



Pinewood Rally – Unity Racing Game

Manu Jose

N00200555

Supervisor: Joachim Pietsch

Second Reader: Timm Jeschawitz

Year 4 2024/25

DL836 BSc (Hons) in Creative Computing

Abstract

The application I have built is a multiplayer racing game made using the Unity engine with assets being designed in Blender, Figma and affinity designer. I also took assets from the Unity asset store. The game itself has low poly graphics and a cartoony aesthetic. I felt this would maximize performance and be more feasible to build by myself. The first steps in development of this project were to research and study various racing games and how multiplayer and artificial intelligence is implemented in them. I researched games similar to the one I intended to make and how they utilize certain features to enhance user engagement and enjoyment. I also did a research report on artificial intelligence in racing games as part of my research and analytics module this year, which I used in deciding which artificial intelligence model to use. That report also gave me an insight into effective game design in terms of user engagement.

Acknowledgements

I would like to thank the staff and faculty of IADT for guiding and teaching me throughout my years here. I would particularly like to thank Joachim Pietsch, my supervisor, and Tim Jeschawitz, the second reader, for their supervision on this project. I could not have undertaken this project without the continued support and teaching from them.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

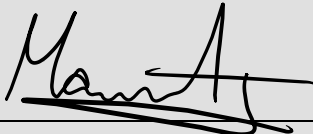
The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Manu Jose

Signed 

Failure to complete and submit this form may lead to an investigation into your work.

Table of Contents

1	Introduction.....	1
2	Research	2
2.1	Introduction.....	2
2.2	Waypoints and Vector Calculations	2
2.3	Machine Learning.....	2
2.4	Artificial Neural Networks	3
3	Requirements	4
3.1	Introduction.....	4
3.2	Requirements gathering.....	4
3.2.1	Forza Horizon	4
3.2.2	Gran Turismo	5
3.2.3	Art of Rally.....	5
3.3	Requirements modelling.....	6
3.3.1	Functional requirements	6
3.3.2	Non-functional requirements	6
3.4	Feasibility.....	6
3.5	Conclusion	7
4	Design	8
4.1	Introduction.....	8
4.2	Program Design	8
4.2.1	Technologies	8
4.2.2	Structure of Unity	9
4.2.3	Application architecture	11
4.3	User interface design	12

4.3.1	Level Design	12
4.4	Conclusion	12
5	Implementation	13
5.1	Introduction.....	13
5.2	Scrum Methodology.....	13
5.3	Development environment.....	14
5.4	Sprint 1	15
5.4.1	Goal	15
5.4.2	Item 1	15
5.4.3	Item 2	15
5.4.4	Item 3	16
5.5	Sprint 2	16
5.5.1	Goal	16
5.5.2	Item 1	16
5.5.3	Item 2	17
5.6	Sprint 3	18
5.6.1	Goal	18
5.6.2	Item 1	18
5.6.3	Item 2	18
5.7	Sprint 4	19
5.7.1	Goal	19
5.7.2	Item 1	19
5.8	Sprint 5	19
5.8.1	Goal	19
5.8.2	Item 1	19
5.8.3	Item 2	20

5.9	Sprint 6	20
5.9.1	Goal	20
5.9.2	Item 1	20
5.9.3	Item 2	21
5.10	Sprint 7	21
5.10.1	Goal	21
5.10.2	Item 1	21
5.11	Sprint 8	21
5.11.1	Goal	21
5.11.2	Item 1	22
5.11.3	Item 2	22
5.12	Sprint 9	22
5.12.1	Goal	22
5.12.2	Item 1	22
5.13	Conclusion	22
6	Testing	24
6.1	Introduction.....	24
6.2	Functional Testing	24
6.2.1	Navigation	24
6.2.2	Car Tests.....	25
7	Project Management.....	27
7.1	Introduction.....	27
7.2	Project Phases	27
7.2.1	Proposal	27
7.2.2	Requirements.....	27
7.2.3	Design.....	27

7.2.4	Implementation	28
7.2.5	Testing.....	28
7.3	SCRUM Methodology	28
7.4	Reflection	28
7.4.1	Your views on the project.....	28
7.4.2	Completing a large software development project.....	29
7.4.3	Working with a supervisor	29
7.4.4	Technical skills.....	29
7.4.5	Further competencies and skills	30
7.5	Conclusion	30
8	Conclusion	31
	References	33

1 Introduction

The overall aim of my project is to have a presentable and fully functional racing game by the end of the academic year. This game should have multiplayer functionality and use artificial intelligence for non-player characters (NPC's) that the user can play against. Racing is a hugely popular genre in the video game industry. However, most games released nowadays have lifelike graphics and physics, cost upwards of €50 per title, are developed by massive companies and often contain hidden costs in the form of downloadable content that the user must pay extra for. This gives consumers the feeling that the price of the game doesn't include everything the game has to offer.

In order to develop this game, I will mainly use the Unity game engine, which runs C# code that will be written using Visual Studio Code. Designing for the game will be done using a variety of programs. Blender will be used to create the 3D assets. It's free, open source and has a large community of people creating content to teach people how to use it. I also have a personal passion for creating models in Blender, so I have a little bit of experience in it. I'll use Figma and Affinity Designer for creating the 2D assets such as buttons, logos, speedometers, and other UI elements. I've gained experience in using both these apps through other modules in Creative Computing. I will also need to use frameworks such as Photon and Steamworks for the multiplayer aspect of the game. Photon is a widely used, real-time multiplayer framework for game development and Steamworks is a set of tools and services to help developers integrate their games with the Steam platform. Photon will be the basis for the multiplayer in my game and, when implemented, Steamworks will allow players to easily play my game with the friends they've connected with on Steam.

2 Research

2.1 Introduction

I did research into artificial intelligence in games as part of my project. There are three main ways to use AI in a racing game. Those are waypoints and vector calculations, machine learning and neural networks. I explored all these options when considering what to use for my game.

2.2 Waypoints and Vector Calculations

One of the easiest ways to implement AI in a racing game is waypoints and vector calculations. This technique involves placing a series of invisible points around the racetrack in 3D space. These act like a path that the AI driver follows. The AI checks its position on each frame and moves towards the current waypoint in the list. Once its close enough to that waypoint it will switch its target to be the next waypoint in the list. This process is repeated until the AI completes one circuit of the track, where it while either go back to the first waypoint in the list or end the race.

To actually move the car the system calculates the direction vector between the car's current position and the target waypoint. That vector is used to steer and accelerate the car in the right direction. However, one issue with this approach is that it can sometimes look robotic or unnatural. The AI either fully accelerates or fully brakes based on conditions, and that can result in jerky or overly perfect movement if not tuned properly. This can be fixed by adding conditions for the AI to follow. For example, if the AI is coming up to a sharp corner, it slows down earlier and steers gradually instead of snapping its steering to the next waypoint.

This system doesn't rely on heavy computation or learning, so it's great for performance, especially since I want my game to run well on lower-end systems too. Even though it's simple, it gives me enough control to fine-tune how the AI behaves on different tracks.

2.3 Machine Learning

There are many ways in which machine learning is used to create an AI driver, however there are three main types, reinforcement learning, supervised learning and unsupervised learning. All of these have positives and negatives. Reinforcement learning works by giving the AI a reward based on how well it performs the task that it was assigned. In this case the task would be driving around a racecourse in the fastest time.

Imitation learning is a type of supervised learning in which the AI is fed demonstrations such as real human driving and uses that data to inform its own actions. A key detail when training this AI is that the source its learning from must be from a control scenario where the trainer does not know they are being observed. Otherwise, the trainer may drive more carefully, or exaggerate certain aspects of their AI, which would create a bias.

Unlike supervised learning, unsupervised learning is trained on data that has no predefined labels or categories. The AI must figure out the patterns and structures present in the data itself. In a racing game this is implemented by mimicking player behaviour such as driving styles and patterns, dynamic difficulty and creating new AI behaviours. The AI can adapt to the way the player drives because it's not working towards a preprogrammed goal like achieving the fast time or most points.

2.4 Artificial Neural Networks

A neural network is a form of artificial intelligence that allows computers to process data similarly to how the human brain does. This allows it to quickly make complex decisions on things like track layout, player actions, entry and exit points, driving lines and apex speeds. A neural network used for racing games can be trained to optimize acceleration, braking, overtaking etc. This helps when the playing is driving against multiple AI opponents because it requires the bots to not only factor in the players movements but also the actions of every other bot in the race. The biggest downside to this method is that it uses a large amount of computational power and needs extensive training, which is unsuitable for many systems.

3 Requirements

3.1 Introduction

To define the requirements for the racing game, inspiration was drawn from a range of existing titles such as *Forza Horizon*, *Gran Turismo*, and *Art of Rally*. While *Forza Horizon* and *Gran Turismo* influenced the structure and flow of racing gameplay, *Art of Rally* provided a strong reference for the stylized, minimalist visual approach. These games were analysed in terms of gameplay mechanics, visual design, and user interface. This process helped shape the functional and non-functional requirements of the project.

3.2 Requirements gathering

3.2.1 Forza Horizon

Forza Horizon is an open-world racing game series known for its expansive environments, large vehicle selection, and balance between arcade-style fun and realistic driving physics. It features both single-player and multiplayer modes, with



seamless online integration. It is highly polished with responsive controls, detailed environments and an extreme level of detail when it comes to the cars themselves. The engine sounds are recorded for each car in the real world. The graphics on the car models are almost life-like. Everything from the interiors to the engine bays are modelled realistically. Unfortunately, all these positive aspects of the game make it unsuitable for low-end systems and gives it a massive download size. The complexity of the cars upgrade system means that there is a large learning curve for new players.

3.2.2 Gran Turismo

Gran Turismo is a long-running racing simulator focused on precision, realistic physics, and authentic driving experiences. It appeals to players who are interested in learning real-world driving techniques and car performance. *Gran Turismo* is similar to *Forza Horizon*; in that they are both realistic racing games with a wide variety of vehicles and realistic physics.



Gran Turismo, however, focuses more on technical fidelity and is more of a racing simulator. The downside is that it is less accessible to casual players due to the steep learning curve and the simulation physics, which also limits its stylization.

3.2.3 Art of Rally

Art of Rally is a stylized top-down rally racing game with minimalist, low-poly graphics and a strong emphasis on atmosphere. It blends simple visuals with tight handling and challenging tracks, offering a unique take on the racing genre. It has a strong design language with a low-poly, stylized aesthetic. This means that it runs and performs well on any system. The simple UI and controls make it accessible for any player. The arcade physics ensures the focus of the game is



more on enjoyment than realism. There is however a limited track list and vehicle choice as well as customization of each of those. There are also fewer game modes compared to the larger titles.

3.3 Requirements modelling

3.3.1 Functional requirements

These functional requirements are necessary for the game to run as intended. They define the minimum of what the game should do and focus on core gameplay features, multiplayer functionality and artificially intelligent drivers.

- Players must be able to control the car using keyboard or controller inputs.
- The game must support at least two players in a local multiplayer session over LAN
- The games user-interface must display race positions, lap count, race time, lap time and speed.
- The game needs a menu system that allows players to choose races and cars as well as host and join multiplayer sessions.
- Artificial intelligence should be able to make a full lap around each track smoothly and react to obstacles on the road

3.3.2 Non-functional requirements

Non-functional requirements determine the quality and constraints of the game such as performance and usability expectations as well as the visuals of the game.

- The game must maintain at least thirty frames per second on low-end hardware
- All menus and UI need to be easily readable and accessible to all types of users
- The visual style should be consistent throughout the game and match the intended
- Multiplayer performance should be reliable over local networks without any noticeable lag or desynchronization

3.4 Feasibility

The project is considered feasible within the constraints of time, resources, and technical skill. The tools chosen for development are accessible, well-supported, and align with the scope of the project. Unity was selected as the primary game engine due to its strong community support, cross-platform capabilities, and built-in tools for physics, animation, and UI. Unity also integrates smoothly with external tools, enabling a streamlined workflow.

Blender was used to create custom 3D models, allowing full control over the visual style and ensuring consistency with the intended cartoony aesthetic. For 2D assets and interface design, Affinity Designer and Figma were used to design lightweight, readable UI elements that enhance usability and match the game's visual direction.

For implementing multiplayer functionality, Photon Unity Networking (PUN) was initially considered, but Unity NetCode for GameObjects (NGO) was ultimately chosen for tighter integration with Unity and better control over synchronization in local multiplayer scenarios. However, the experience with Photon remains a viable option for future online features.

The development was undertaken on readily available hardware with software licenses that are free or affordable for students. The lightweight visual style also ensured performance feasibility across a wide range of devices. Overall, the toolset and scope were carefully selected to match the available resources and skills, making the project achievable within the academic timeline.

3.5 Conclusion

The requirements outlined in this section were informed by a combination of industry research, analysis of existing racing games, and practical considerations for development. By studying games like *Forza Horizon*, *Gran Turismo*, and *Art of Rally*, I was able to identify key gameplay and design elements that align with my game's goals. The functional and non-functional requirements reflect the core features needed to deliver a fun, accessible multiplayer racing experience, while also considering technical constraints such as hardware performance and network reliability. Together, these requirements provide a clear foundation for development and ensure that the final product meets both player expectations and project goals.

4 Design

4.1 Introduction

The design phase focuses on planning and structuring the key components of the game before full implementation begins. This includes defining the visual style, gameplay mechanics, user interface, and technical architecture needed to support both single and multiplayer functionality. Given the project's low-poly aesthetic and focus on accessibility, the design prioritizes clarity, simplicity, and performance. The phase also involves creating visual assets, designing levels, and outlining how various systems—such as car physics, lap tracking, and network synchronization—will interact within the Unity engine. This section documents the design choices made and how they align with the requirements established in the previous chapter.

4.2 Program Design

The game is structured around Unity's scene and GameObject system. Key scenes include the Main Menu, and Racetrack. I have a separate scene for each racetrack available in the game. Each scene contains relevant GameObjects, such as UI elements, player cars, environment assets, and system managers. Prefabs are used extensively for modularity and consistency, especially for repeatable elements like player vehicles, checkpoints, and UI panels.

Unity's component-based model allows behaviours to be added via scripts, which are attached to GameObjects. For example, player inputs, movement and physics handling are implemented in a Drive script attached to each player vehicle. There is also an AI Controller that references the Drive script, meaning that the Drive scripts receive inputs from the AI Controller that tells the car how to behave. For the player cars the Drive script gets its inputs from a Player Controller script that takes the players inputs.

4.2.1 Technologies

The technologies being used to create this application are:

- Unity (Game Engine)

- C# (Scripting Language)
- Blender (3D Modelling)
- Affinity Designer (2D Art & UI Assets)
- Figma (UI/UX Design & Prototyping)
- Photon Unity Networking (Multiplayer Networking)

These technologies were chosen because they provide a strong foundation for developing a stylized, multiplayer racing game. Unity offers a comprehensive and flexible game development environment with built-in support for 3D physics, animation, and user interface design. C# is Unity's primary scripting language and is well-documented and widely used in game development. Blender enables the creation of custom 3D models tailored to the cartoony art style of the game. For visual and interface design, Affinity Designer and Figma were used to produce lightweight, clean, and consistent UI elements. Photon Unity Networking (PUN) was selected for its ease of use, real-time synchronization capabilities, and reliable cloud-based multiplayer architecture, making it ideal for implementing peer-to-peer and online gameplay.

Other possible technologies which could have been used were Unreal Engine, Unity NetCode for GameObjects, or Adobe Illustrator. These technologies were not suitable because Unreal Engine is more geared toward high-end photorealistic games and can be unnecessarily complex for an indie-styled racing game. Adobe Illustrator, while industry-standard, was avoided due to cost, and Affinity Designer provided similar capabilities with a more affordable license. These alternatives are more appropriate for larger teams or games with different technical and visual requirements.

4.2.2 Structure of Unity

Unity follows a component-based architecture where all in-game objects are built using GameObjects and components. Each object in a scene is made up of a combination of scripts, colliders, renderers, and other modules that define how it behaves or appears. Unity projects are organized into a structured folder system within the Assets directory, which acts as the central hub for all game resources.

Key Folders in the Project:

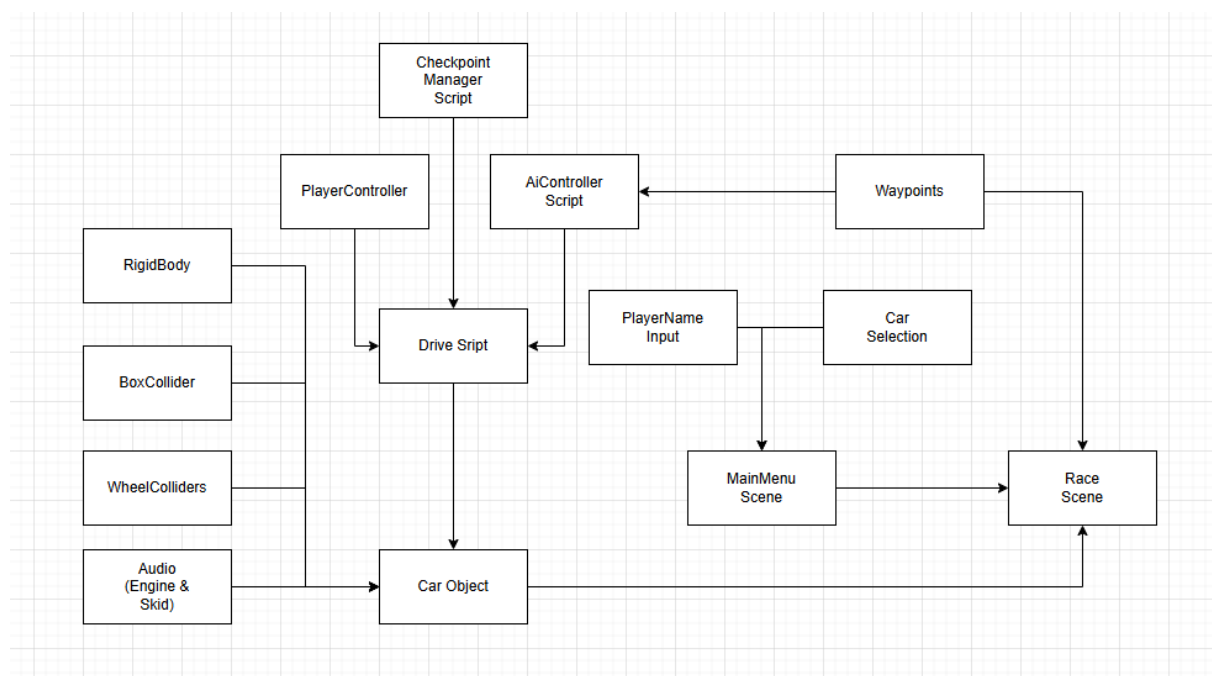
- Assets/ - The main directory containing all the game files and resources.
 - Scenes/ - Stores different Unity scenes (e.g. Main Menu, Lobby, Racetrack).
 - Scripts/ - Contains all C# scripts that define game logic (e.g. Car Controller, Race Manager).
 - Prefabs/ - Holds reusable GameObjects like cars, checkpoints, and UI panels.
 - Materials/ - Contains the materials used for objects in the game (colours, textures).
 - UI/ - Includes canvases, images, buttons, and layouts for the user interface.
 - Photon/ - Contains Photon-specific assets, prefabs, and configuration scripts for multiplayer.
 - Models/ - 3D models imported from Blender for cars and environment assets.
 - Audio/ - Stores sound effects and music used in the game.
 - Animations/ - Holds animation clips and controllers if used for UI or objects.

Unity Hierarchy and Flow:

- Scenes: Each game level or menu is saved as a .unity scene file. Scenes hold all the GameObjects active during that part of the game.
- GameObjects: The basic building blocks of Unity, used for everything from cars and cameras to menus and checkpoints.
- Components: Behaviours or properties attached to GameObjects - e.g., Rigidbody for physics, Collider for interaction, custom scripts for game logic.
- Scripts: Written in C#, scripts control logic such as movement, UI updates, multiplayer syncing, and race progression.

4.2.3 Application architecture

The architecture of the game is explained in the diagram below. The car object has the necessary components for it to function, those being the rigidbody, and colliders. It also has player controller, AI controller, checkpoint and drive script. The drive script has values and methods passed down to it because it handles the actual physics and forces that drive the car. The AI controller has the waypoints for the race passed into it. These waypoints are game objects in the 3D space of the racetrack. The car object is then placed in the race scene where the player can control it or the AI drives it. Before the race scene, the main menu takes in the players name and car selection to then pass to the race scene where they are displayed.



4.3 User interface design

The UI design consists of the heads-up display on the car and the main menu. I designed them in Figma. I tried to make them look like a sign and be flat because it fits with the games simple theme and aesthetic.



4.3.1 Level Design

I made and textured the racetracks in Blender using data from google earth to get accurate height values. I didn't make it at accurate scale but instead made the tracks wider in accordance with the size of the cars that I had modelled.

The textures I made were just simple block colours because I wanted to keep the aesthetic and feel of the game.

4.4 Conclusion

The design phase paid out the groundwork for creating an efficient racing game that focus on performance and visual consistency. Key design decisions such as using component-based architecture in Unity, low poly visuals and modular prefabs allowed for streamlined development and scalability. By planning out the key systems like artificial intelligence, car physics and user interface early on, the implementation phase was smooth and focused. The design aligns with the project's goals of stylized visuals and engaging gameplay while also supporting good performance.

5 Implementation

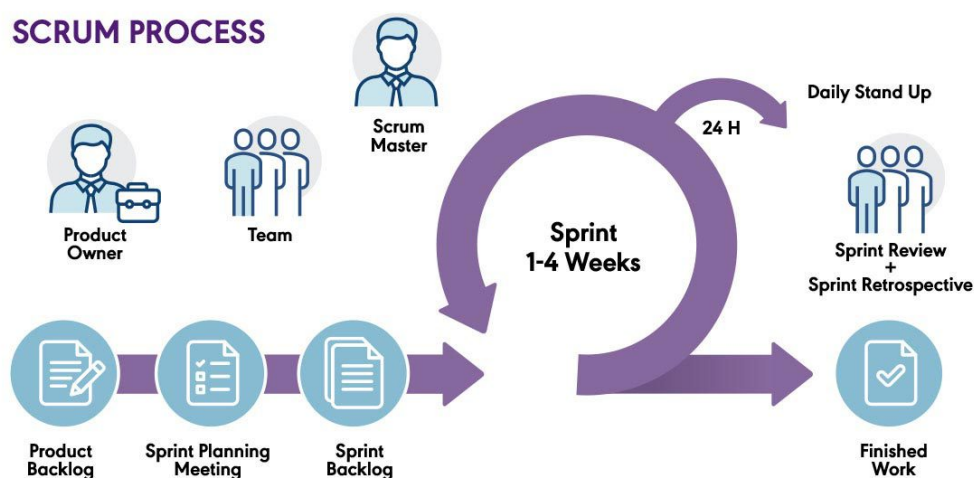
5.1 Introduction

This section outlines the process of bringing the game design to life within Unity. It details how the planned systems, assets, and interfaces were built and integrated to form a fully functional racing game. Each part of the implementation, from car movement and race logic to multiplayer setup and UI integration, was developed using Unity's tools and scripting in C#. The implementation phase focused on translating design concepts into interactive features while maintaining performance, consistency, and the intended visual style. This section also highlights the challenges faced during development and how they were resolved through testing, iteration, and adaptation.

5.2 Scrum Methodology

For the implementation of this project, the SCRUM methodology was adopted as an agile framework to manage development tasks efficiently and stay organized throughout the academic year. SCRUM emphasizes iterative progress through Each sprint concluded with a review and reflection on what was completed, what could be improved, and what the next steps would be.

The use of SCRUM also helped manage risk by allowing features to be developed incrementally rather than all at once. For example, early sprints focused on core gameplay elements like physics and controls, while later sprints added layers of polish such as UI



refinement and multiplayer support. This iterative development process ensured that the project was always in a playable state and helped identify bugs or issues early on.

The methodology also provided a framework for reflecting on progress and making informed decisions. Features were frequently tested and adjusted based on feedback or technical constraints. By breaking the project into smaller, manageable pieces and continuously evaluating the results, SCRUM allowed the project to evolve organically while still meeting the original goals. Overall, it proved to be an effective method for managing a solo game development project within the scope of an academic timeline.

5.3 Development environment

The development of the game took place primarily in Unity, using Visual Studio as the integrated development environment (IDE) for writing and debugging C# scripts. Unity was chosen for its wide feature set, real-time editor feedback, and excellent support for both 3D game development and multiplayer functionality through third-party frameworks like Photon Unity Networking.

All game assets, scripts, and scenes were managed within Unity's editor interface, allowing for rapid prototyping and testing. Visual Studio provided syntax highlighting, error checking, and deep integration with Unity, which made it easier to write and debug scripts efficiently. The Unity Console was used frequently during testing to log output, monitor performance, and catch any runtime errors or warnings.

Version control was handled using Git through GitHub, which allowed for regular commits, backups, and version tracking. This also made it easier to roll back changes when bugs were introduced or when experimental features needed to be removed. GitHub Issues and commit messages were also used to keep track of completed tasks and bugs found during testing.

Additionally, Trello was used to organize tasks in a kanban-style board, reflecting the SCRUM methodology used throughout development. Each card represented a task, such as implementing a UI feature or designing a new track segment and was moved from "To Do" to "In Progress" to "Done" as the project progressed. This visual task management helped maintain focus and ensure a clear record of what was completed in each sprint.

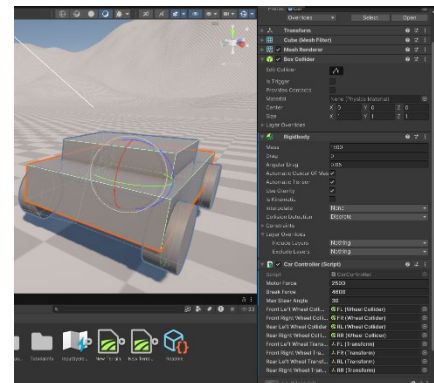
5.4 Sprint 1

5.4.1 Goal

The goal for sprint one was to create a foundation from which I could further develop the racing game. I did this by implementing basic car movement and making a simple testing environment to test the driving mechanics.

5.4.2 Item 1

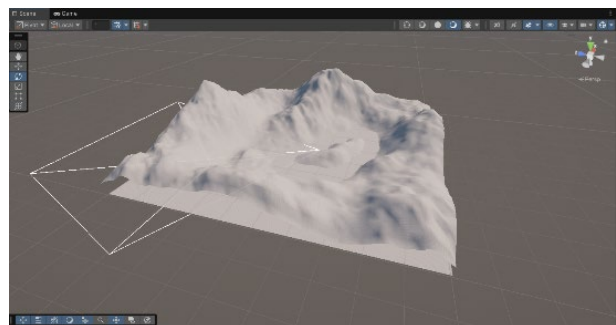
I created a Car Controller script using Unity's rigidbody and wheel colliders. The player can steer, accelerate, brake and reverse using keyboard controls. Forces and torque are applied to the rigidbody to simulate an arcade-style driving experience. The simple



The wheel transforms are separate objects to the body of the car. I had trouble with getting them to rotate correctly because the rotation is calculated in the script and is linked to the wheel colliders. I fixed this issue by tweaking the values and axis from which to rotate the wheels and constantly testing the code.

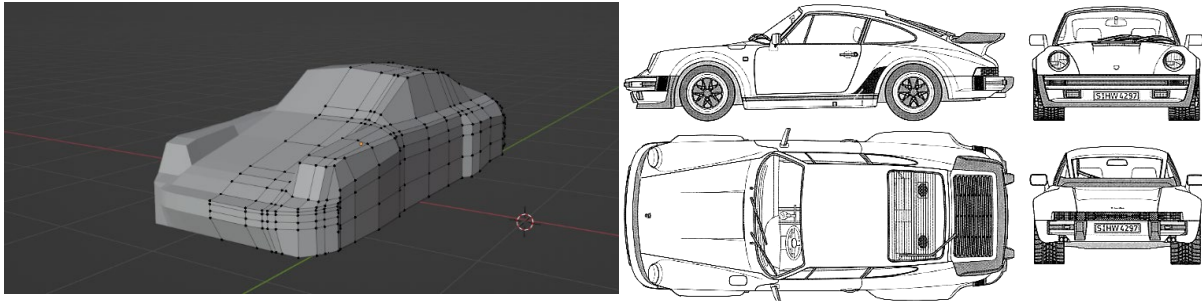
5.4.3 Item 2

The second item for this sprint was to create a temporary environment to test the Car Controller. I did this with Unity's terrain object. I used one terrain object as a plain flat surface that would act as road to drive on and elevated it in certain areas to test how the car responds to inclines.



5.4.4 Item 3

I also started using Blender to model a simple car that I can use in the game. I used a blueprint as a reference and modelled my car around that. Following tutorials, I found on YouTube helped greatly with the process of learning Blender.



5.5 Sprint 2

5.5.1 Goal

The goal for this sprint was to further develop the controls of the car as well as its physics and keep practicing and improving my skills in Blender. I also started modelling a couple racetracks in Blender that I can use in the game.

5.5.2 Item 1

At the beginning of the project, I used a single script called CarController.cs to manage everything related to the player's car — input handling, physics, acceleration, steering, and braking. This worked fine for quick prototyping, but as the game grew more complex, I started to notice that the script was becoming bloated and difficult to manage. It became clear that I needed a cleaner and more modular approach, especially as I began planning for AI-controlled cars and multiplayer functionality.

To solve this, I decided to split the original script into two parts:

Drive.cs and PlayerController.cs.

The Drive script is now responsible for handling all the physics-based movement like applying force to the rigidbody, turning the wheels, and simulating acceleration.

Meanwhile, the Player Controller script is used to capture the

player's input and feed it into the Drive script. This separation made it much easier to debug issues and to eventually plug in different types of input sources — whether it's a human player or an AI.

```
public void Go(float accel, float steer, float brake)
{
    // Make sure input values are within valid ranges
    accel = Mathf.Clamp(accel, -1, 1);
    steer = Mathf.Clamp(steer, -1, 1) * maxSteerAngle;
    brake = Mathf.Clamp(brake, 0, 1) * maxBrakeTorque;

    // Calculate how much power to give the wheels
    float thrustTorque = 0;
    if(currentSpeed < maxSpeed)
        thrustTorque = accel * torque;
    else
        thrustTorque = accel * torque * (maxSpeed/currentSpeed); //Cars acceleration decreases as it approaches max speed

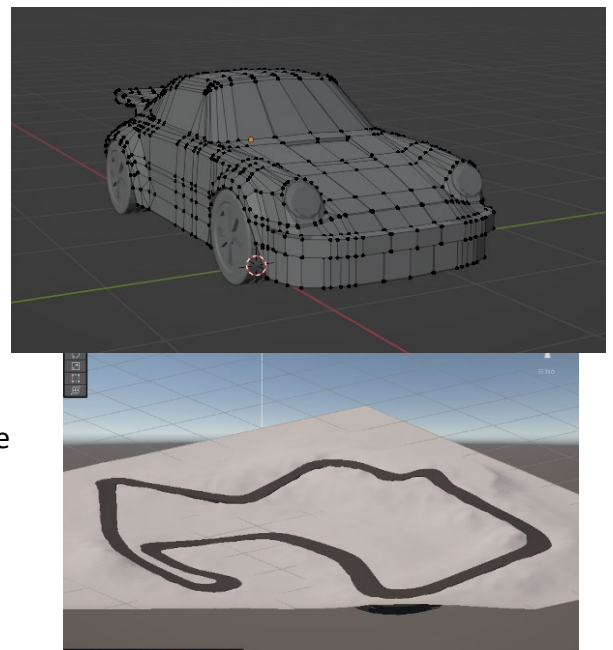
    // Update each wheel
    for (int i = 0; i < 4; i++)
    {
        // Apply power to the wheel
        WCs[i].motorTorque = thrustTorque;

        // Front wheels can steer
        if (i < 2)
            WCs[i].steerAngle = steer;
        // Back wheels can brake
        else
            WCs[i].brakeTorque = brake;

        // Update the visual position and rotation of the wheel model
        Quaternion quat;
        Vector3 position;
        WCs[i].GetWorldPose(out position, out quat);
        Wheels[i].transform.position = position;
        Wheels[i].transform.localRotation = quat;
    }
}
```

5.5.3 Item 2

I improved my Blender models and started working on a racetrack model. I used a Blender add-on to find a racetrack on google earth and retrieve the terrain data. I then used that to create the track model as accurate as possible. I had issues with the car model when importing to unity because the cars orientations weren't the same In Unity. In Blender, the z-axis is up, and the y-axis is the forward axis. In Unity it's the inverse, so I had to account for this when importing my assets to Unity.



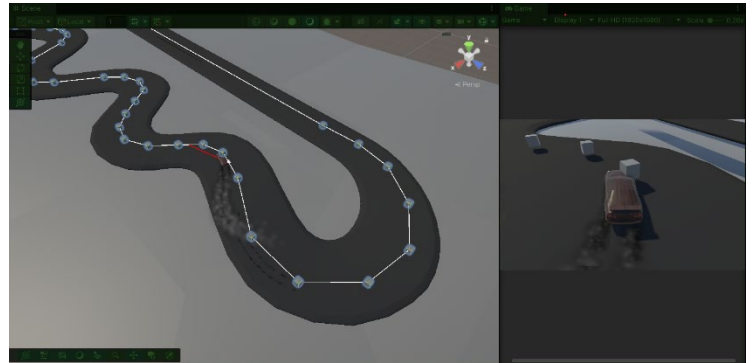
5.6 Sprint 3

5.6.1 Goal

The goal for this sprint was to start working on the artificial intelligence system and keep working on my Blender assets.

5.6.2 Item 1

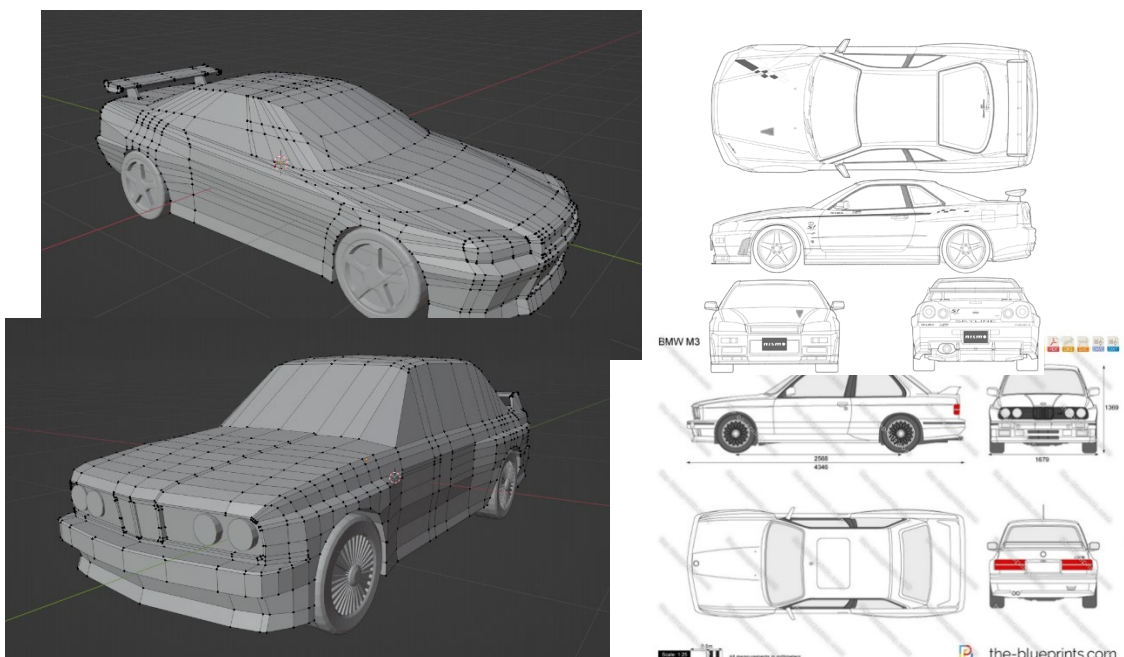
For the AI in the game, I used a simple waypoints model. The car accelerates and keeps the steering pointed at the current waypoint. The brake is also applied as the car gets closer to the waypoint. This prevents the car from



skidding too far if there's a sharp turn but has the unfortunate effect of also causing the car to break when there's no need to. To combat this, I made it so that the AI keeps track of the next waypoint as well as the current waypoint. If the angle to the next waypoint is greater than 20 degrees, for example, the acceleration will be set to 0 and the brake will be set to 1 until the upcoming angle is below 20 degrees.

5.6.3 Item 2

I also used this sprint to create more car models, making sure to keep in mind that I need to set the up and forward axis to work in Unity. I tried to keep the models simple while keeping them recognizable as the cars they're meant to represent.



5.7 Sprint 4

5.7.1 Goal

The goal of this sprint was to further develop the AI so that it moves more smoothly around the track and deal with what happens when the car gets stuck.

5.7.2 Item 1

With the way I was doing the waypoints, the car wouldn't steer to the next waypoint until it reaches the current one. To fix this I added a tracker in front of the car that the car will steer to. This way the, the tracker will start moving to the next waypoint before the car and the car will dynamically steer to the next waypoint. I also updated the braking and acceleration. Instead of them being solid values of 1 and 0, the force of each will increase or decrease incrementally based on how sharp the corner is and how fast the car is going. I also added a respawn system so that if the car gets stuck it will respawn at the last waypoint that it passed and reset the cars rotation if it is flipped upside down. I also added a ghosting script so that when the car respawns, it will has collisions with other cars turned off and the cars material will flash transparent for a few seconds.

```
void ResetLayer()
{
    ds.rb.gameObject.layer = 0;
    this.GetComponent<Ghost>().enabled = false;
}

void RightCar()
{
    this.transform.position += Vector3.up;
    this.transform.rotation = Quaternion.LookRotation(this.transform.forward);
}

// Update is called once per frame
void Update()
{
    if (transform.up.y > 0.5f || rb.linearVelocity.magnitude > 1)
    {
        lastTimeChecked = Time.time;
    }

    if (Time.time > lastTimeChecked + 3)
    {
        RightCar();
    }
}
```

5.8 Sprint 5

5.8.1 Goal

For this sprint I needed to make the respawn system work with player cars. I also needed to add engine sounds and skidding sounds to the car.

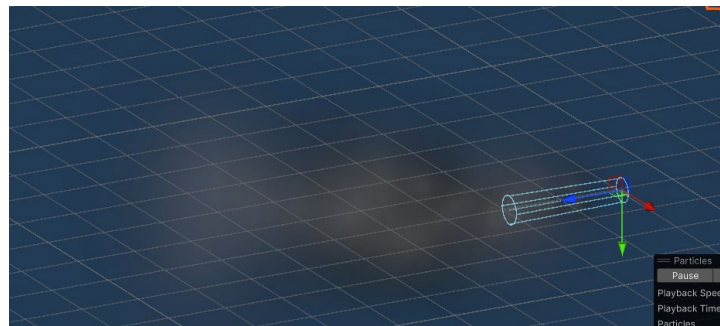
5.8.2 Item 1

To make the car respawn, I couldn't use the same code as I did for the AI because the player doesn't access the waypoints. So, I created a checkpoint system. The checkpoints followed the same course as the waypoints, but they also had collision triggers on them so that the player car would know which checkpoint it's on. When the player car got stuck it would be reset to the centre of the last checkpoint the player passed. I also changed the AI code to use this checkpoint system to maintain consistency in the gameplay.

5.8.3 Item 2

I found a couple engine and tyre skid sounds and applied them to the car. These were reliant on the drive script, so they were not affected by the player controller or the AI controller. The engine sound was played on awake and looped for the duration of the race level. The engine sounds pitch is raised or lowered as the car speeds up or slows down to give the illusion of changing gears.

The skid sound relies on the wheel colliders, which have forward and sideways slip values. If the values for forward and sideways slip are greater than or equal to a certain value, the skid sound will play. I



also added a smoke prefab and a skid trail that is positioned on each wheel and will emit under the same conditions.

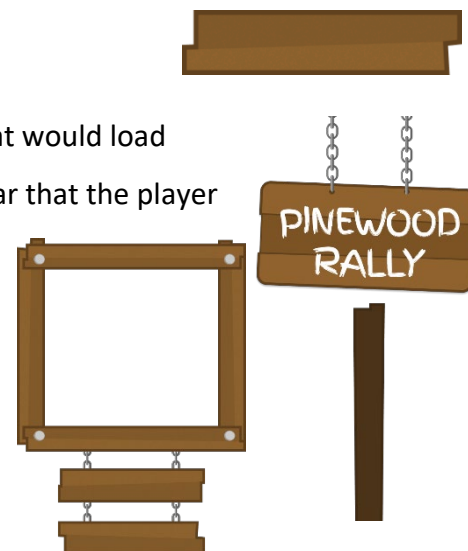
5.9 Sprint 6

5.9.1 Goal

This sprint, I started to create the main menu scene, the first scene that would load when the game is launched. I also wanted to add a way to select the car that the player would be racing with.

5.9.2 Item 1

I created my own main menu assets such as buttons and the background using Figma. I had to make sure the individual components had no background. Importing and using them in Unity was a simple and straightforward process. There's also a player name input field in the main menu. The name that the player inputs is saved into Unity's PlayerPrefs so that the player doesn't have to input their name each time.



5.9.3 Item 2

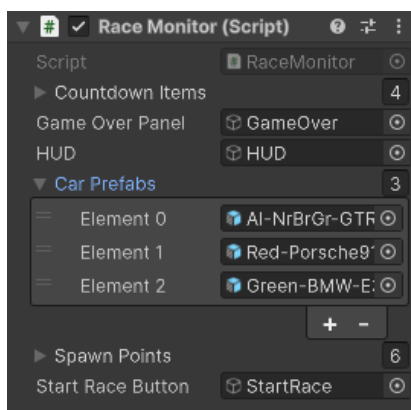
For selecting the car, I made a rotating display with each of the car choices. The cars were placed in a circle around a central camera. Using “A” and “D” the player can look at different cars. The game remembers the car that’s currently visible and uses that in the race level. This selection is also saved into PlayerPrefs.



5.10 Sprint 7

5.10.1 Goal

Although I have car selection working in the main menu, it doesn’t relate to anything in the actual race scene. The goal of this sprint was to set up the race monitor that starts the race and load the players car correctly.



5.10.2 Item 1

The first step was to create the race monitor script. This script contains the prefabs of each car that I’ve created as well as the countdown at the start of the race and the spawn points for all the cars. When the race starts a random car prefab with the AI script enabled is spawned at each spawn point and the countdown plays. The player car has the player controller enabled, and the AI controller disabled. The race monitor also spawns the player at a random spawn point each time as well.

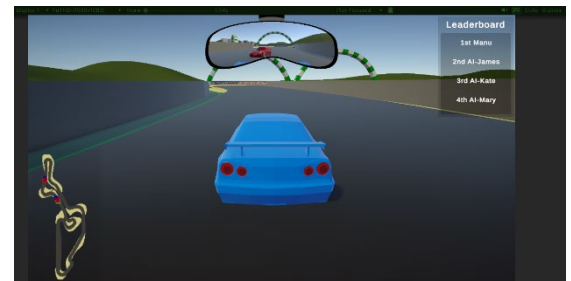
5.11 Sprint 8

5.11.1 Goal

I needed to create a heads-up display (HUD) with information that the player might need during the race. I also decided to add a rear-view mirror so the player could see any cars coming up behind them.

5.11.2 Item 1

The HUD contains a map of the track with arrows that show the location of each car and a leaderboard that displays the top four racers. It also has the rear-view mirror on it. The map uses a secondary camera that looks down on the scene. I added the arrows as objects on the cars that are on a separate layer so only the map camera can see them. I made it so that the player can enable or disable the HUD by pressing “H”.



5.11.3 Item 2

The leaderboard works off the checkpoint system. The car at the furthest checkpoint is in first place. On occasions where multiple cars are at the same checkpoint, the car that entered that checkpoint first gains the higher position in the race.

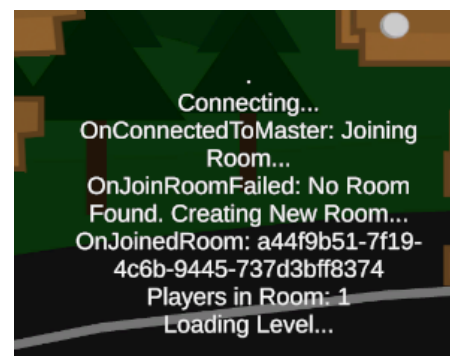
5.12 Sprint 9

5.12.1 Goal

The goal for sprint nine was to implement multiplayer functionality using the Photon engine.

5.12.2 Item 1

I managed to successfully add photon to unity and set it up so that it creates a room for the player to play in with other players. Unlike for the local game, networked players don't need the drive and controller scripts. Instead, I had to add Photons own components to each necessary game object such as the car body and wheel objects. The networked payer then received the cars position and rotation as well as any physics components such as the rigidbody. This phase was difficult to test because although Unity 6 has the capability to show multiple games running at the same time, it responded with many errors. So, I decided to build the game and use that version as the second player. This worked but it took a long time to build after each update to the code or assets.



5.13 Conclusion

To summarize this section, the game currently has two scenes, the main menu and the racetrack scene. The main menu allows the player to pick a name and car. When the player

presses the start button the game loads the next scene. In this scene the game loads car prefabs at predetermined spawn points and enables the AI controller script on any AI cars. When the race starts, a countdown plays and when that ends the player gains control of the car. The drive script holds the main physics of the car and receives values for acceleration, steering and braking from the player controller and AI controller scripts. The player has a heads-up display that shows a mini map that views the cars positions on the track, a leaderboard that displays their current position and a rear-view camera.

The AI uses a waypoint system with a tracker ahead of the AI car. The tracker follows the waypoints, and the car follows that tracker. The AI also has various statements that controls its braking and acceleration if there's a sharp corner coming up. The player controller reads the players inputs for acceleration, braking and steering and passes them to the drive script.

6 Testing

6.1 Introduction

I tested the game countless times throughout its development. I ran the game to see what differences I'd made after each new implementation of a feature. This was particularly important when laying out the waypoints and deciding what acceleration, brake values to give the AI car to make sure that the car successfully made it around the track with any major issues.

6.2 Functional Testing

6.2.1 Navigation

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Player name input	Type players name into the input field	Player's name is saved in PlayerPrefs and reloads next time the game is run	PlayerPrefs saved the name, and the input field successfully retrieved and displayed it	
2	Start race button	Click start button	Race level should be loaded	Race level loaded successfully	
3	Quit button	Click quit button	Game should close	Game closed successfully	
4	Restart button	Click restart button	Game should reload current race scene	Scene reloaded successfully	
5	Car selection	Select car using A and D	Game should load the race scene with the	Game loaded the scene with the correct car model	

			selected car as the player car		
5	Multiplayer button	Click multiplayer button	Log should show the system connecting to network and creating room, then loading scene	Log showed the intended responses and loaded scene	

6.2.2 Car Tests

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Test that the AI car follows the waypoints correctly		Car follows waypoints and completes the circuit	Car successfully followed waypoints and completed the circuit	
2	Test cars respawn points		Car should respawn at the last checkpoint	Car respawned at the last checkpoint that it passed.	
Multiplayer tests	Test that networked cars spawn into the scene		Networked cars spawn into the scene and move	Networked cars spawned and moved correctly but	

			based on player inputs	respawn and ghosting weren't working	
--	--	--	---------------------------	---	--

7 Project Management

7.1 Introduction

This chapter describes how the project was managed and how well the student kept within the project guiderails. It shows the phases of the project, going from the project idea through the requirements gathering, the specification for the project, the design, implementation and testing phases for the project. It also discusses Trello, GitHub and project member journal as tools which assist in project management.

7.2 Project Phases

7.2.1 Proposal

The initial proposal outlined the goal of building a stylized, low-poly multiplayer racing game using Unity. The idea was inspired by arcade racers and games with accessible design and fun gameplay. The scope was realistic for a solo project and focused on core gameplay features, basic AI, and local multiplayer support, with the potential for future online functionality.

7.2.2 Requirements

Requirements were gathered through research into existing racing games and their mechanics. Functional and non-functional requirements were defined based on features like car control, multiplayer functionality, and visual consistency. Tools like personas and use cases were used to guide feature priorities and usability expectations. Requirements also included performance targets, accessibility, and responsive UI.

7.2.3 Design

The design phase focused on planning the technical architecture and visual identity of the game. A modular system using Unity's prefabs and component-based structure was chosen for flexibility. Custom 3D assets were made in Blender, and the UI was prototyped using

Figma and Affinity Designer. This stage also covered level design, player HUD elements, and the waypoint system for AI.

7.2.4 Implementation

The game was implemented incrementally using SCRUM methodology. Features were developed in sprints, starting with basic car physics and movement, followed by AI, race logic, and multiplayer. Key systems like the Drive script, Player Controller, AI Controller, and UI components were built and integrated. Each sprint helped refine and build on previous work, ensuring a consistent development pace.

7.2.5 Testing

Testing took place throughout development, focusing on functional testing of car mechanics, AI behaviour, and user interface interactions. Each feature was tested after implementation to catch bugs early and ensure everything worked as expected. Multiplayer was tested in local network conditions to evaluate synchronization and lag. Informal user testing also provided valuable feedback on usability.

7.3 SCRUM Methodology

The sprint system allowed me to focus individual parts of the development process. Instead of worrying about the bigger picture. Each sprint had a clear goal, whether it was building basic car movement or refining AI behaviours, I could break it down into smaller, more manageable tasks. This process helped me catch bugs early and adjust priorities moving forward rather than doing everything all at once. Using this approach meant I could also prototype, and test features quickly and made it easier to adapt when unexpected issues came up.

7.4 Reflection

7.4.1 Your views on the project

In my opinion, the project went well. The game performs well, and no massive bugs have been found. If I were to start from the beginning again, I would have more time making

some features integrate better with each other. For example, I created a shader graph in Unity so that the game would be cel-shaded but when implemented, I found that it wouldn't work with the ghosting script and didn't appear as intended on certain parts of the car models I had created. I also tried to implement multiplayer using the Photon Engine but had issues that I couldn't fix before the deadline. Nonetheless, it gave me a fundamental understanding of how multiplayer works in games. The AI aspect of the game was very insightful and interesting, however. Between the research I did on it and the actual implementation of AI into my own game, it taught me the impact that effective game design can have on the players experience.

7.4.2 [Completing a large software development project](#)

This project taught me how important proper planning is when designing and developing large software projects. I learned to adapt when features weren't working as expected and how to stay focused on one goal, prioritizing core functionality over less essential features. I also gained experience with prototyping and testing my work based on feedback. Overall, this experience has helped me understand the full lifecycle of a major game project.

7.4.3 [Working with a supervisor](#)

Working with a supervisor was a very positive experience. I was given space to experiment with my ideas while receiving helpful feedback and direction on what I should be focused on next. Having an outside perspective on what I was building helped me stay on track and evaluate my progress realistically.

7.4.4 [Technical skills](#)

Throughout the project I improved my skills in both Unity and C#. I now have a better understanding of Unity's component-based system and of how to work with prefabs, scenes and physics systems. Even features that I failed to implement such as the shader graph and multiplayer have strengthened the skills I needed to create this game by teaching me how to use the Unity's shader graph and Photon's engine for multiplayer. I also learned how to use Blender, which I only had a very basic understanding of before undertaking the project. I can now confidently design and create a 3D model that is optimized for use in games and

can correctly export it to Unity. I particularly liked using Blender because it allowed me to explore a more creative side of my personality.

7.4.5 Further competencies and skills

In addition to technical skills, I also developed a variety of soft skills. Time management and task prioritisation were essential in completing this project. Communicating my progress through documentation and supervisor meetings were also a valuable skill. These skills will definitely help me in any future workplace environments.

7.5 Conclusion

The aim of this project was to create a low-poly racing game with multiplayer capabilities that balanced semi-realistic physics with fun gameplay. By focusing on core features like car control, AI opponents and clean UI, I was able to create an enjoyable gaming experience. Although I wasn't able to achieve all the goals of the initial proposal, the experience I gained through the process has been invaluable. Unity, C# and Blender were key technologies that supported development and allowed for efficient repetition of design and implementation phases.

Research into artificial intelligence and similar racing games played an important role in shaping the behaviours of the AI drivers, with techniques like waypoints and vector calculations helping control the AI. The project was managed using SCRUM methodology, which helped break the work up into more manageable sprints and kept development on track.

Overall, the project has given me valuable hands-on experience in game development and large-scale project planning, from design to implementation. It has also helped build both technical and soft skills that would be beneficial in a professional setting. Future improvements could include expanding the game with more tracks, enhancing the AI system and adding multiplayer functionality.

8 Conclusion

This project's main goal was to design and develop a stylized, low-poly racing game with both singleplayer and multiplayer functionality using Unity. The game aimed to strike a balance between realism and arcade style gameplay, while remaining accessible and lightweight enough to run on lower-end systems. The core features included responsive car physics, AI opponents, modular UI elements, and a flexible structure to support local multiplayer.

The technologies chosen played a crucial role in bringing the game to life. Unity provided a powerful game engine with support for 3D physics and asset management, while Blender enabled the creation of custom low-poly assets. Figma was used to design clean, intuitive UI elements. C# is the language used by Unity so gaining an understanding of it was fundamental in creating the game.

The project design was driven by research into modern racing games and artificial intelligence in games. Techniques such as waypoint-based AI and conditional braking/acceleration were implemented to simulate challenging yet fair AI opponents. The design process also included careful planning of user interface, scene flow, car selection mechanics, and race logic, all of which were structured using Unity's prefab and component-based model.

Implementation was guided by the SCRUM methodology, allowing for structured, incremental development through a series of focused sprints. Each sprint added meaningful progress, from core driving mechanics to AI systems, race logic, HUD design, and multiplayer setup. Testing was performed continuously to ensure functional correctness, smooth AI navigation, and overall gameplay reliability. From a project management perspective, dividing the workload into phases helped keep development organized and manageable.

Overall, this project gave me a deeper understanding of the game development lifecycle and taught me valuable skills that I can use in the workplace. Skills like Unity and Blender as well as planning and testing will be essential if I want to continue in the game development industry. Future improvements to this project could include refining the AI system further, polishing the visuals and UI and most importantly, fully implementing online multiplayer mechanics using frameworks like Photon.

References

1. Beirne, D. (2007). *Artificial Intelligence for Games Racing Game AI An Investigation into AI Techniques for Motorsport Simulation Games* myGameDemos. com.
<http://mygamedemos.com/Abertay/David%20Beirne%20CS%201130A%20Artificial%20Intelligence%20for%20Games%20-%20Racing%20Game%20AI.pdf>
2. Blender 4.0 Reference Manual — Blender Manual. (2024). Blender.org.
<https://docs.blender.org/manual/en/4.0/>
3. Bottino, A., Picardi, D., & Strada. (2020). *POLITECNICO DI TORINO A comparison of Different Machine Learning Techniques to Develop the AI of a Virtual Racing Game Supervisor Candidate*.
<https://webthesis.biblio.polito.it/secure/18168/1/tesi.pdf>
4. Build A Multiplayer Kart Racing Game - Unity 6 Compatible. (2025). Udemy.
<https://www.udemy.com/course/kart-racing/?couponCode=ST7MT290425G1>
5. Cechanowicz, J., Gutwin, C., Bateman, S., Mandryk, R. L., & Stavness, I. (2014). Improving player balancing in racing games. *Annual Symposium on Computer-Human Interaction in Play*. <https://doi.org/10.1145/2658537.2658701>
6. Chan, M. T., Chan, C. W., & Gelowitz, C. (2015). Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques. *International Journal of Computer Games Technology, 2015*, 1–6.
<https://doi.org/10.1155/2015/839721>
7. Funselektor Labs. (2020). *Art of Rally* [Video game]. Funselektor Labs Inc.
8. Gil-Aciron, L. A. (2022). The gamer psychology: a psychological perspective on game design and gamification. *Interactive Learning Environments*, 1–25.
<https://doi.org/10.1080/10494820.2022.2082489>
9. Jaffe, A., Miller, A., Andersen, E., Liu, Y.-E., Karlin, A., & Popovic, Z. (2012). Evaluating Competitive Game Balance with Restricted Play. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 8*(1). <https://ojs.aaai.org/index.php/AIIDE/article/view/12513>

10. JimenezBloggerFebruary 03, E., & 2009. (2009, February 3). *The Pure Advantage: Advanced Racing Game AI*. Game Developer.
<https://www.gamedeveloper.com/design/the-pure-advantage-advanced-racing-game-ai>
11. Playground Games. (2021). Forza Horizon 5 [Video game]. Xbox Game Studios.
12. Polyphony Digital. (2022). Gran Turismo 7 [Video game]. Sony Interactive Entertainment.
13. Stryker, C., & Kavlakoglu, E. (2024, August 16). *What is Artificial Intelligence (AI)?* IBM. <https://www.ibm.com/topics/artificial-intelligence>
14. Tomlinson, S., & Melder, N. (n.d.). *An Architecture Overview for AI in Racing Games*.
https://www.gameapro.com/GameAIPro/GameAIPro_Chapter38_An_Architecture_Overview_for_AI_in_Racing_Games.pdf
15. Unity Technologies. (2023). Unity - Manual: Unity 6 User Manual. Unity3d.com.
<https://docs.unity3d.com/6000.0/Documentation/Manual/UnityManual.html>