TRUST ELPLAN

# SUPER SUPER MAX

## GAME DESIGN DOCUMENT

Rodrigo Juez

Alejandro Benimeli


11/01/2022

# Content

# Overview of the Delivery

## Introduction

The objective of this document is to show the progress of the SUPERSUPERMAX game at the time of the final delivery. This does not mean the game is fully finished, as artists and developers, we always want to add more, and improve the mechanics, but we have limited resources and time.

We've tried to give the sensation of final and finished game, but everything is designed to add more tracks, gamemodes and

## General Delivery Description

The delivery includes:

- **Unity project**: Where we have developed the game, includes the assets, sounds, scripts…
- **Executable game:** The final build we've used for testing.
- **User manual:** Explains how to play the game to an end user.
- **GDD:** This very document, which explains the design.

## 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 working on separate features. Each one of us had different roles and tasks assigned. They can be seen in the scope section of the document.

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.

## 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.

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.

# summary

## Genre and Theme

SUPERSUPERMAX is a racing game, with a low poly aesthetic, and themes based on the scale of objects.

## Main mechanics (summary)

Driving is the center of the game, all the mechanics revolve around that.

We tried to carefully choose the grip of each car and power, so that the player can choose if he wants a very fast car in the straights but that it must brake a lot in the curves or it will have under/oversteering.

We also added a short burst of nitro so that the slower cars can accelerate a bit more, but if the user misuses it could work against him as the car can easily spin.

Interacting with objects in a scene, we included many tires, roadblocks, and other non conventional items (like giant barrels or giant rubber erasers) so that the player can spend time in the lobby messing around and in the races we have a dynamic component.

## Target Platforms

| Minimum Specs | |
|---|---|
| Operating System | Windows 10 x86, 64 bit |
| CPU | 2 Core 64 bit CPU not older than 6 years |
| GPU | Intel, AMD or Nvidia GPU not older than 8 years |
| RAM | 4 GB or more |
| Screen | 16:9 Aspect Ratio (Preferably 1920×1080) |
| Hard Drive Space | 1 GB |

**Note:** We ran out of time but we've done a layer of abstraction to switch between tactile and keyboard controls so that we can do an Android app. We will finish developing it after the delivery is done so that we can play on our phones.

## Scope

The game is infinite, that is, the player can play as long as it's fun, you could argue that when you unlock all vehicles you finish but we think that there is still more fun to have until you master all vehicles in all races, plus knocking around stuff in the lobby and pursuit is fun too.

| Development Team (Total 2) | | |
|---|---|---|
| | **Alejandro Benimeli Miranda** | **Rodrigo Juez Hernández** |
| **Roles** | Team Lead, Developer, Tester | Tech Lead, Developer, Tester |
| **Tasks** | UI (menus, loading, animations, design language and consistency), Documentation, gamemodes events (race checkpoints, positioning system, coin earning system, time counter, collision detection for pursuit), modeling (lobby layout). | Car simulation (engine simulation, gears, drift, particles, sound systems, mass of the vehicle), vehicle selector (shop system to buy vehicles), input management, AI (one AI for racing and one for Pursuit). |

## Influences

We inspired the game in many mobile, PC and real life tracks. We tried to do a compendium of everything we liked of them with our designs and aesthetic under one single game.

- Absolute Drift
  - Videogame
  - This is an indie mobile video game. It's about drifting with a classic anime car. We liked the drift mechanics and how sometimes it's very difficult to steer, and sometimes the acceleration is higher. Our videogame does a similar thing, some cars are very fast but you have to be very handy with the brake to turn or you'll go straight or spin.
  - We also liked the "interactables", that is, objects that are throughout a scene and you can play with them, knock them out, etc.

- Pako
  - Videogame
  - We really like the aesthetic and the concept of "dumb police" that only follow you around and tries to crash you, that is we designed Pursuit mdoe, but our game takes into account all the different

handling mechanics and power of the rest of the game, whereas Pako is a simple pursuit and nothing more.

- Ryza Roads
  - Videogame
  - We just simply like the aesthetic of the cars, so we tried to look for low poly cars too.

- Circuit de Spa-Francorchamps
  - Real Life Track
  - When designing the “School Track” we took some curves out of this track. For example the first chicane in that track is the famous “Eau Rouge”, a chicane (two sequential turns) paired with a hill that makes the car very difficult to handle!
  - This does not mean we copied the exact track, we only got some ques.

- Laguna Track:
  - Real Life Track
  - The famous “Corkscrew” at Laguna track, we also baked it into the “School Track”, we really like this turn because it forces you to go very steady with the accelerator and handling.

- Monza
  - Real Life Track
  - “Mountain Track” is heavily inspired by this Italian track, specially the long straights where the cars go extremely fast and is very easy to spin out due to the sheer speed.

- F1
  - Sport
  - The name of the game is heavily inspired by the fan song made for Max Verstappen and the company name is straight out of the Fernando Alonso “EL PLAN” joke.
  - We also wanted an AI that behaved like real drivers, not “train machines” that only go through a rail and don't make mistakes. That is why we designed a race AI that sometimes spins, or drifts because maybe it accelerates too much. They also sometimes try to avoid other cars or crash just like real life pilots.

# Project Description

## Game modes

### Race

The player must complete two laps in a circuit chosen by them, competing against four enemy artificial intelligences (except in some cases where there may be less AIs, like harder tracks or tracks with very thin roads).

The player must also follow the track at all times (there may be shortcuts in very specific cases, but this is a general rule). If the player tries to avoid parts of the track, the lap doesn't count until they go back to the part they skipped and follow the track from there.

At the end of the race the position the player ends in determines the amount of coins that are rewarded.

### Time attack

The player must complete two laps in a circuit chosen by them, trying to do so in the least amount of time possible.

Like in a Race, the player has to follow the track at all times and not skip parts to gain an unfair advantage, obtaining lower times by not fully completing the circuit.

At the end of time attack the amount of time it took the player to complete the laps determines the amount of coins that are rewarded.

### Pursuit

The player spawns in a big open area. Every certain amount of seconds, random cars spawn in the borders/corners of the map. This keeps happening until the mode ends.

These cars follow the player and will try to crash into them. The player must run from them and avoid them for as long as they can. If one of those cars crashes into the player (or the player crashes into the border of the map), they immediately lose. Every time an enemy car crashes into another vehicle or the border of the map, they explode, avoiding a situation where too many enemies coexist at the same time.

At the end of pursuit the amount of time the player managed to survive determines the amount of coins that are rewarded.

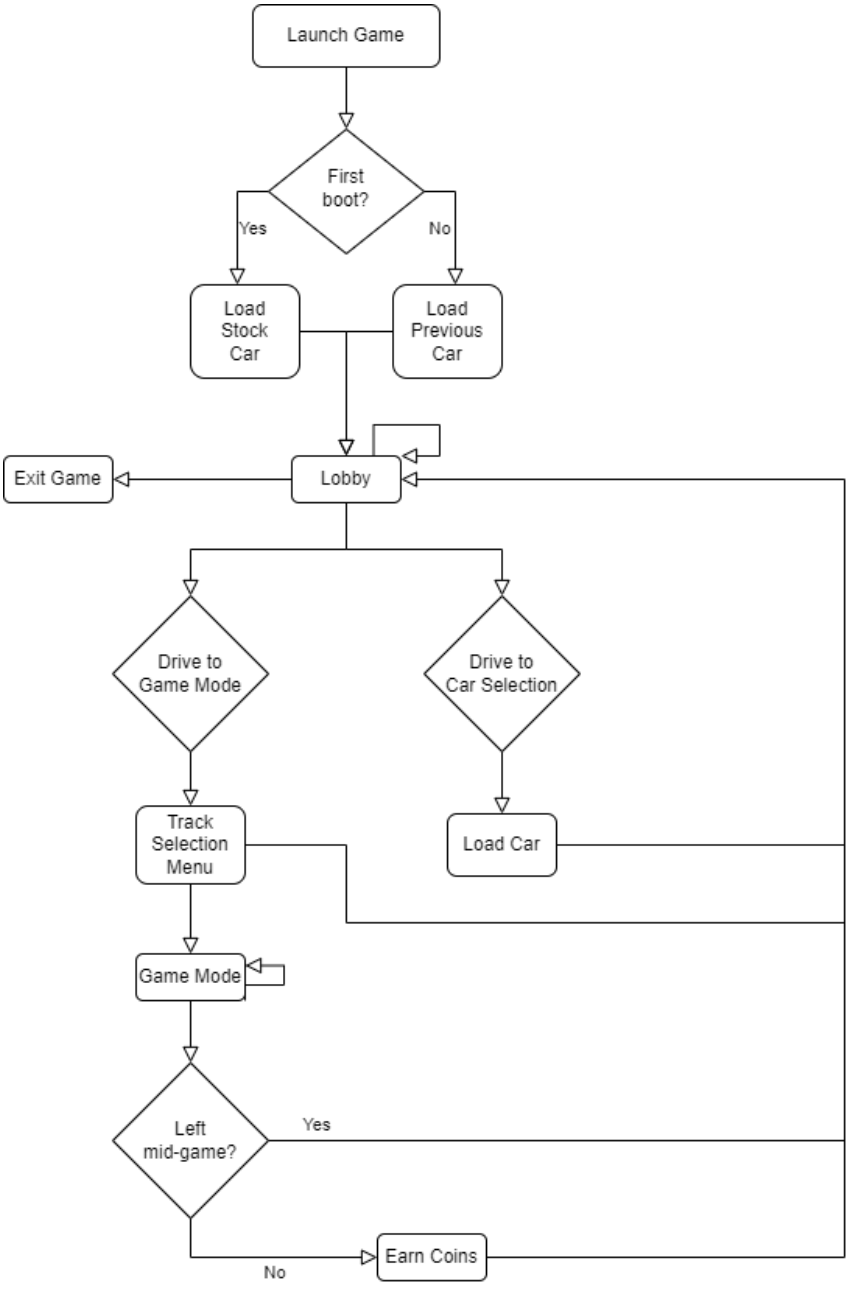
# Game Mechanics

- Driving
    - It's the most important mechanic of the game, as except for some menus, everything is done by driving.
    - Turbo, drifting, front end grip/rear end grip
- Different vehicles with different stats
    - Each vehicle must drive differently to give variety and a sense of progression as you unlock better cars.
    - To obtain this:
        - Each vehicle has its own torque curve (different motor) and motor force.
        - Each vehicle has a different mass.
        - Some vehicles are front wheel drive, others rear wheel drive and others all wheel drive.
- Coins and Purchasable Vehicles
    - The player earns coins when finishing a game mode. The amount depends on their performance. Afterwards, these coins may be used to purchase different vehicles.
    - Race gives 1000 coins to the winner, 750 to the runner up, 500 to the 3rd, 250 to the 4th and 50 to the player if they're last.
    - Time Attack gives 1350 coins minus 250 multiplied by the amount of time in seconds that have passed since the start of the mode. It always gives a minimum of 50 coins.
    - Pursuit gives 3.33 coins per second that the player is alive. This means it takes exactly 5 minutes to earn 1000 coins. There is no upper limit to this amount.

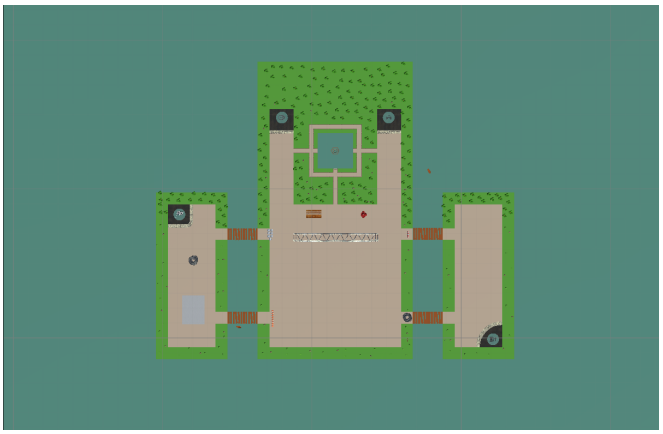
# Structure and Levels Developed

## Navigation between Levels

The following diagram shows how the player can navigate through the different levels and menus in the game. We can clearly see how the lobby is the center of the game, as intended by us.

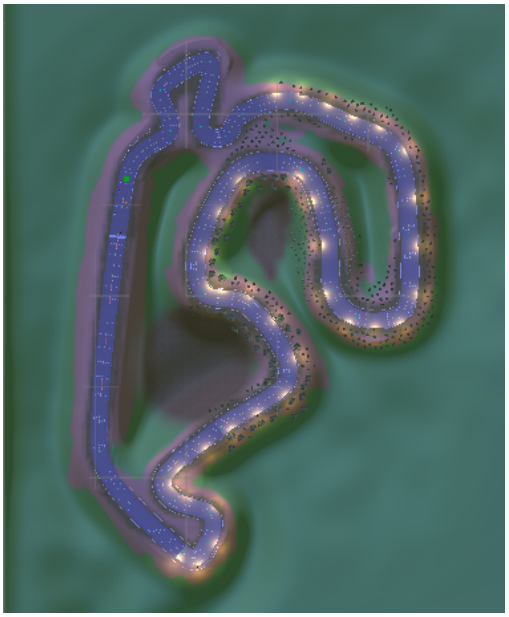


## LOBBY



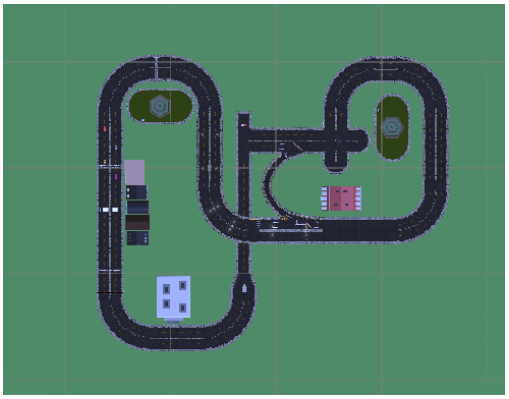
Built by Alejandro using assets from the Unity Asset Store. The lobby has the different portals to the different game modes and to the vehicle selection screen. It also serves as a place for the player to learn the controls or to just drive around with no goal in mind. We added props such as tires or fences that the player can crash into.

## MOUNTAIN



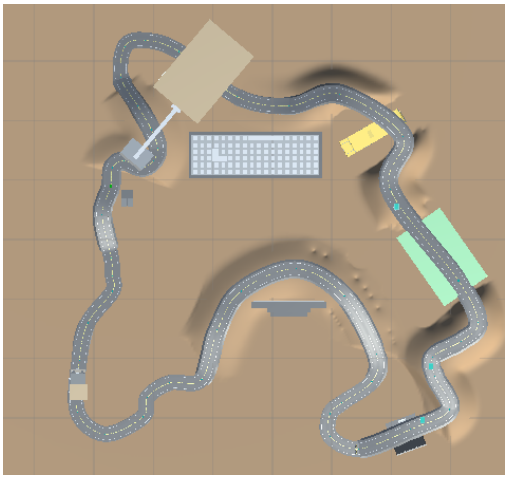
Built by Rodrigo using the Road Architect plug-in. A link to this plug-in can be found in the resources section. The circuit is built at the bottom of a mountain range, following the curvature of the different mountains that form it. This circuit is used for Race and Time Attack. Inspired by Monza main straights followed by a complex chicane.

## CITY



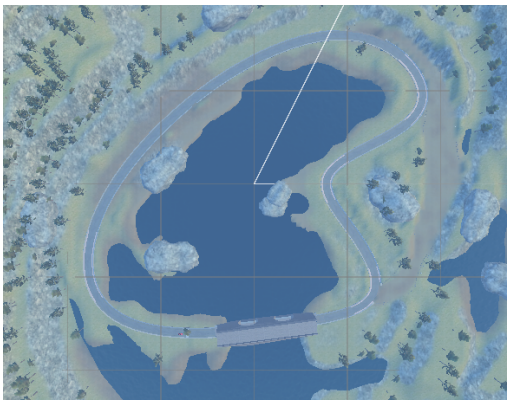
Built by Rodrigo using assets from the Unity Asset Store. The track uses the roads of a small city. There are different elevations and at some point one road goes under a bridge which is another road. This circuit is used for Race and Time Attack.

## SCHOOL



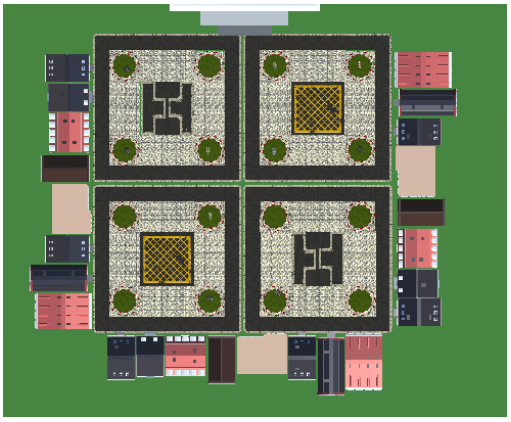
Built by Rodrigo using the Road Architect plug-in. Plays with different elevations and has school themed props in a giant scale. Some props like some notebooks are found on the road and can be pushed aside by driving into them. This circuit is used for Race and Time Attack. The first chicane is inspired by Eau Rouge in Spa-Francorchamps Track, then the descent from the “giant book” decoration is inspired by “The Corkscrew” in Laguna Track.

### Lake



This track was obtained from the Unity Asset Store and the link to it can be found in the resources section. This circuit is used for Race and Time Attack.

### Town



Built by Rodrigo using assets from the Unity Asset Store. Flat and open square map with some props scattered around. Random cars that follow the player spawn in the corners. This circuit is used for Pursuit.

## Vehicles

**Bus**

Slow speed but bad acceleration. Very hard to turn.

## Cheap car

Slow speed but bad acceleration. Good grip.

## Old car

Slow speed (slightly faster than the cheap car) but bad acceleration. Good grip. Manual gear change.

## Scooby Doo van

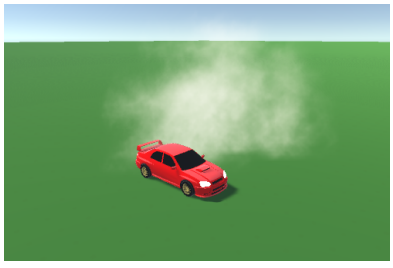
Slow speed. Low grip when turning and has understeering (hard to turn at high speeds).

## Homer car

Fast speed but bad grip. The pink one has a bit better grip.

## Camaro

Fast speed but spins a lot in curves. The pink one has a bit better grip.The yellow one has better grip.



## Subaru

Fast speed and good grip. 4 wheel drive.

# Class and Scripts Structure

As we will explain the scripts in the design section, here we just show the structure or organization of all the scripts we developed for the game

- **Events and Game Modes**
  - Race.cs
  - TimeAttack.cs
  - Pursuit.cs
  - CarExplodeOnCollision.cs
  - PlayerCollisionDetection.cs
  - DeleteSelf.cs
  - **Checkpoints**
    - TrackCheckpoints.cs
    - CheckPoint.cs
- **UI and HUD**
  - GameManager.cs
  - FollowingCameraController.cs
  - UpdateCoins.cs
  - **Scene Management**
    - MenuTrigger.cs
    - SceneLoader.cs
    - LoaderCallback.cs
  - **Vehicle Select**
    - awakeManager.cs
    - vehicleList.cs
- **AI**
  - trackWaypoints.cs
  - inputManager.cs
- **Car Logic**
  - CarController.cs
  - SmokeSystem.cs
  - Audio.cs

## Graphical Elements Developed

**Menus:** To choose which track to play when selecting a game mode.

**Car Speedometer:** It shows the speed, revs and gear that the car is in at any given time. It also shows the charge of the turbo.

**Game Modes HUDs:** Elements like displaying the current lap, the position of the player, the timer in pursuit and time attack.

**Car Store:** Shows the player the different cars they can buy, lets them cycle through them and displays their price. A button allows the player to purchase one if they have enough money.

**Loading Screen when loading complex scenes:** A loading animation is played when loading tracks or the lobby.

## Other Elements Developed

- Explosion Particle Effect: For when enemy cars collide in Pursuit.
- Tire Smoke: When the wheels slide when drifting.
- Motor Sounds.
- Animation in Game Mode Menus when hovering or clicking on a track.

## Input/Output

The player interacts with the game using their keyboard. We abstracted the control system to be able to easily allow other control methods in the future (controller for example) but for this delivery only the keyboard controls are functional.

The script that handles the input is called *inputManager*, which is the script that also handles both AIs driving (Race AI and Pursuit AI).

The control scheme is the following:

| CONTROL | BIND | EXPLANATION |
|---|---|---|
| Accelerate | W | Moves the car forward |
| Brake/Go Backwards | S | Stops the car or makes it go backwards |
| Turn Left | A | Turns the car left |
| Turn Right | D | Turns the car right |
| Drift | Spacebar | Locks the rear wheels, allowing the car to drift |
| Hand Brake | Left Control | Locks the wheels, braking almost immediately |
| Turbo | Left Shift | Gives the car a speed boost for a few seconds |
| Go to Lobby<br>Exit Game | Esc | Goes to the lobby and if the player is in the lobby, it exits the game |
| Reload Scene | R | Reloads the current scene (Only applicable to some scenes, like the lobby or the different game modes) |

## GAMEPLAY VIDEO

Click on the image or the link below to see the gameplay video!


https://www.youtube.com/watch?v=al_LB4WfxSo

## Other Elements not considered above

- Racing AI with basic car avoidance and also recovery from accidents.
- Pursuit AI
- Checkpoint system: To avoid skipping parts of the track. It is also used to calculate the position of the player in regards to the AI racers.
- Portals in the lobby to go to other levels or menus: These portals activate when the player drives through them, allowing the player to choose what to play by just driving.

# Design

## Events and Game Modes

- **Race.cs**: 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, gives them coins and calls a coroutine that waits 5 seconds and takes the player back to the lobby automatically.
- **TimeAttack.cs**: 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 (along with their best time), they are given coins and then they are taken back to the lobby.
- **Pursuit.cs:** The same happens with Pursuit as with Race. When a Pursuit ends the time the player survived is displayed, they are given coins and then they are taken back to the lobby.
- **CarExplodeOnCollision.cs:** This script is added as a component to the cars that are spawned in pursuit so that when they crash into another vehicle or the border of the map, they display an explosion animation and the car is destroyed.
- **PlayerCollisionDetection.cs:** This script is added to the player car as a component so that when it crashes into an AI car or the border it calls the EndPursuit function in Pursuit.cs.
- **DeleteSelf.cs:** It's a simple script that has a countdown and then it destroys the Unity object it is attached to. We use it to destroy the game object that is created with each explosion when its animation has finished.

### Checkpoints

- **TrackCheckpoints.cs**: 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.

It also calculates the position of the player in relation to the AI cars each time any car crosses a checkpoint.

- **CheckPoint.cs**: 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.

## 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.cs**: Updates the interface based on the speed, gear, nitro and RPM needle based on flags and variables public in the CarController component.
- **FollowingCameraController.cs**: 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..
- **UpdateCoins.cs:** This script is used to read the amount of coins a player has from where they're stored in a PlayerPref and dis

## Scene Management

- **MenuTrigger.cs**: 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.
- **SceneLoader.cs**: It's a script that holds a function that receives a scene name and changes the scene to it. It first changes the scene to the loading scene and sets the scene we want to change to as a callback (Loader Callback).
- **LoaderCallback.cs:** Script used in the loading screen. As soon as the loading screen is fully loaded (the first update), it starts changing to the scene we originally wanted to change to asynchronously (to do it in a different thread so the loading animation is more responsive).

### Vehicle Select

- **awakeManager.cs:** Controls both the appearance of the vehicle shop (Show the car turning, the buttons, etc) and the functionality (What happens when the buttons are clicked, the purchase of vehicles using coins and changing the current vehicle of the player and saving it in a PlayerPref).
- **vehicleList.cs:** Very simple script that only is a class with a list of game objects. This allows us to add this script to an object to have a list of all the vehicle prefabs that we use.

## 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 3, keyboard, Race AI or Pursuit 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.

## Car Logic

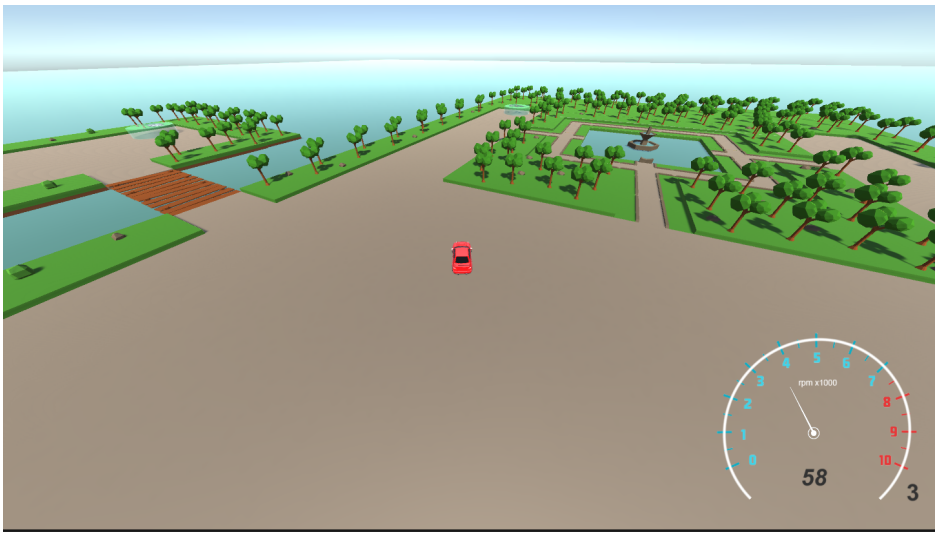
- **CarController:** The car gets its orders from the inputManager class, this is to abstract the control system (AI controls, keyboard controls, mobile controls…), then we generated a function for each simulated part of the car:
  - **calculateEnginePower**: updates the RPM to which the engine is running 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.
- **audio.cs:** This script is in charge of controlling the sounds that the car engines make.

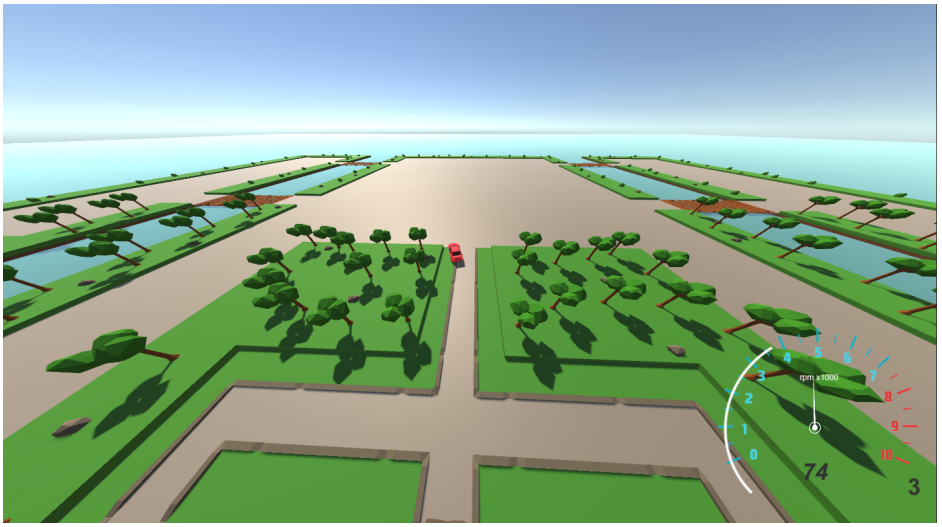
# Functionality implemented

## 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.

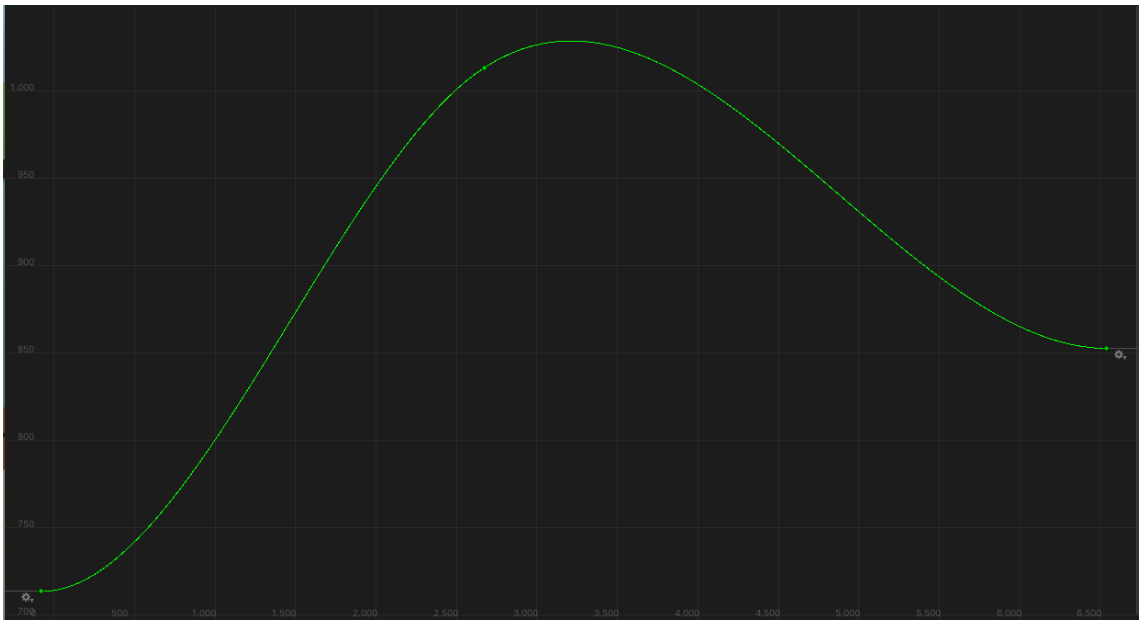


## car mechanics

The game simulates a realistic engine, gearbox and steering.

### Engine

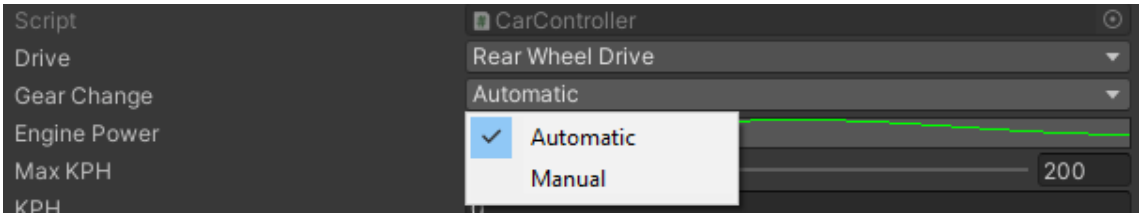
In real life different cars have different torque curves, we tried to simulate this by creating a torque curve, this way when we have low RPMs it accelerates slowly, and we need to change gear (with q or e or automatically) to get the most out of the engine.



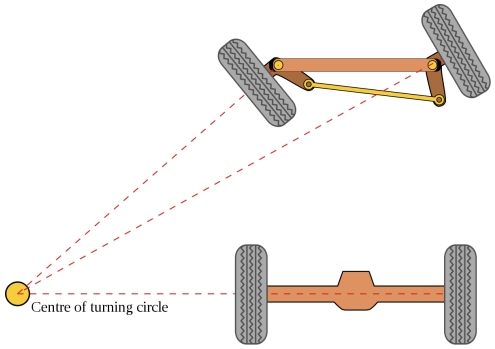
This way we can specify what power the engine has in each RPM easily by modifying the curve on the Unity interface

### 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. Most cars are automatic. Only one is manual (Old Car).



## 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

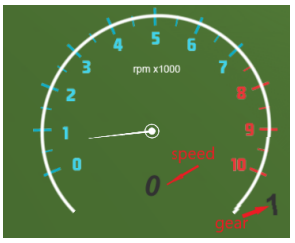
We implemented a particle system to simulate smoke only when the car drifts (the wheel collider registers a slip angle).

## 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.

## Tachometer HUD

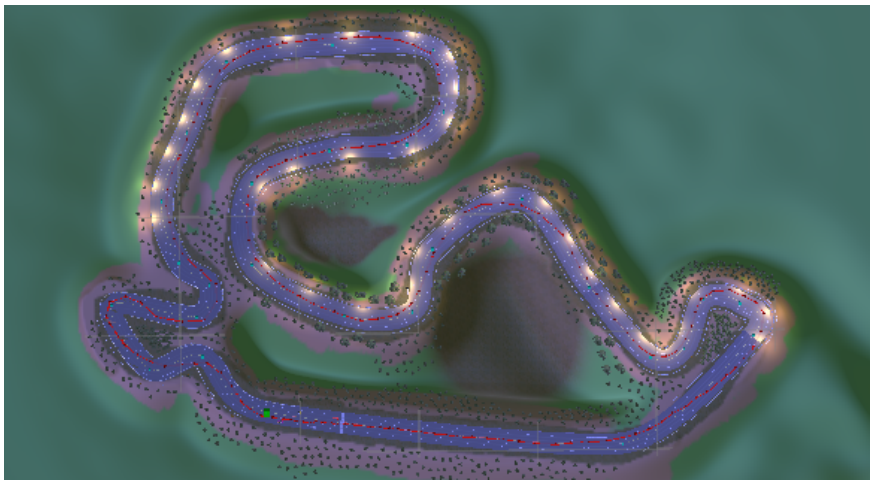
This is the interface we designed to display the tachometer, speed and current gear.



## Race 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.



## Pursuit AI

This AI works by calculating its position and the position of the player to calculate the vector, which is used to change the direction of the steering. This one only follows the player, trying to crash into them. It has no collision avoidance system like the race one.

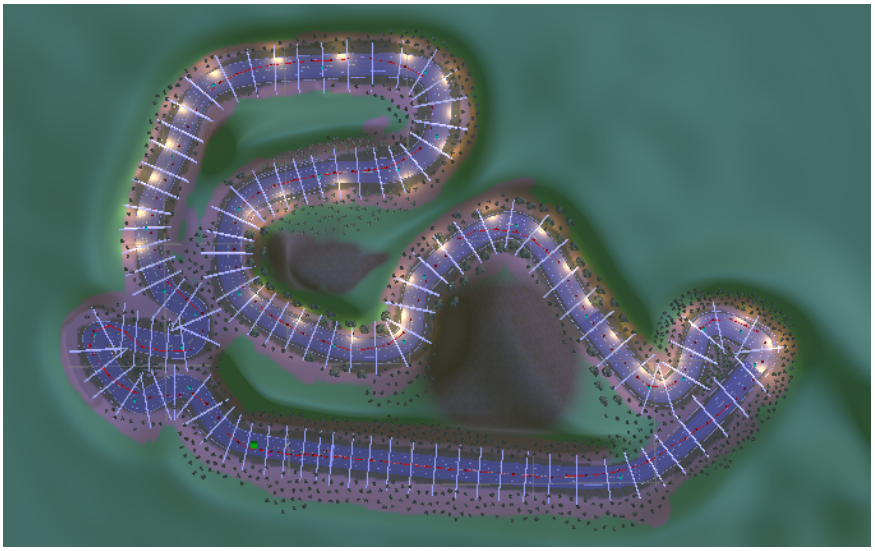
## Car Store

Allows the player to cycle over all the cars and choose the one they want to use. There is a purchase button for the cars that are not owned by the player.

## Checkpoints

We made a checkpoint system to make sure players don't skip parts of the track 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.



## Race positioning system

We also use the checkpoints to update the position of the player. The way this works is by keeping the following information for each car that is in the race (player car and AIs):

- Current lap number
- Reference to the last checkpoint they crossed
- Time at which they crossed the last checkpoint

Each time any car crosses any checkpoint, using the information listed above we recalculate the position of the player in relation to the AI cars.

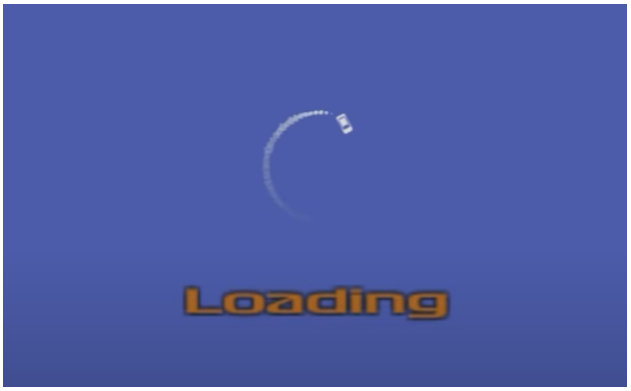
## Map selection menus and maps

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.


## Loading screen

When loading into complex scenes (such as the lobby or the different circuits) we show a loading screen while it loads to avoid having the game lag in the previous screen while its loading.




The way we do it is by loading a scene with a loading animation and, that scene, loads the scene we want to load in the background asynchronously. This way, until the scene is fully loaded, the loading animation will play.

## GAME MODES

The game modes implemented are **Race, Time Attack** and **Pursuit**.

Both Race and Time Attack 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 Pursuit mode is a bit different as it has no checkpoints. In this mode we only detect when the player has collided with anything and stop the timer and end.

Race


Time Attack


Pursuit

# Resources

## 3D Assets

- 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
- Food pack - 3D Microgames Add-Ons | 3D | Unity Asset Store
- Free PBR Barrel | 3D Props | Unity Asset Store
- Lake Race Track | 3D Roadways | Unity Asset Store
- Low Poly Cars | 3D Land | Unity Asset Store
- Low poly European City Pack | 3D Urban | Unity Asset Store
- Low Poly Police Car Pack | 3D Land | Unity Asset Store
- Low-Poly Retro City Car 03 | 3D Land | Unity Asset Store
- Low Poly Street Pack | 3D Urban | Unity Asset Store
- Low Poly Tree Pack | 3D Trees | Unity Asset Store
- School assets | 3D Environments | Unity Asset Store
- School Supplies | 3D | Unity Asset Store
- Ukraine free cars | 3D Vehicles | Unity Asset Store
- ARCADE: FREE Racing Car | 3D Land | Unity Asset Store
- 3D Tire | 3D Props | Unity Asset Store

## UI

- simple icon pastel tone | 2D Icons | Unity Asset Store

## Plug-in

- RoadArchitect

## Audio

- Nintendo Wii - Mii Channel Theme - YouTube
- Pitstop Boys - Super Max F1
- Police Siren Loop Sound Effect [FREE DOWNLOAD | ROYALTY FREE]
- 1 Hour Relaxing Engine Sound (diesel electric locomotive 2TE10)

# MVP

In the first document we specified the following MVP:

- Functioning car with gears and a torque curve.
- Lobby where the player can drive around.
- Time Attack:
    - 1 map.
- Pursuit:
    - Basic AI.
    - 1 map.
- 2 different cars with different stats.

We have accomplished everything from the MVP of the first document. Not only did we do everything, but we also implemented more functionality into the game:

- We added a whole new game mode: A race mode with enemy players controlled by an artificial intelligence.
- We added 7 cars, instead of 2. Along with this we made a shop where you can buy new cars and change between them.
- Coin system where each game mode awards coins depending on the player performance.
- 4 circuits instead of 1 for the Time Attack and Race modes.

# PROJECT MANAGEMENT

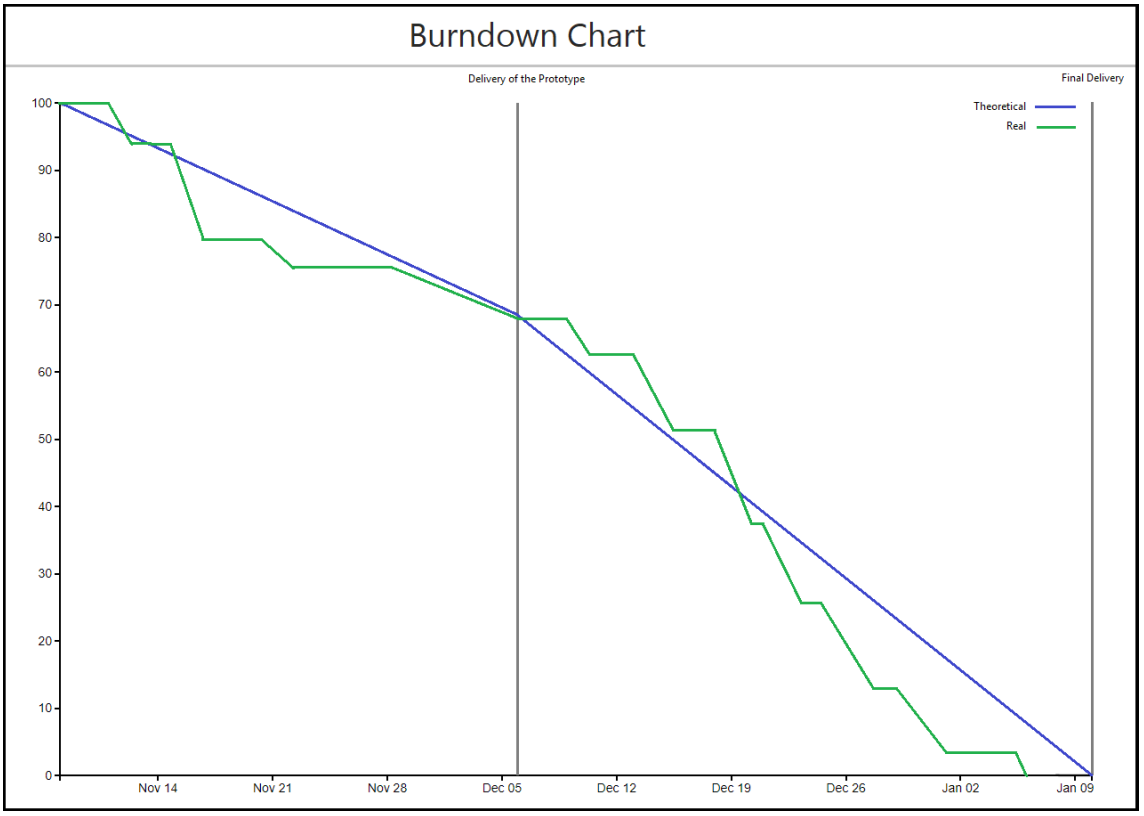
We set weekly meetings to show each other what we developed during that week and assign tasks to be completed for the next week. Everyone was free to work whenever it was more convenient for them, as it was hard to find times where we both could work at the same time due to the coursework from other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

Although we managed to complete everything in time, it doesn't mean that we finished every deadline in time. In fact, due to adding new functionality and errors we made when estimating how much each new feature would take as we had never used Unity before, some deadlines were not finished in time or were finished way before the expected date (which allowed us to add the new functionality).

Here is an approximation of our progress on the videogame over time, from when we started to the final deadline. The theoretical (blue) line was estimated calculating that 30% of the work would be done for the prototype and the remaining 70% for the full delivery.




We tried to balance the workload so that we didn't do too much or too little work at any point. However, this was not always possible due to exams or assignment deliveries for other courses. Despite this, we did our best to compensate before or after those so we wouldn’t be too far behind schedule.

This was only until the Christmas break, because after finishing our classes we dedicated more time to working on the videogame to finish it as soon as possible. As we can see, we finished around a week before the delivery date. The remaining time was used to redact this and other documents that are also included in the delivery.

# 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.

This was done at every step of the project, so if new bugs arised, we would check the things that were changed that led to the error. This also allows us to be confident that something works (or it's very unlikely that there is a bug), before adding new functionality.

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.
- Positioning system would always place the player at the last spot.
- Positioning system would always place the player at the first spot and give a null reference exception when an AI car crossed a checkpoint.
- The music that plays when you finish a game mode would play at the start.
- The timer in pursuit wouldn't reset in some cases when the game ended and you started a new one.
- The "best time" text sometimes appeared in the race modes.
- The game doesn't exit when pressing ESC.
- The help banner doesn't show after a race even if we hold the H button.
- The store doesn't report the car prices correctly.
- We had to do a complete rework of the buying system because we used an alternative method to PlayerPrefs and it didn't save correctly.
- The vehicle first shown when entering the vehicle store isn't the one that is really selected on the list.
- The cars freeze after a while in the store.