TRUST ELPLAN

SUPER SUPER MAX

PROTOTYPE

Rodrigo Juez Alejandro Benimeli

06/12/2021

content

Development Team

Members: Alejandro Benimeli Miranda and Rodrigo Juez Hernández

Overview of the Delivery

INTRODUCTION

The objective of this document is to show the progress of the SUPERSUPERMAX game in the form of a prototype.

GENERAL DELIVERY DESCRIPTION

We tried to include everything we have developed until the delivery date, as you will see we got to the final deadline with our objectives mostly complete. We also included an executable file along with the Unity project on moodle.

This delivery includes most of the basic functionality needed to do the game like physics, logic and AI scripts.

FUNCTIONALITY OF THE DELIVERY

The expected features for this prototype layed out in the design document were:

- Basic car features:
  - Engine
  - GearBox
  - Skid physics.
- Circuits
  - 1 complete circuit for time attack.
- Time Attack Mode.

Unfortunately we didn't complete every every expected requirement, on the other hand we did more advanced features that weren't due this delivery. We decided to change plans because we preferred to do more advanced and complex features first.

The functionality of the delivery included is:

- Car physics and logic.
- Game Manager logic.
- Basic AI.
- Basic menus and races.

ENVIRONMENT AND WORKFLOW TOOLS

The environment used to develop the game is the following:

- The game engine used is the ***Unity Engine***, as is specified in the assignment.
- Along with Unity, we use ***Visual Studio*** as our IDE because of its great compatibility with the Unity Engine.
- For version control we use a ***GitHub*** repository and for time and task management ***Trello***.
- For communication we used ***Discord*** as it's a great communication service to work at home.

TEAM AND WORKPLACE

Our team consists of two members, each of us worked on separate features, Rodrigo Juez Hernández worked on car physics, logic and AI as he understood more of torque curves and gear ratios, while Alejandro Benimeli Miranda worked more on 3D modelling the track, lobby, menus and events.

Our workplace was our own homes, we mainly worked separately until we had to integrate our features where we met on Discord and talked, we used github to register the changes we each did and to join our features, for example I wrote a lot of code so I only pushed ".cs" files while Alejandro mostly did interface, adding 3D models so he pushed the main project files.

ADDITIONAL RESOURCES

- https://assetstore.unity.com/packages/3d/environments/urban/low-poly-park-61922
- https://assetstore.unity.com/packages/3d/vehicles/land/stylized-free-drift-car-197718
- https://github.com/SebLague/Curve-Editor

PROJECT MANAGEMENT

We set weekly meetings to show each other what we developed during that week and assign tasks to be completed for the next week. Everyone was free to work whenever it was more convenient for them, as it was hard to find times where we both could work at the same time due to the coursework from other assignments.

In the first document we set some deadlines:

Description	Date
Basic car handling	11/11/2021
1 basic circuit for time attack	11/11/2021
Complete car handling	25/11/2021
1 complete circuit for time attack	25/11/2021
Delivery of the Prototype	06/12/2021
Lobby Map	13/12/2021
Map for pursuit mode	13/12/2021
Add a different car skin	20/12/2021
Basic AI for pursuit mode	28/12/2021
Delivery of the Complete Game	11/01/2022

However, we had to do some adjustments that we felt necessary to accomplish as much as we could for the prototype deadline.

We decided to change plans because we considered that it was better to do more complex and advanced features first, when we could work more time together and leave simple and repetitive tasks (like decorating circuits, or doing new circuits) for Christmas.

Changes:

- We made the lobby, which was supposed to be after the deadline, because without it there was no way to move between game modes (except going in the editor and changing scenes). Instead of wasting time implementing a temporary way of moving between them that we would then discard once the lobby was done, we thought it'd be better if we made it plus some "portals" to start a game mode.
- We decided to leave the circuit in a basic state and focus on improving the Time Attack game mode and including one that wasn't on the original plan: Race. This new mode shares tracks with Time Attack and includes AI racers.

DESIGN/ANALYSIS

In the original design document we specified 2 modules for Car and Camera, one for Game Manager and two for Tracks and Lobby. The truth is we weren't Unity savvy when we designed the document so we have changed the modules and design since.

- Car Logic
  - CarController
  - SmokeSystem
- UI and HUD
  - GameManager
  - MenuController
  - MenuTrigger
  - FollowingCameraController
- Events and Gamemodes
  - Race
  - TimeAttack
- Checkpoints
  - CheckPoint
  - TrackCheckpoints
- AI
  - trackWaypoints
  - inputManager

Car Logic

- **CarController:** The car gets its orders from the inputManager class, this is to abstract the control system (AI controls, keyboard controls, mobile controls…), then we generated a function for each simulated part of the car:
  - **calculateEnginePower**: updates the RPM to which the engine is running at, based on input of accelerator.
  - **shifter:** Reads if the user wants to change gear, or calculates the correct gear if it's in automatic.
  - **activateNitrous:** does some checking to see if you can activate nitrus, adds force and sets a flag to show the reaction in the UI.
  - **Brake:** gets the input and slows down the car.
  - **Accelerate:** gets the engineRPM and transmits it to the wheels (the rear ones or all wheels depending on the setting of the car).
  - **Steer:** based on the steer input calculate the turning radius of the wheels and apply ackerman Steering to improve handling.

    - **addDownForce:** adds force downwards the faster the car goes to improve handling at higher speeds.
    - **updateWheelPoses:** gets the wheel turn angle and updates the 3d model rotation.
- **SmokeSystem:** gets the Smoke GameObject and sets one on each wheel, only activates the smoke when the slip angle is higher than the slip limit we set on the inspector, this way the user knows when the car is doing burnouts (when accelerating) or starting to drift.

UI and HUD

We changed the UI from the design document, because when we tried to play the game with the original design (fixed rotation) it was quite difficult so we opted for a smooth rotation. Plus the Speedometer was changed for a tachometer to show the engine RPMs and gear.

- **GameManager**: Updates the interface based on the speed, gear, nitro and RPM needle based on flags and variables public in the CarController component.
- **MenuController**: It's a script that holds a function that receives a scene name and changes the scene to it. It's intended to be called by the different buttons in the map selection menus.
- **MenuTrigger**: Detects when a vehicle goes through the object this script is added too and changes the scene to a map selection menu. The game mode depends on which portal is triggered.
- **FollowingCameraController**: Makes the camera follow the player from behind at a certain distance and angle. The movement is smoothed to avoid sharp movements by a function we developed ourselves instead of using an already made one like LookAt..

Events and Gamemodes

- **Race**: As most of the race logic is inside TrackCheckpoints (lap count, positions, etc), the only thing that's specific to this gamemode is what to do when the main player finishes the race. So we defined the EndRace function that displays the position in which the player ended and calls a coroutine that waits 5 seconds and takes the player back to the lobby automatically.
- **TimeAttack**: The same happens with time attack as with race. When a time attack ends the time it took the player to finish the race is displayed and then they are taken back to the lobby.

Checkpoints

- **TrackCheckpoints**: Follows a singleton Pattern, because we need to access some variables from other scripts (Race, for example) and that way we make sure there is only one instance that we can access because it's a static attribute of the class.

It's the module in charge of controlling that each car goes through the checkpoints, counting laps, determining the order in which racers finish and calling a function to end the game mode. To make it more flexible and to allow it to work with different game modes, this function is a UnityEvent that is passed through the inspector, which allows us to choose a function from another script, for example Races.cs, to execute it when the main player finishes all the laps.

Among other things, it has a list of checkpoints (ordered, where the finish line is the first), a list of cars and other lists to contain the current lap of each car, etc.

- **CheckPoint**: It’s an individual checkpoint. They have a reference to the *TrackCheckpoints* object that holds it, that is set via a function during the initialization of *TrackCheckpoints.*

  When a racer goes through a checkpoint, it calls a function from *TrackCheckpoints*, which determines which car crossed it by checking their transforms against the transform of the object that crossed the checkpoint.

AI

- **trackWaypoints**: mainly for debugging purposes, it draws Gizmos to show the waypoints on the track and generates a line between them, this way we can clearly check if the cars are correctly following the path specified.
- **inputManager:** this is not an AI only script but it's the most important, the class abstracts the controls of the car, you can choose between 2, keyboard or AI (we are planning on adding mobile controls) but the car doesn’t know which one is being driven by it as it only knows the controller is accelerating, braking, turning…
  Then we mapped those controls to a keyboard and implemented an AI that calculated the closest waypoint, then calculates a vector to a more advanced waypoint, with that vector calculates the steering angle and finally it accelerates permanently (although we have some ideas on how to change this to avoid collisions). We also created a collision avoidance system that takes the closest AI driven car and calculates a vector that steers the other way when it is dangerously close.

IMPLEMENTATION

How to run the game:

The requirements are the Unity version **2020.3.13f1** as it's the one that the teacher used during our class.

To run the game you need to unzip the project and import it to Unity Hub, we wanted to fit the unity project itself in moodle so we had to delete temporary files, so when you launch the project it wont load any scene, what you must do now is in Assets tab you must click on “Scenes” folder and double click on Lobby to load it, now you click the play icon.

We also included an executable, but it was too big to upload to moodle, so we uploaded it to a folder in Google Drive.

The executable is easy to do, just unzip the folder and double click SUPERSUPERMAX.exe file. To exit you must press ALT+F4, as we have yet to implement an exit button.

Functionality implemented

control

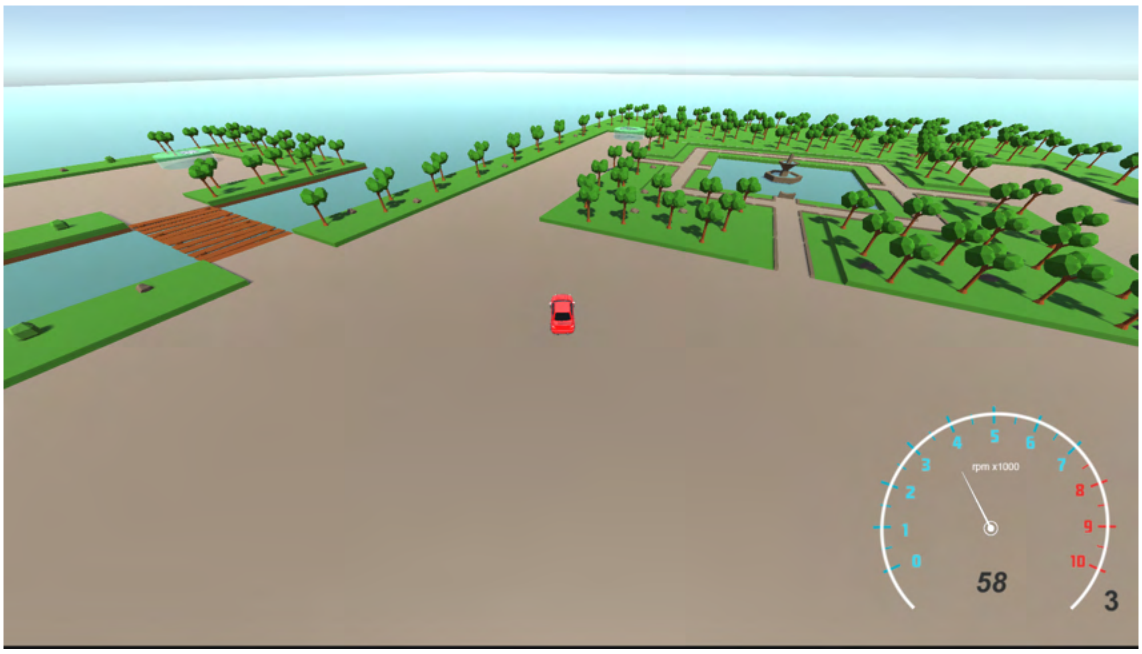
The game has the following control map:

Key	Action
W	Accelerate Forward
S	Accelerate Backwards
CTRL	Handbrake
SHIFT	Boost
Q	Upshift
E	Downshift
Space	Drift Mechanics

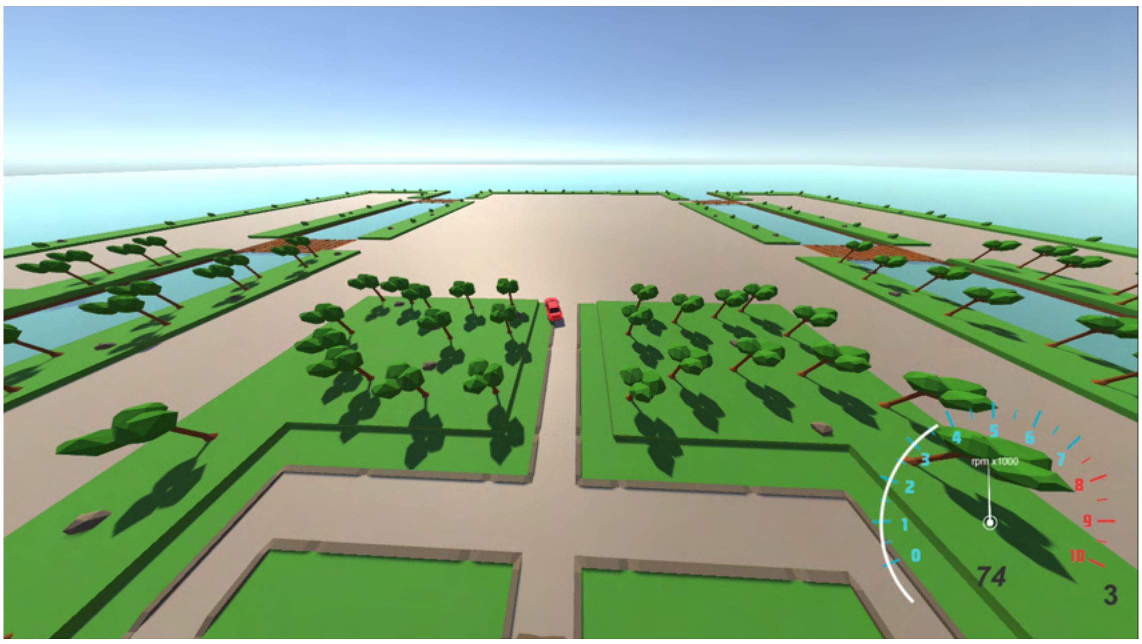
These controls are used through a keyboard, the same actions can be performed automatically by the AI, this is because we created an inputManager that abstracts the control, this way we can create a car and select if it's driven by keyboard or an AI that we designed and easily manage all the cars.

camera

On the design document we specified that the camera would follow the car but without rotating, this way we could give the sensation of more minimalism and that everything was “toy-like”, unfortunately when we implemented it we realized that racing was much more difficult without seeing ahead so we implemented a camera that rotates with the car, but instead of fixing it to the car we tried to keep the spirit of the original design and smooth the rotation, this way it retains the minimalism and simplistic feeling.



The FOV by default is 80 but it increases when activating boost to give the sensation of speed. Here is an example of the increased FOV.

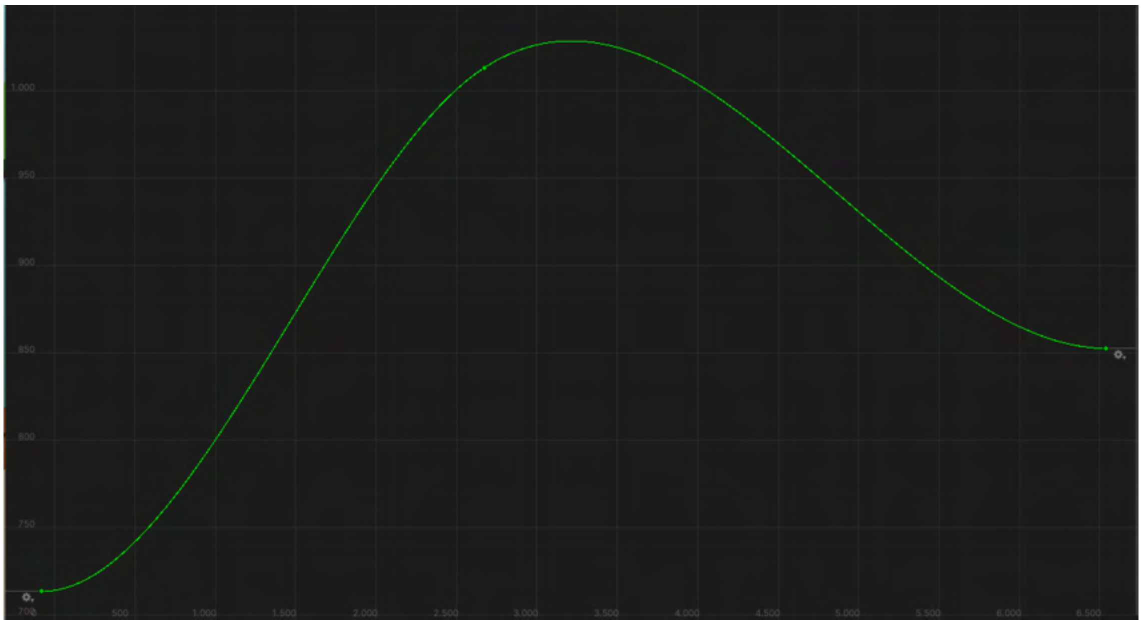


car mechanics

The game simulates a realistic engine, gearbox and steering.

Engine

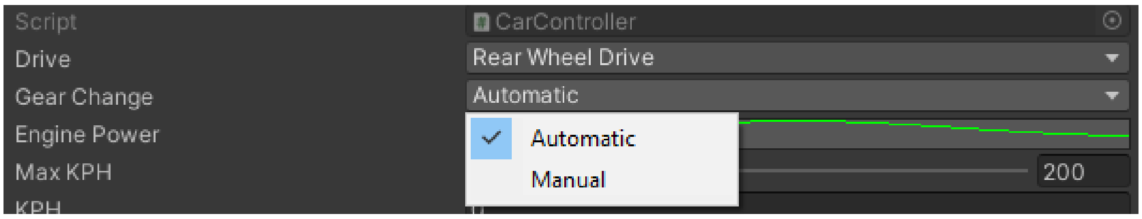
In real life different cars have different torque curves, we tried to simulate this by creating a torque curve, this way when we have low RPMs it accelerates slowly, and we need to change gear (with q or e or automatically) to get the most out of the engine.



This way we can specify what power the engine has in each RPM easily by modifying the curve on the Unity interface

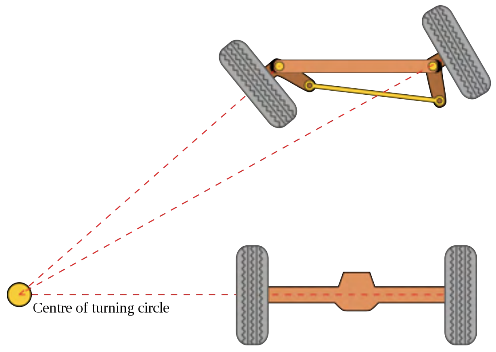
Gearbox

We take the RPMs simulated by the engine and transmit it to the wheels (two wheel drive or four wheel drive depending on the car) by multiplying it by a gear ratio, and we set minimum and maximum RPMs so that the automatic gearbox knows where to change gears, you can manually change the gears by switching the car to manual mode through the Unity interface (we will develop a menu that changes this for the final delivery).



Script CarController
Drive Rear Wheel Drive
Gear Change Automatic
Engine Power
Max KPH 200
KPH

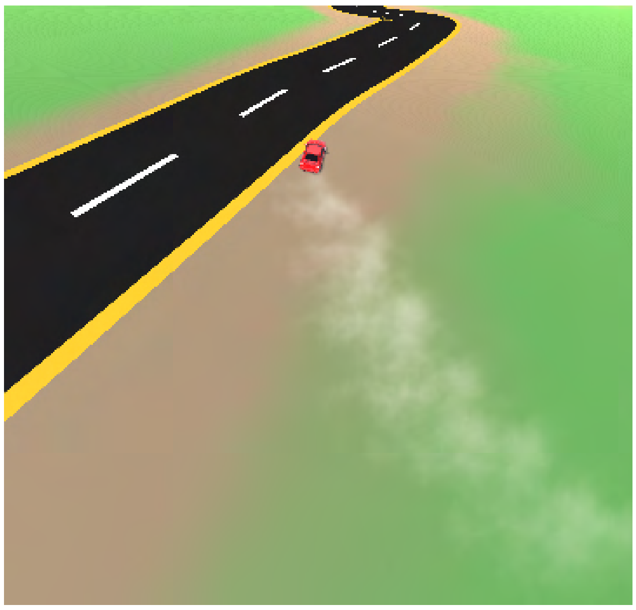
Steering



Real cars have what is called “Ackerman Steering” . This steering means that the two wheels are not switched the same angle, so that they have more grip. We implemented this system with an equation in the steering script.

Smoke

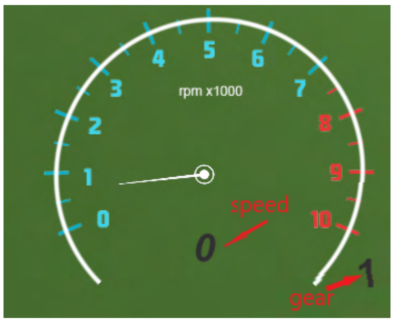
We implemented a particle system to simulate smoke only when the car drifts (the wheel collider registers a slip angle).



Nitro

We implemented a boost system activated with Shift, it adds speed to the car and refills automatically. To check how much you have left you need to look at the white line in the tachometer.

HUD

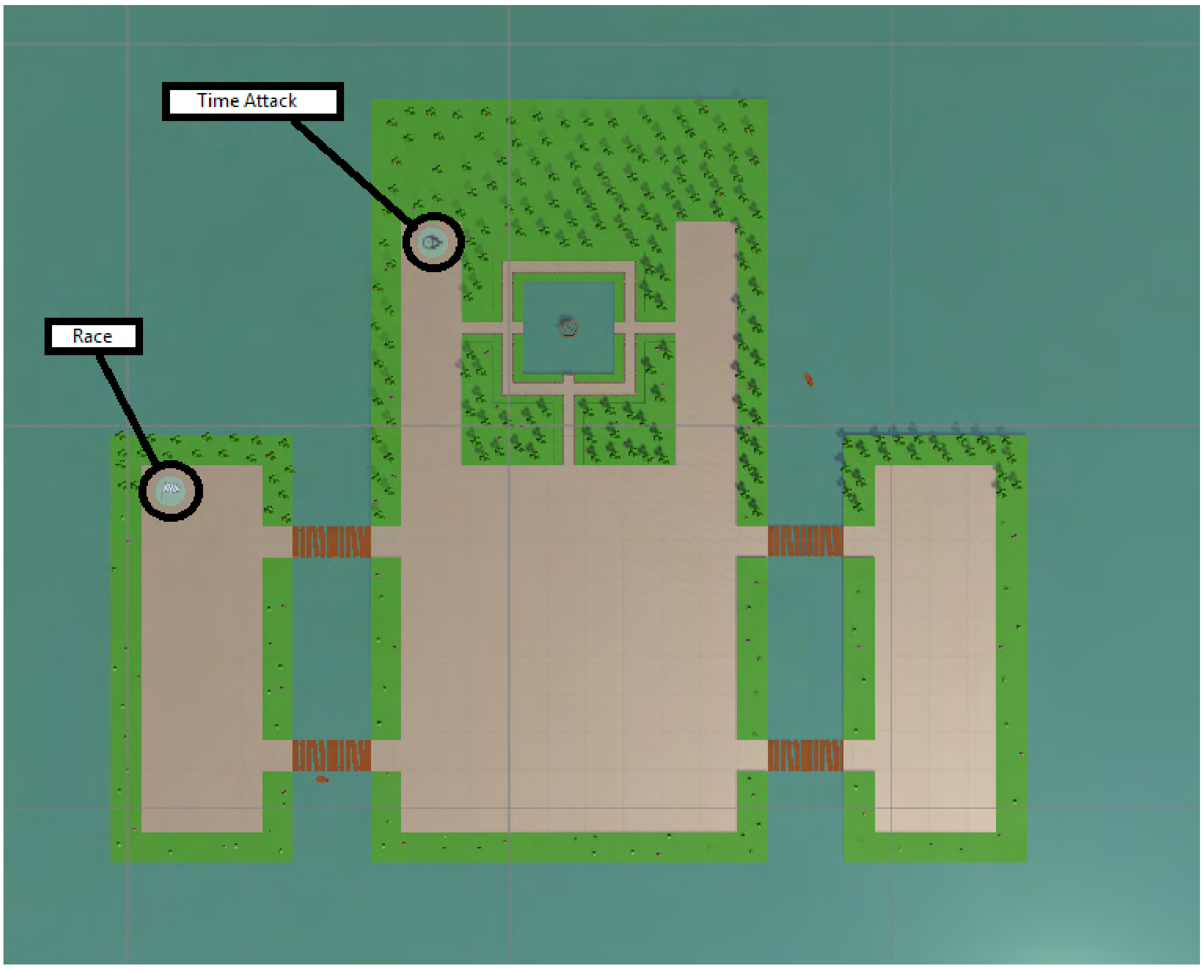


This is the interface we designed to display the tachometer, speed and current gear.

LOBBY

We created a small lobby from which the player can choose which gamemode to play by driving. To choose a gamemode the player has to collide with a small cylinder that has a picture indicating which gamemode it is.

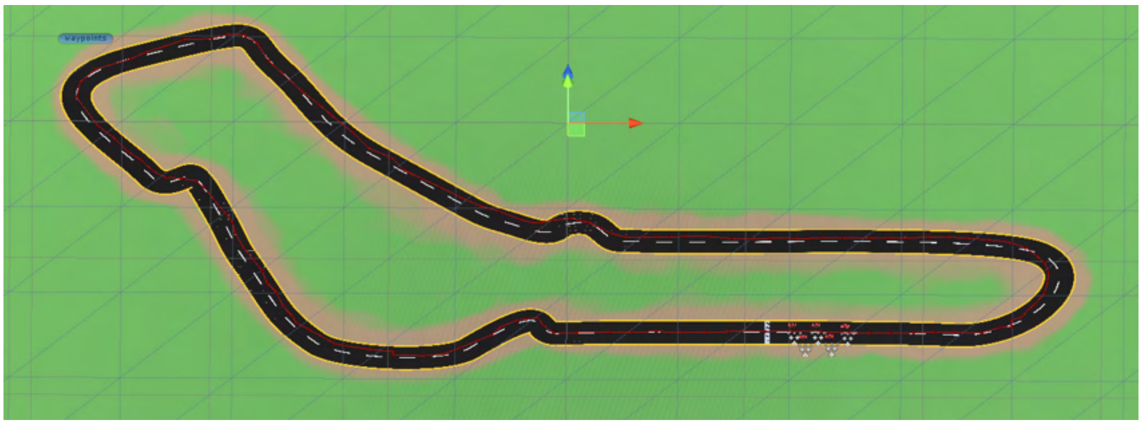
In the following picture we can see where the portals to the two implemented game modes are located.



AI

We implemented a basic AI based on waypoints. We have waypoints marking the path and the car calculates the vector from its position to the next vector and gets the x component of the vector to change the direction of the steering.

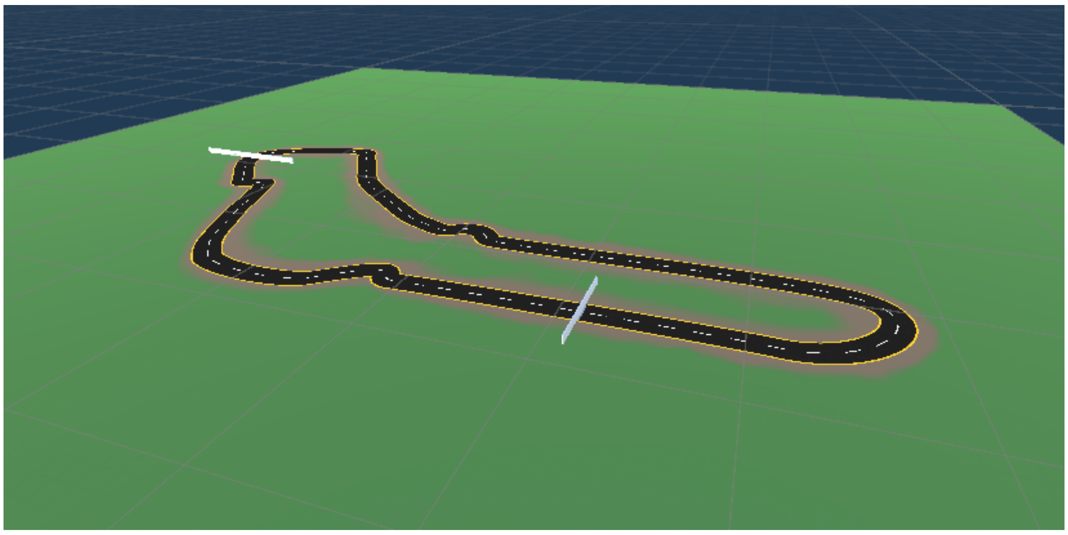
We had to also add a collision avoidance system that calculates a vector to the nearest AI car and gets its normal vector to try to change course until a minimum distance (set in the Unity inspector) is met.



Checkpoints

We made a checkpoint system to make sure players (and AIs) don't skip parts of the track (We have only 2 checkpoints in the track now as it is just a proof of concept, but the system is made to allow any number of them and in the final delivery there will be many of them) and detect when the player has completed the required amount of laps. This system also lets us obtain the order in which cars cross the finish line when they do their laps.

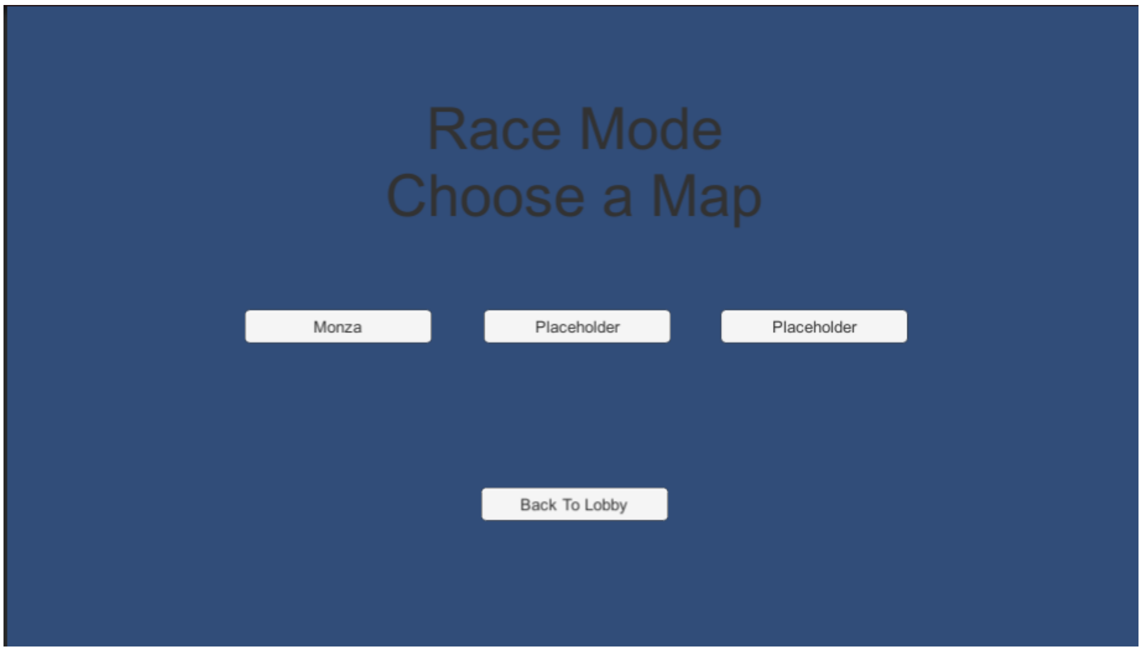
Note: If a car doesn't go through every checkpoint in the correct order, the lap will not count. So if you encounter any problem with lap counting, redo an entire lap making sure you are following the track.



Map selection menus and maps

We have implemented menus to select different maps for each gamemode. In this delivery we will only have one circuit, which is the same for all game modes we have now.

The way we've handled maps is to create a new scene for every game mode and for every map and then instantiate some prefabs that we have created to easily create such maps. (Only changing the circuit prefab between scenes of the same gamemode).



We still have to set a design language for all the menus (as they're purely functional now) and add settings like Manual or Automatic during races.

GAME MODES

The game modes implemented are **Race** and **Time Attack**, both are very similar in event coding as you need to complete a lap and it detects when you start and end (see Checkpoints section) but Race also adds cars driven by the AI (see AI section), and shows the place you finish. The interface and menus for both modes are very rudimentary but the logic is done.

Testing

We've tested the game ourselves by playing the modules we just finished doing and then gave our friends the build (.exe) file so that they could try it out and report back bugs.

Bugs identified and solved during testing:

- AI breaks when it completes a full lap.
- AI cars crash into each other and stop racing.
- You can shortcut the lap by just looping through the finish line/start grid.
- Gearbox changes gear too fast, or too slow.
- Engine reports negative RPMs.
- Smoke freezes on air when it stops drifting.