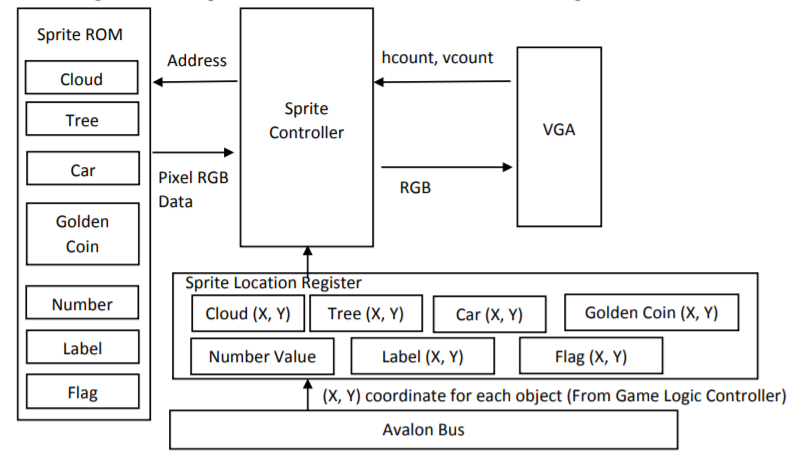
**Sprite**

The core of our video game display technology is Sprite, which utilizes small size of different elements to produce repeatedly pattern of pictures, minimizing the requirement for large screen pixels RGB information. The Sprite use multiple layers to overlap the pictures and generate a movement effect by control the movement of small elements in different layers, by this way obtain a video game display function.

The Sprite Control Module’s job is to generate the 640\*480 pixels RGB value for VGA Controller based on the control signal provided by the Game Logic Module. The Game Logic Module provides the X and Y coordinates for different objects, like trees, clouds, cars, scores, speed, which is also called Sprites. These Sprites are like label pictures store in ROM, and can be pasted to the screen according the X and Y coordinates. We list Sprites we will use in our games.

**Sprite Controller**

The Sprite Controller Module merges the different Sprites to generate the final RGB information for displaying on the screen. On the Linux side, the software generates the coordinate (X, Y) of different Sprites and passes coordinates to the Avalon Bus through the Kernel Driver. Then these data accompanied with address information in update the register file related to the specific Spite, the Sprite Controller fetches the RGB information from specific Sprite picture ROM, which finally merges all the pictures according to the priority and send to VGA module. The picture is finally showed on the screen. To make the Sprite move, the software just update the (X, Y) coordinate register file continuously.

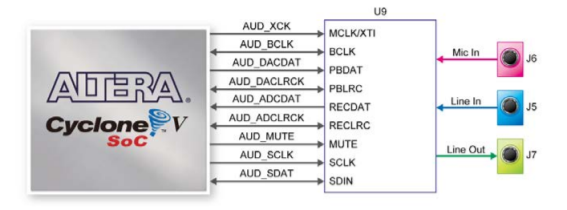


*Figure 3*

**4.1.3 Module 3 – Audio module**

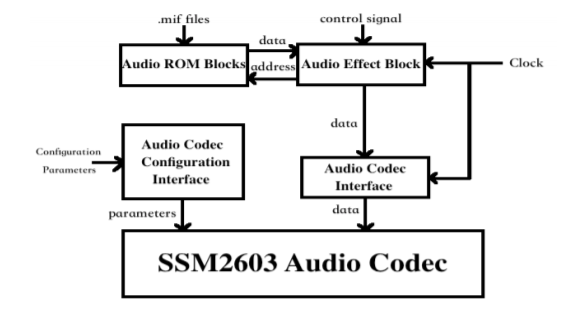
In the audio design, one background music, and two sound effects namely acceleration and crashing are included in this project. All the three audio files are configured with sample rate of 44100 Hz and 16 bits quantization to ensure the sound quality. In order to store the sound files in the on-chip ROM, we have to prepare MIF files with sizes small enough so that the audio files do not consume too much memory of the on-chip ROM.

Audio Controller:The Socket board will provides the Analog Devices SSM2603 Audio Codec (Encoder/Decoder). This chip supports microphone-in, line-in, and line-out ports, with a sample rate adjustable from 8kHz to 96kHz. The SSM2603 is controlled via a serial I 2C bus interface, which is connected to pins on the Cyclone-V SoC FPGA. The schematic diagram of the audio circuitry is shown in the figure below.



*Figure 4*

In this project, the Audio Controller is mainly consisted of four components, namely Audio ROM blocks, Audio Effect block, Audio Codec Interface, and Audio Codec Configuration Interface. The figure below is the function block diagram of the Audio Controller.



*Figure 5*

Audio ROM blocks: The audio data are well prepared in advance to take as little on-chip ROM space as possible, while preserving the sound quality.

Audio Effect block: This block operates at 50 MHz clock rate and receives the control signals passed from the software. When a valid control signal is passed to this block, the corresponding audio data will be fetched from the Audio Data blocks, and then passed to the SSM2603 Audio Codec through Audio Codec Interface.

Audio Codec Interface: This interface operates at 50 MHz clock rate. It sends the audio data to the SSM2603 Audio Codec using shift registers.

Background music and sound effects can be played concurrently. A circular queue can be designed to handle the background music only. And a mixer can be designed to combine the sound effects in circular queue and FIFO, and then send the mixed data to SSM2603 Audio Codec in order to play the background music and the sound effects at the same time.

**4.2 Module work flow**

This description of a general workflow focuses on the different steps of creating 3D game car and racing track using traditional tools and techniques commonly used in the game industry, and is described from the point of view of an asset production pipeline in a game studio where different steps are done. However, this is not the definitive workflow for creating 3D game characters, but just one example of how the process could be executed. For example, in a small independent game studio all of the following steps could be performed by a single artist, instead of several specialized artists dedicated to each different task. In addition to describing the different steps in the workflow of creating a 3D game character, some of the technical concepts related to the process will also be explained, as well as how artists need to take them into consideration in their work.

**4.3 Coding**

**Wheels**

Vehicle4W->WheelSetups[0].WheelClass = Ucar\_racingWheelFront::StaticClass();

Vehicle4W->WheelSetups[0].BoneName = FName("PhysWheel\_FL");

Vehicle4W->WheelSetups[0].AdditionalOffset = FVector(0.f, -8.f, 0.f);

Vehicle4W->WheelSetups[1].WheelClass = Ucar\_racingWheelFront::StaticClass();

Vehicle4W->WheelSetups[1].BoneName = FName("PhysWheel\_FR");

Vehicle4W->WheelSetups[1].AdditionalOffset = FVector(0.f, 8.f, 0.f);

Vehicle4W->WheelSetups[2].WheelClass = Ucar\_racingWheelRear::StaticClass();

Vehicle4W->WheelSetups[2].BoneName = FName("PhysWheel\_BL");

Vehicle4W->WheelSetups[2].AdditionalOffset = FVector(0.f, -8.f, 0.f);

Vehicle4W->WheelSetups[3].WheelClass = Ucar\_racingWheelRear::StaticClass();

Vehicle4W->WheelSetups[3].BoneName = FName("PhysWheel\_BR");

Vehicle4W->WheelSetups[3].AdditionalOffset = FVector(0.f, 8.f, 0.f);

// Adjust the tire loading

Vehicle4W->MinNormalizedTireLoad = 0.0f;

Vehicle4W->MinNormalizedTireLoadFiltered = 0.2f;

Vehicle4W->MaxNormalizedTireLoad = 2.0f;

Vehicle4W->MaxNormalizedTireLoadFiltered = 2.0f;

**Engine**

Vehicle4W->MaxEngineRPM = 5700.0f;

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

Vehicle4W->EngineSetup.TorqueCurve.GetRichCurve()->AddKey(0.0f, 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);

// Gears

// For Four Wheel drive

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

// Drive the front wheels a little more than the rear

Vehicle4W->DifferentialSetup.FrontRearSplit = 0.65;

// Automatic gear changing

Vehicle4W->TransmissionSetup.bUseGearAutoBox = true;

Vehicle4W->TransmissionSetup.GearSwitchTime = 0.15f;

Vehicle4W->TransmissionSetup.GearAutoBoxLatency = 1.0f;

#include "CoreMinimal.h"

#include "WheeledVehicle.h"

#include "game\_developmentPawn.generated.h"

class UPhysicalMaterial;

class UCameraComponent;

class USpringArmComponent;

class UTextRenderComponent;

class UInputComponent;

class UAudioComponent;

UCLASS(config=Game)

class Agame\_developmentPawn : public AWheeledVehicle

{

GENERATED\_BODY()

UPROPERTY(Category = Camera, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

USpringArmComponent\* SpringArm;

UPROPERTY(Category = Camera, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

UCameraComponent\* Camera;

UPROPERTY(Category = Camera, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

class USceneComponent\* InternalCameraBase;

UPROPERTY(Category = Camera, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

UCameraComponent\* InternalCamera;

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

UTextRenderComponent\* InCarSpeed;

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

UTextRenderComponent\* InCarGear;

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

UAudioComponent\* EngineSoundComponent;

public:

Agame\_developmentPawn();

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly)

FText SpeedDisplayString;

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly)

FText GearDisplayString;

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly)

FColor GearDisplayColor;

UPROPERTY(Category = Display, VisibleDefaultsOnly, BlueprintReadOnly)

FColor GearDisplayReverseColor;

UPROPERTY(Category = Camera, VisibleDefaultsOnly, BlueprintReadOnly)

bool bInCarCameraActive;

UPROPERTY(Category = Camera, VisibleDefaultsOnly, BlueprintReadOnly)

bool bInReverseGear;

FVector InternalCameraOrigin;

virtual void SetupPlayerInputComponent(UInputComponent\* InputComponent) override;

virtual void Tick(float Delta) override;

protected:

virtual void BeginPlay() override;

public:

void MoveForward(float Val);

void SetupInCarHUD();

void UpdatePhysicsMaterial();

/\*\* Handle pressing right \*/

void MoveRight(float Val);

/\*\* Handle handbrake pressed \*/

void OnHandbrakePressed();

/\*\* Handle handbrake released \*/

void OnHandbrakeReleased();

/\*\* Switch between cameras \*/

void OnToggleCamera();

static const FName LookUpBinding;

static const FName LookRightBinding;

static const FName EngineAudioRPM;

private:

void EnableIncarView( const bool bState );

void UpdateHUDStrings();

bool bIsLowFriction;

UPhysicalMaterial\* SlipperyMaterial;

UPhysicalMaterial\* NonSlipperyMaterial;

public:

FORCEINLINE USpringArmComponent\* GetSpringArm() const { return SpringArm; }

FORCEINLINE UCameraComponent\* GetCamera() const { return Camera; }

FORCEINLINE UCameraComponent\* GetInternalCamera() const { return InternalCamera; }

FORCEINLINE UTextRenderComponent\* GetInCarSpeed() const { return InCarSpeed; }

FORCEINLINE UTextRenderComponent\* GetInCarGear() const { return InCarGear; }

FORCEINLINE UAudioComponent\* GetEngineSoundComponent() const { return EngineSoundComponent; }

};

#include "game\_developmentHud.h"

#include "game\_developmentPawn.h"

#include "Engine/Canvas.h"

#include "Engine/Font.h"

#include "CanvasItem.h"

#include "UObject/ConstructorHelpers.h"

#include "Engine/Engine.h"

#define LOCTEXT\_NAMESPACE "VehicleHUD"

#ifndef HMD\_MODULE\_INCLUDED

#define HMD\_MODULE\_INCLUDED 0

#endif

Agame\_developmentHud::Agame\_developmentHud()

{

static ConstructorHelpers::FObjectFinder<UFont> Font(TEXT("/Engine/EngineFonts/RobotoDistanceField"));

HUDFont = Font.Object;

}

void Agame\_developmentHud::DrawHUD()

{

Super::DrawHUD();

const float HUDXRatio = Canvas->SizeX / 1280.f;

const float HUDYRatio = Canvas->SizeY / 720.f;

bool bHMDDeviceActive = false;

#if HMD\_MODULE\_INCLUDED

bHMDDeviceActive = GEngine->IsStereoscopic3D();

#endif // HMD\_MODULE\_INCLUDED

if( bHMDDeviceActive == false )

{

Agame\_developmentPawn\* Vehicle = Cast<Agame\_developmentPawn>(GetOwningPawn());

if ((Vehicle != nullptr) && (Vehicle->bInCarCameraActive == false))

{

FVector2D ScaleVec(HUDYRatio \* 1.4f, HUDYRatio \* 1.4f);

// Speed

FCanvasTextItem SpeedTextItem(FVector2D(HUDXRatio \* 805.f, HUDYRatio \* 455), Vehicle->SpeedDisplayString, HUDFont, FLinearColor::White);

SpeedTextItem.Scale = ScaleVec;

Canvas->DrawItem(SpeedTextItem);

// Gear

FCanvasTextItem GearTextItem(FVector2D(HUDXRatio \* 805.f, HUDYRatio \* 500.f), Vehicle->GearDisplayString, HUDFont, Vehicle->bInReverseGear == false ? Vehicle->GearDisplayColor : Vehicle->GearDisplayReverseColor);

GearTextItem.Scale = ScaleVec;

Canvas->DrawItem(GearTextItem);

}

}

}

#include "CoreMinimal.h"

#include "GameFramework/HUD.h"

#include "game\_developmentHud.generated.h"

UCLASS(config = Game)

class Agame\_developmentHud : public AHUD

{

GENERATED\_BODY()

public:

Agame\_developmentHud();

};

**Performance Analysis**

**5.1 Performance Measures**

In the world of gaming, there is no doubt that the performance of any game matters a lot. Here are some of the measures upon which the performance of the game can be analysed:

* + **Frame Rendering -** Frame Rendering is one of the most critical performance metrics because it measures what the viewer sees while playing your game. It is measured using several aspects of Frames Per Second or FPS. Specifically, Median FPS, FPS Stability, and Variability Index. 30fps is widely considered a minimum for smooth playability, while 60fps is considered a more playable target.
  + **CPU Usage -** CPU usage measures the average load on the CPU across the whole recorded session. High average usage is a predictor of high-power consumption and thermal throttling, which can really hurt user experience. What constitutes “high usage” really depends on the nature of the game and the device it’s running on.
  + **GPU Usage -** This is an essential predictor of user experience problems in games because it measures the demand that a game’s software makes on the graphical processor. Excessive demands can lead to noticeable problems like unstable frame rates, micro-stutters, and high-power consumption.
  + **RAM Usage -** RAM usage is a vital measure for evaluating the performance of the game. It is the amount of memory that is being used while playing the game. Excessive usage of ram causes stuttering in the game.
  + **Network Usage -** Nowadays, almost every game relies on an internet connection, so network usage has become a key metric when evaluating performance and power consumption. High network usage can cause slowdowns, prolonged wait times, unexpected data plan usage, and excessive CPU and battery usage.

**5.2 Performance Analysis**

* + - Frame Rendering
      * Average frames per second: 21fps
      * Highest frames per second: 33 fps
      * Lowest frames per second: 11fps
    - CPU Usage
      * Highest CPU usage: 100%
      * Lowest CPU usage: 92%it
      * Average CPU Usage: 95%
    - GPU Usage
      * Highest GPU usage: 45%
      * Lowest GPU usage: 32%
      * Average GPU usage 38.5%
    - Memory Required
      * Minimum 8 GB
      * Recommended 16 GB
    - Network Usage
      * Highest network usage:0%
      * Lowest network usage:0%
      * Average network usage:0%

**FUTURE DEVELOPMENT PLANS AND CONCLUSION**

**6.1 Limitations of the Program**

* 1. Platform Limitations: Currently the game only runs on windows machines with no support for mobile device and Linux operating systems.
  2. Technical Bugs: The physics engine on which the game’s physics is based on fails to initialize under certain circumstances.
  3. Game Breaking Bugs: Due to severe time constraint the Artificial intelligence of the game is not properly completed hence, it fails to initialize most of the time.
  4. Lack of training for the AI: Due to time constraints and lack of game-play experience the AI of the game is not trained which creates un-explainable bugs.

**6.2 Future Plans**

Future work would involve incorporating other path finding techniques into the existing game system so that it can be more exciting. The waypoint system can be used to support a different implementation of the search for the race car’s path such that it is more dynamically determined based on its current position. This would make it more difficult for the human players to predict the behaviour of the game-controlled cars. This unpredictable behaviour would mean a more challenging gameplay for the user. Furthermore, while this game was designed for entertainment, the software can be extended to become a driver simulator for educational purposes. By modifying some of the parameter values of the car, such as the braking coefficients and torque values, and by modifying the game environment to resemble that of a municipal road system, Racer can be adapted to become a simulation software for learner-level drivers or for familiarizing drivers with the roadway layouts and traffic laws of foreign or unknown cities.

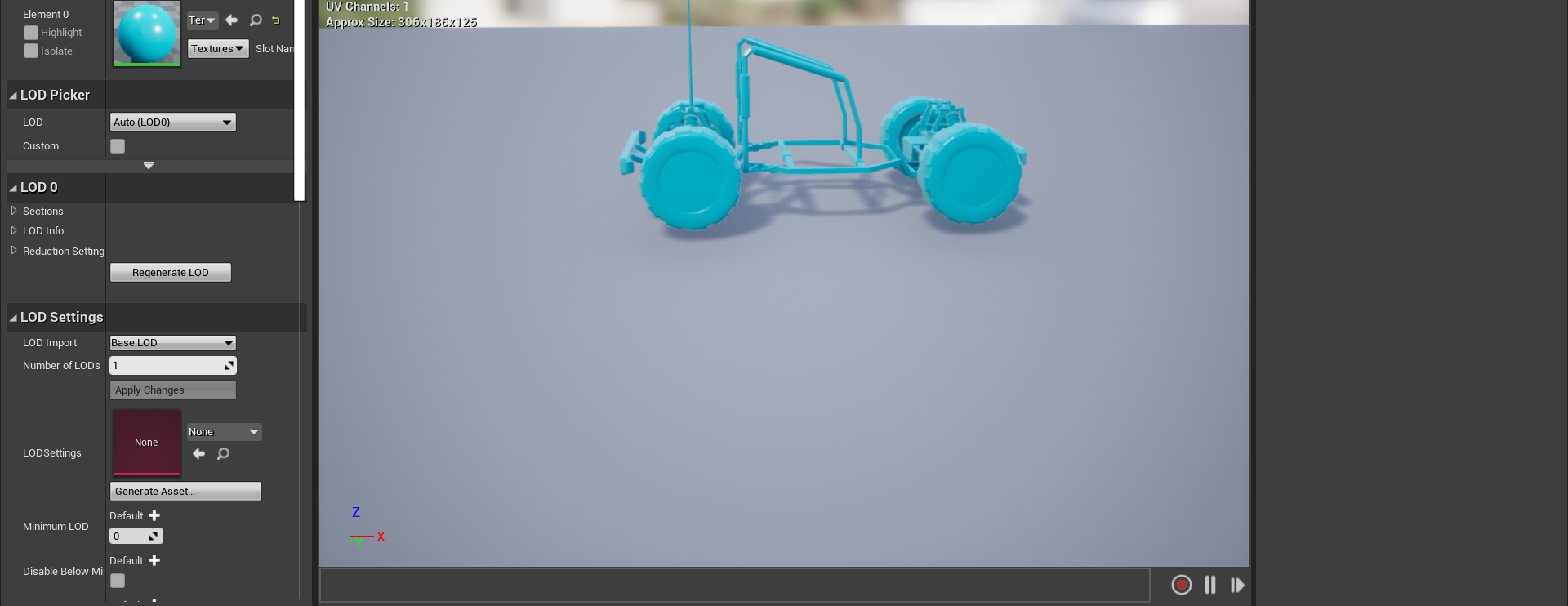
**6.3 Conclusion**

From this project, we improved our skills in programming both hardware language and software language. In the Sprite Display Module design, we have gone through programming, testing, modifying and assembling. We adopt Top-Down Design Methodology, divide the big project into several small task and modules, design general port interface. By this way, we improve our progress and make our project easy to locate where the problems are because we use small modules to limit the bug region. We believe the most valuable lesson we have learned in this project is how to debug in hardware and how to locate the bug. The bugs and problems we encountered through this project help us to enhance our knowledge of hardware design. During the audio design in this project, we have learnt that the physical on-chip ROM limit should always be kept in mind during hardware design, and all the data designed to be loaded on board should be modified to the correct format beforehand. When preparing the proper audio file, rather than repeatedly loading different testing audio files into the system and compiling the whole program every time we change the audio files, we can simulate the looping playback environment using MATLAB, and then load the qualified audio file version on board for further testing. The simulation strategy is much more efficient. Before integrating the audio module into the whole game logic program, we should test the audio module by sending the expected signals to examine whether the audio system can output correctly. Following such procedure, we can save a lot of debugging, because we have divided the whole design and testing procedure into multiple subsections which make it easier and faster to locate the error locations. We used the software to implement the game logic and hardware to display graphics and audio sounds and communicated each other with software-hardware interface. During this procedure, we’ve encountered a lot of problems in implementing the game logic and keyboard control but we were finally able to solve all of them though we still have a lot to improve.

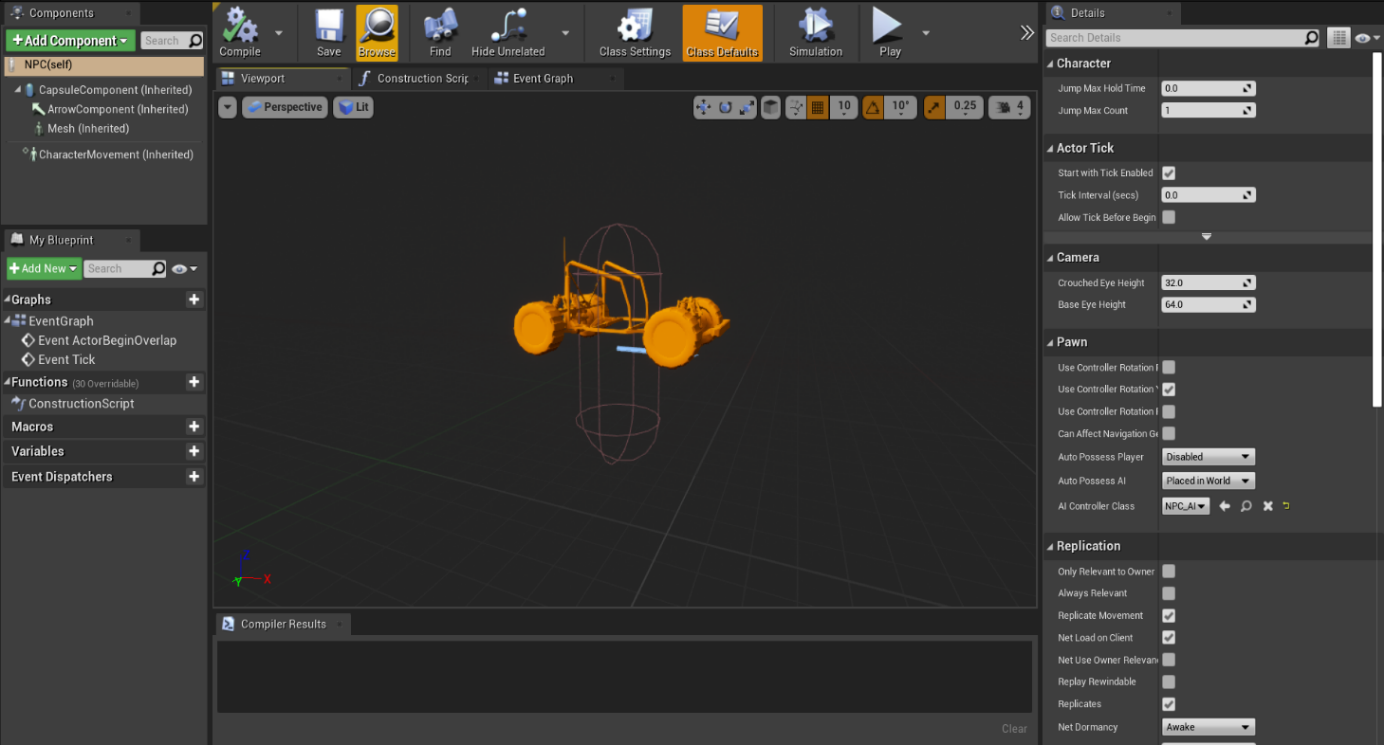
**BASIC LOOK OF THE GAME**

**7.1 Snapshots of the Game**

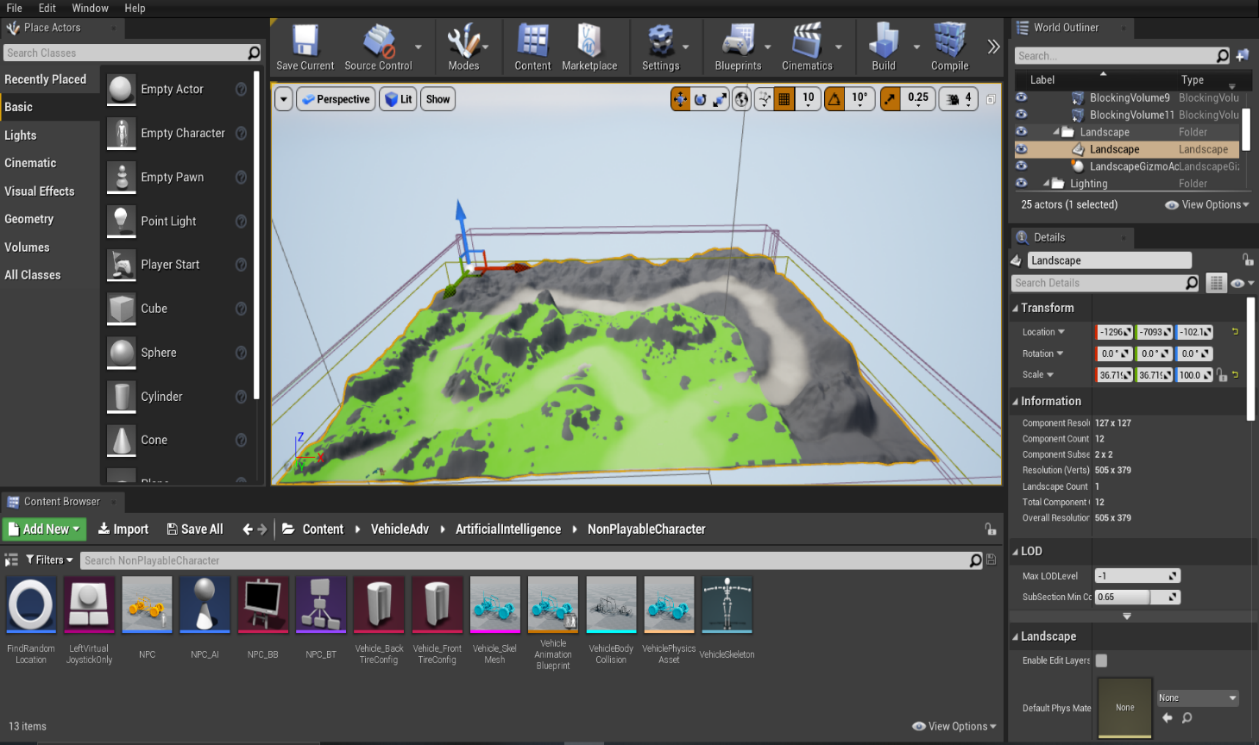
* 1. **Player Car Model:**



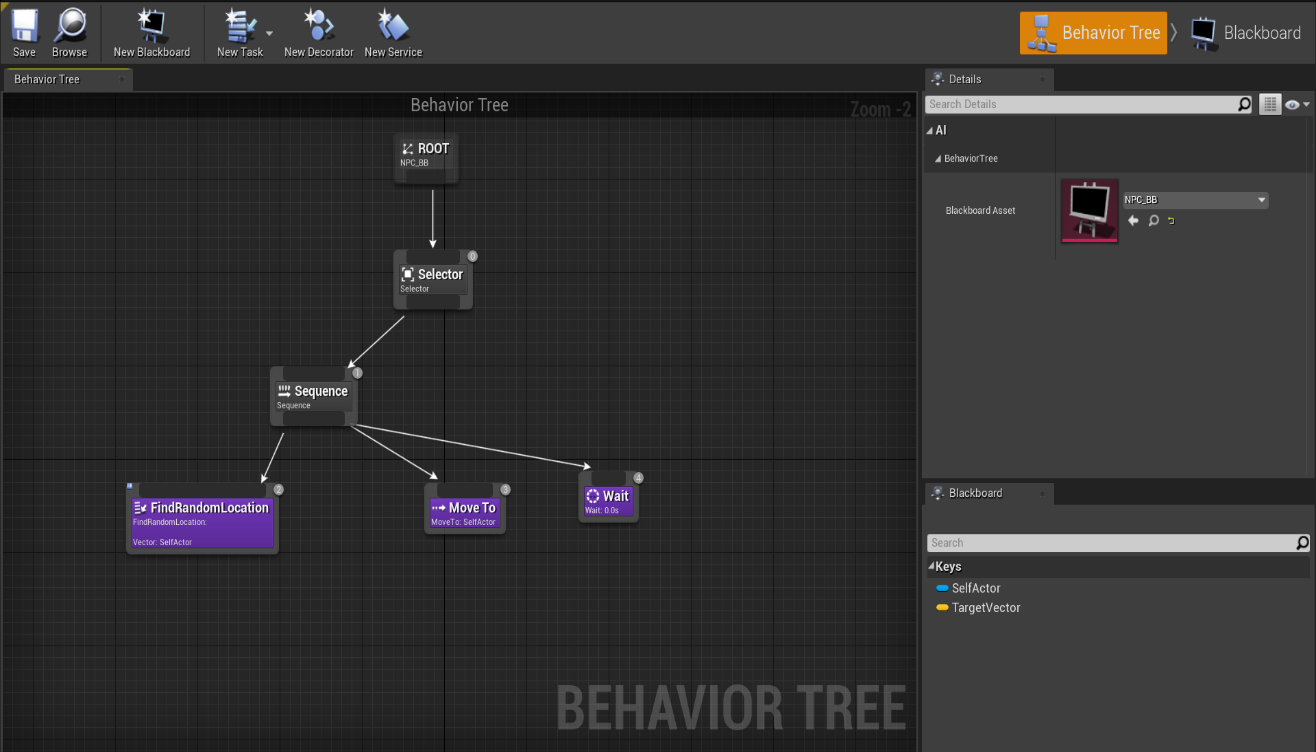
* 1. **NPC (Non-Playable Character)**



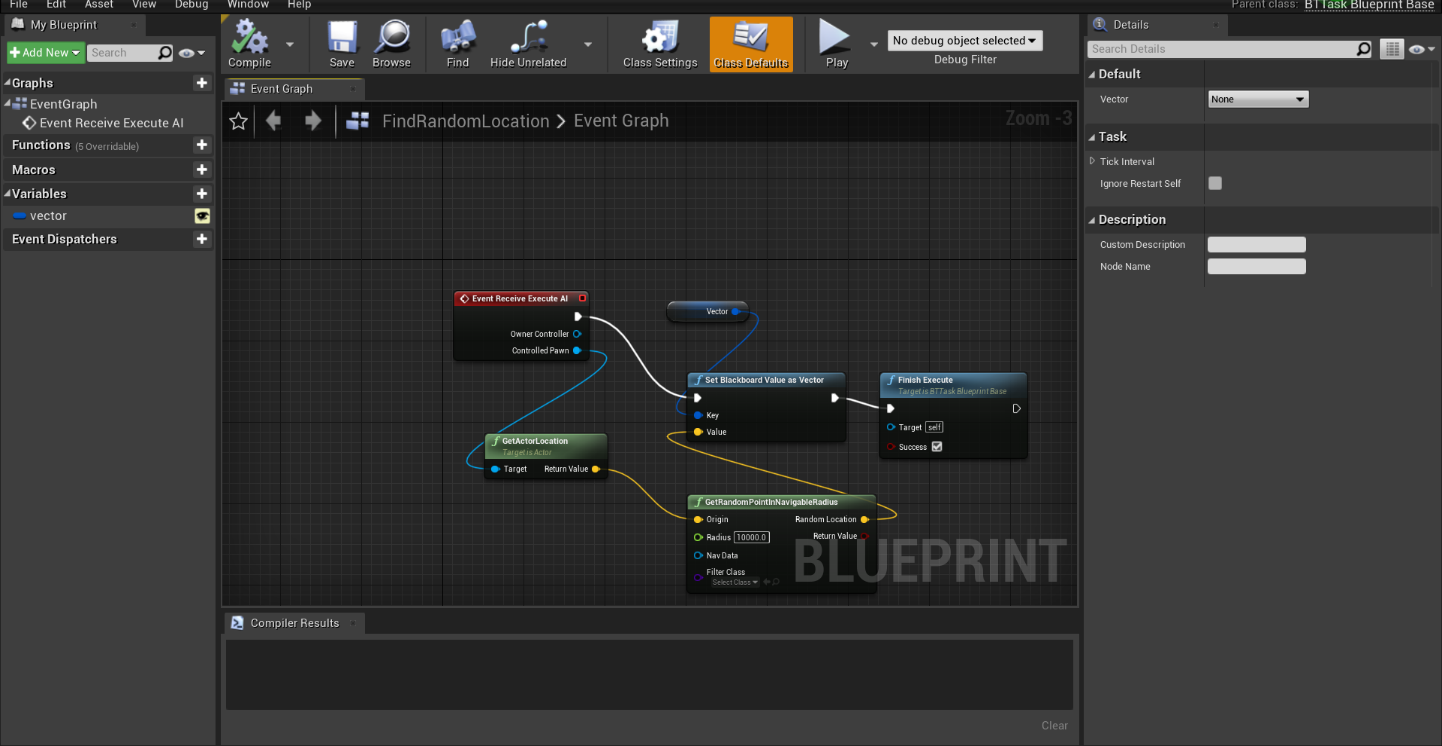
* 1. **Navigation Mesh**



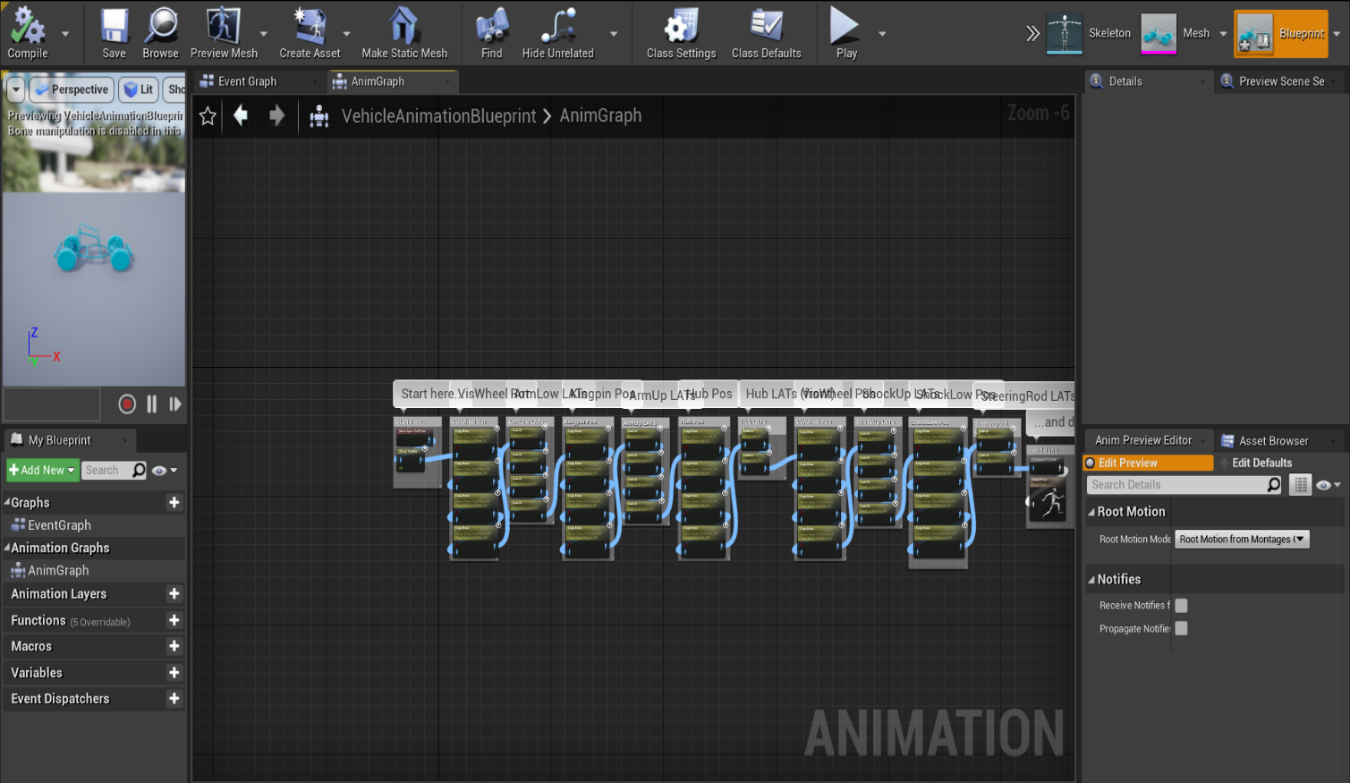
* 1. **Behaviour Tree**



* 1. **Behavior Tree (Part 2)**



* 1. **Car Model Animation**



**REFERENCE**

* https://www.cs.columbia.edu/~sedwards/classes/2015/4840/reports/racing.pdf
* https://docs.unrealengine.com/en-US/GettingStarted/RecommendedSpecifications/index.html
* https://www.google.com/amp/s/blog.gamebench.net/game-performance-metrics-that-matters%3fhs\_amp=true
* https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.measuringbehavior.org/files/2012/ProceedingsPDF(website)/Special%2520Sessions/Measuring%2520Driver%2520and%2520Pilot%2520Behavior/de\_Winter\_et\_al\_MB2012.pdf&ved=2ahUKEwi8-\_fy89nsAhVv7HMBHQLIBwYQFjAAegQIAhAB&usg=AOvVaw0fhGypLJ5xVqc97rvZ3ixl
* https://www.google.com/amp/s/www.newgenapps.com/blog/unreal-engine-reveiw-pros-cons-suitability/%3famp