Version 1.1 2024/25

University of Staffordshire

Speedway Frenzy

Technical Design Document

PIERETTI, Adam

B00007I@student.staffs.ac.uk

Contents

Project Introduction

Project Goals

For my project I am looking into creating a hover car racing game prototype. Overall the level will include all the aspects of a normal racing game in some manner with an emphasis on a futuristic world.

Challenges and Risks

Working on this project will primarily need me to retrain myself to create Unreal Engine levels with a different goal in mind than the third-person character games I'm accustomed to. Vehicle physics are different to that of Characters which will mean that there will be high risks that the vehicle will not move perfectly by the deadline meaning it will be a tough challenge.

Hardware Requirements

The hardware in use must be able to run at 120 HZ. This is because the game has a large amount of sections that cause lag and can make the game unplayable.

Platforms

Target Platform

Currently the project is only being designed for PC as the specifications are the easiest and most transferrable if I decide to take things any further. There are inputs made for default controllers as well.

Engine Specific Specifications and Limitations

Preferred memory space - 32GB. Minimum memory requirements - 16GB.

Engine Summary

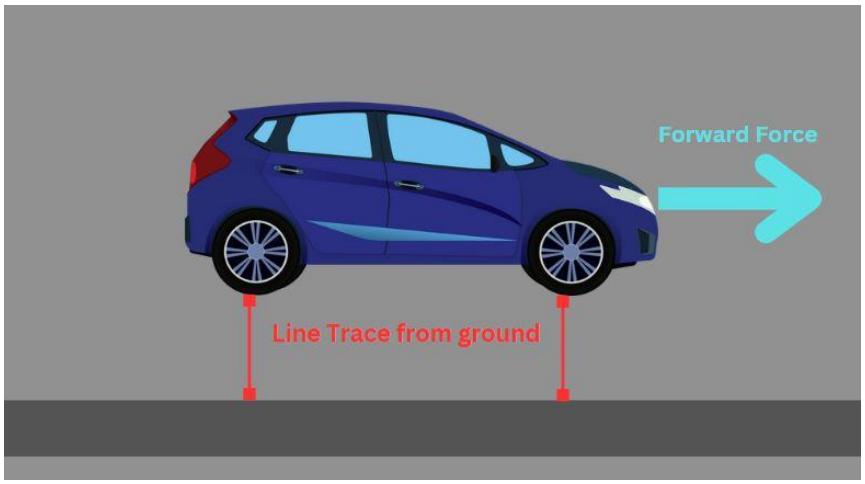
Unreal Engine 5.4 will be the version in use for the entirety of the project. Currently there are no plugins that are being used in the projects development. Datasmith FBX importer will also be in use so that models from Sketchfab can be imported.

Systems and Diagrams

System 1

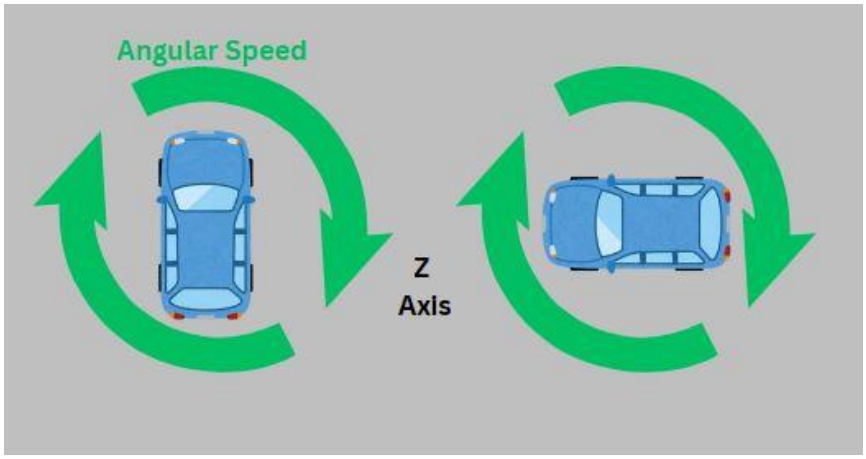
Hover car movement

Hover component



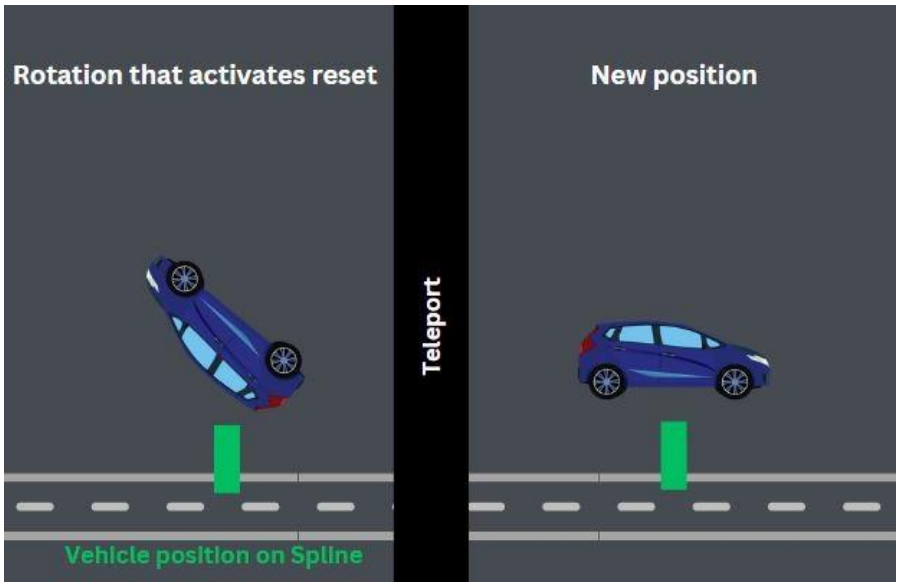
To simply explain the car uses a line trace at the location of the component and draws a line using the get up vector that is equal to the hover distance variable (I set mine to 250). There is a forward force the car can use to move depending where it spawns.

Steering



In order for the car to turn and move it needs to know how fast it should be turning. This is where the Angular Speed variable comes into effect. By setting this in Torque degrees node the car is able to act similar to how you would set regular speed by using a set Current and Target speed variables.

Reset position

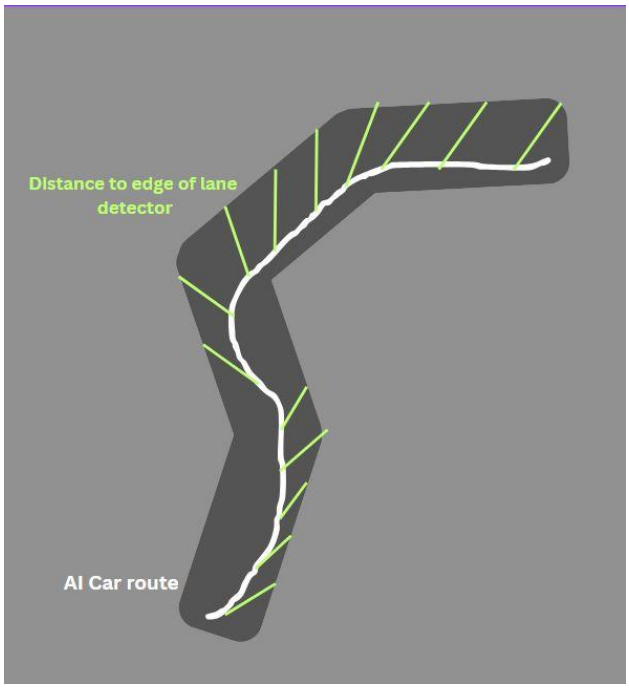


The reset function will only trigger under the condition that the vehicle has flipped too much on either the X or Y axis as the vehicle should only be rotating on the Z axis. There are some location in the game where the race track goes upwards so the angle that would trigger the reset needs to be at lowest higher than the angle of the hill locations.

System 2

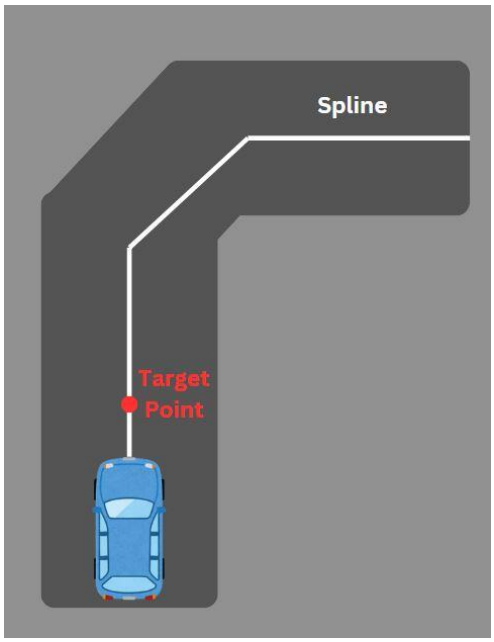
AI cars

Road edge detection



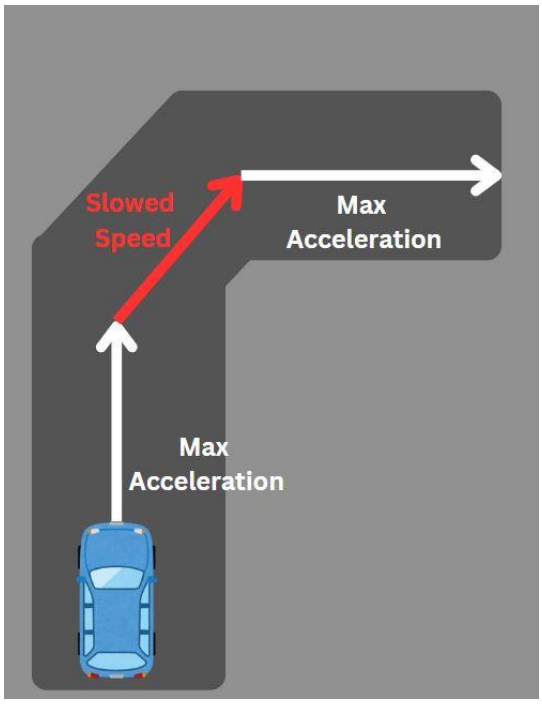
Upon spawing the AI car traces the edge of the roads distance from the spline that should be placed in the middle of the road. Once this has been done the car knows where the roads edge is and will continuously trace to the edge to make sure the car is staying on the spline. Should it need to the steering contriols will be used to turn the car.

Target point



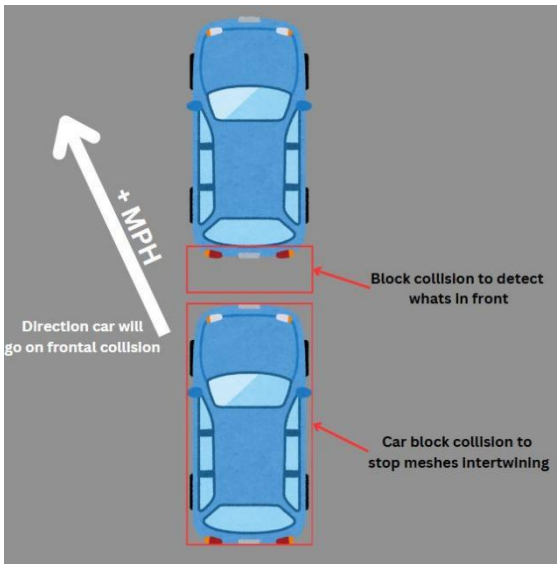
The Target Point was the most heavily changing set of nodes as it including new functionality that needed to be included when vehicle collisions became an issue in racing. The Target Point is the point that is set ahead of the car and is consistently trying to stay on the spline. By creating a variable of the Spline Distance the Target point can use the Right Vector and Forward Vector to determine if it is on the spline or not.

Speed Control



The speed adjustment blueprint is more complex than speed setup. However, after watching a video that went more in depth into the AI setup I understood how it works quite well. In the AI car there is a vector variable created that calculates a target point a little ahead of the car. When the spline reference is set in the AI's begin play section the car knows it has to follow this track. The Target point is there to detect when the car leaves the spline. It has different functionality in other parts of the blueprint which I will talk about later, however here all it does is slows the car when it is told the car should be turning on the Z Axis. Without the steering mechanics implemented this has no effect.

Overtake



To stop crashes for the most part I needed to create two collision boxes. Both have different booleans that will trigger when overlapped. To stop incoming collisions there is a collider on the front which will detect if anything is too close to the front and indicate for the car to either slow down or reverse on a range of 1 to -1. The overtake boolean is set in the speed blueprint and is multiplied by whatever the car deems the speed should be as an extra condition. Lastly in the Target point blueprint there is an extra condition set with all the different conditions added together. These conditions select floats that indicate that the target point must change position so that it is not inside any object that the car is trying to avoid, therefore forcing the car to change trajectory.

System 3

Race Manager

Leaderboard

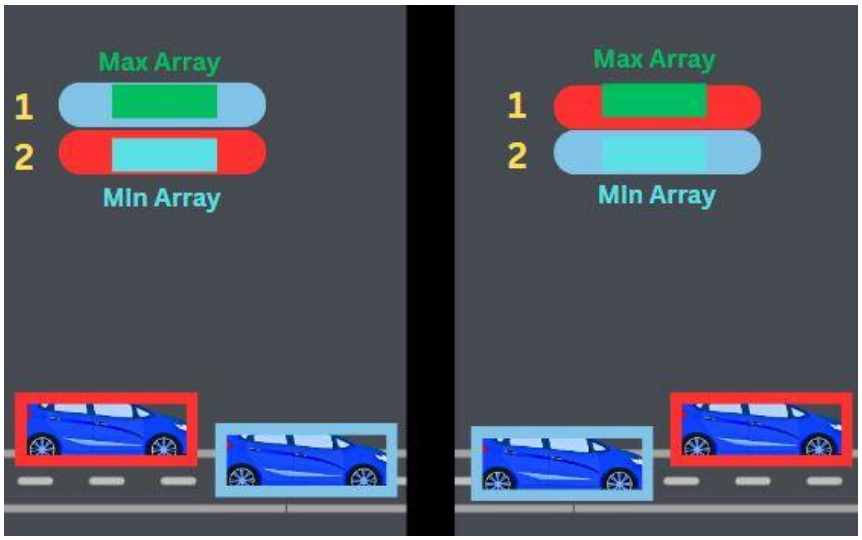
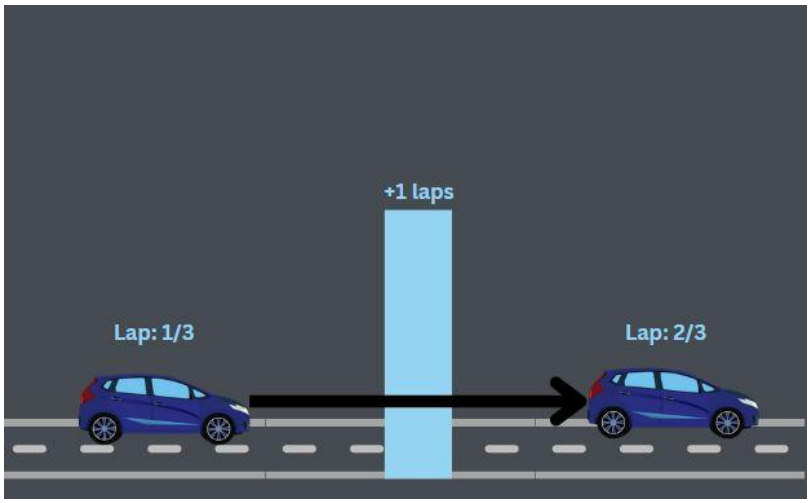


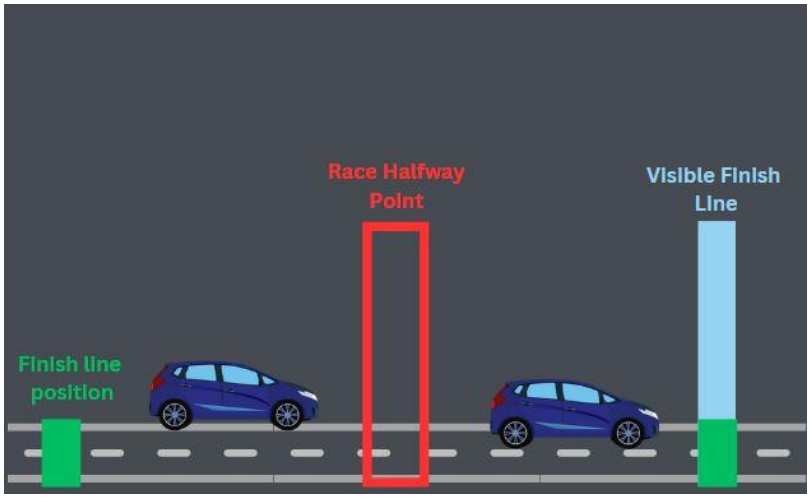
Table representing car on spawn

Car	**Progress Number**
Player pawn	5.5435
AI Car 1	5.4211
AI Car 2	5.4201
AI Car 3	5.3222
AI Car 4	5.2113
AI Car 5	5.1167

The leaderboard works by getting all cars on an array from the parent pawn which in this case is the player pawn. Once this is done the gamemode will sort all the cars into a position based on their placement on spawn. This is calculated in the pawns blueprint and will be applied based on the forward facing position on the map. Once complete the car with the maximum integer (based on the spline progress number) will be placed highest as they will be positioned the furthest forward and the car with the lowest will be placed in last on the widget. This will be repeated on tick until the race is over. The condition that a car has more laps can be taken from the Finish Line blueprint so that should cars lap eachother the one that's in first remains there.

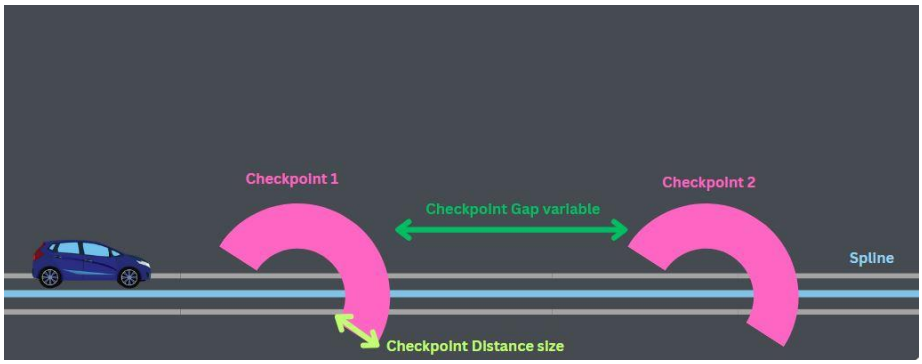
Finish line





The finish line works by giving adding 1 to the lap text in the UI every time the car overlaps with it. To stop the player from being able to constantly overlap with the Finish Line I opted to set remove the visibility of the overlapping actor until the car passes a checkpoint half way through the race track. The checkpoint is halfway in the track so that the player cannot by pass the course by using a short cut in the other direction. When the player overlaps with the checkpoint the actor becomes visible again and the player can add another lap to there score. The finish line is only projected to the player using the IsPlayer node which stops other AI cars from overlapping in a bad way.

Checkpoint placement

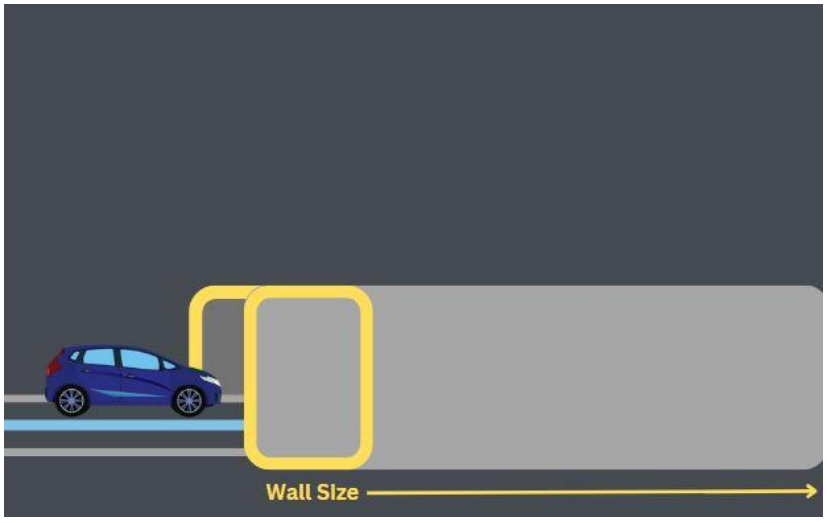


Individually placing checkpoints on the track in the perfect place would be tedious and somewhat messy compared to the more free form track creation. To bypass this I created a race manager in the level that could place actor referenced in it anywhere on the spline that makes up the race track. Placing the checkpoints meant creating an array for the checkpoint actor. Once this was done all the checkpoints would be spawned depending on two variables:

- The distance between the checkpoints.
- The width of the checkpoint.

Using an index that would place the nearest spot to the Race Manager on the spline the checkpoints would be placed with the array adding a checkpoint to the spawn index until the checkpoints finish a loop of the spline. This does mean there will be a randomly placed checkpoint near the starting one though this issue can be fixed by just deleting it as they are all independent actors when placed in the level. The function to do this needs to be called in editor or it cannot be used. To clear the checkpoints is far simpler, only requiring the looped actors and clearing them depending on if the function is activated in the place checkpoints function.

Wall placement



This mechanic was also how I made the walls for the level. The only difference in the function for the walls is that the distance between them is instead set to a far lower number (Not 0 as that causes crashes) which pairs the walls on top of each other on the spline in a similar way to how the checkpoints work. The boundaries are very important to my race track as the cars can quite easily fall off the map so need some limits.

Optimisation and Profiling

Profiling Systems

[What is your process for testing functionality?]

Trace debugs will be the main method to make sure all detection mechanics are working (specifically the hover car).

Profiling Graphics

[What is your process for testing graphics performance?]

Firstly size maps will be used to determine what waste is in the project files.

Profiling Network/Multiplayer (If Applicable)

[What is your process for testing functionality?]

…

Coding Standards

Style Guide

Commenting specific colours depending on the type of event will be the go to. (E.g. Purple for functions, Red for Custom events, etc). Another important aspect to attain a clean blueprint with be making sure the node connections are organised in a way so that those reading the blueprints can clearly see where they are being connected. This basically means no overlapping connection and the use of reroute nodes.

Moscow

Must	Should	Could	Won't
Working Hover Car (Custom Physics)	AI drivers for the player to race against.	More aesthetically pleasing atmosphere.	A playable character who can interact with the map
Racetrack for Hover Car to drive onand a rule system	Polished simulated physics.	A variety of different cars for the player to use.	A menu screen

Timeline

Task	Start Date	Completion Date
Hover Car	16/12/24	22/01/25
AI Car	06/01/25	31/01/25
Track Design	20/01/25	27/01/25
Asset allocation	06/01/25	17/01/25
UI Design	27/01/25	10/02/25