Development of a city driving videogame

Alejandro García López

Final Degree Work

Bachelor's Degree in
Video Game Design and Development

Universitat Jaume I

July 3, 2023

Supervised by: Raul Montoliu Colás, PhD.

To Wendy

Acknowledgments

I would like to thank the team at Catness where I learnt most of the things I know of Unreal Engine.

Moreover, the team of Epic Games for creating the really awesome technology for developing video games free and accessible to everyone.

I would also like to thank Raúl Montoliu Colás his suggestions and help in the process of writing this report. Finally, thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which has become very useful as a starting point and guide in writing this report.

Abstract

This project involves the creation of an immersive video game centered around driving through a bustling city, complete with realistically moving vehicles. The primary objective is to develop an advanced, rule-based intelligence system for the vehicles, ensuring their behavior mimics real-world traffic patterns. Additionally, special attention will be given to accurately replicating the movement of bicycles within the game.

At its core, this game will focus on crafting and refining an AI system that effectively manages the flow of traffic throughout the city. It will be a third-person simulation/-casual genre video game developed using the powerful Unreal Engine 5, optimized for computer platforms. The player will assume the role of a delivery person riding a bicycle and will be tasked with timely package deliveries across different areas of the city.

By combining realistic vehicle dynamics, and precise bicycle movement, players will be fully immersed in an authentic urban environment. The use of Unreal Engine 5 will ensure stunning visuals and a seamless gameplay experience. As the delivery guy, players will face the challenge of navigating through a dynamic city, dodging traffic, and delivering packages within strict time limits.

Contents

CHAPTER 1

INTRODUCTION

Contents

This chapter serves as a concise overview of the development process, outlining the objectives, motivations, and initial state of the project. Its primary focus is to provide a clear understanding of the planned work and its purpose. In a nutshell, the project consist of a design and development of a single player, 3rd person action/casual game in a city which will challenge your driving skills.

Game play video: Delivery Drift - YouTube

1.1 Game Overview

As described in the abstract, the player assumes the role of a Delivery Guy tasked with making timely deliveries. The game environment will feature highlighted green beacons showing over buildings, indicating the delivery destinations and providing a general direction to follow. However, successfully completing the deliveries requires more than just punctuality; the player must also navigate through a bustling cityscape, evading vehicles at all costs. Any collision, no matter how minor, will instantly end the game.

Figure 1.1: Delivery Drift overview

1.2 Work Motivation

One of the games that thrilled me more was Grand Theft Auto, particular y **GTA IV** [15], there was where I discovered the open world games. One of the things that I enjoyed the most was exploring the city by car. So with that idea in mind Delivery Drift is a simulation game that tries to recreate the sensation of driving thought a city. In the image 1.2, we can see the ambient inspiration.

This type of games are usually popular and attract the interest of the general public which is mainly why I decided to create this game, moreover, these type of games usually create a sensation of immersion and challenging game play. Although to create a slight distinction to other similar video games there is a small twist, the driving is done on a bike instead of a car, this idea came to my mind because I use the bike as a mean of transport, but then I realised that is more interesting to be done by bike because the speed is felt more and it's easier to pass though cars.

Figure 1.2: Delivery Drift top, GTA IV Bottom [5]

1.3 Objectives

The project's primary objectives can be summarized as follows:

- Provide players with a profound sense of immersion within a bustling cityscape and a compelling desire to explore its intricacies. This would be acquired by designing an interesting city and using highly detailed assets.
- Create a good feeling of the bicycle controls and mechanics, by creating good animations and a realistic bicycle movement.
- Design and implement all systems described in Chapter 3 (see Chapter 3).
- Make all the vehicles look realistic and interact with each other in a way that feels natural and challenging.

- Expand my knowledge of Unreal Engine, as by the start of the project, my knowledge is basic.

1.4 Environment and Initial State

The beginning of the idea was to create a complex and interconnected system which would need to be consistent in time and space scale. At the start I didn't had the knowledge to make such a big project. During the first weeks of development, I completed a very extensive tutorial, Professional Game Development in C++ and Unreal Engine by Tom Looman [19], that gave me a good starting point into understanding the basic functioning of unreal and how the different systems relate. After that the project was still a difficult challenge but with various tutorials, websites and forums to solve the future problems it was viable. Although time and resources were limited, and some ideas were left apart, the core of the game was possible.

The creation of this game is inspired by some already existing games and will rely on some assets and web pages.

- The game is inspired by the driving aspects of the popular game **Grand Theft Auto** [16], it also shares some similarities with the game **Delivery Truck Simulator** [3] and **Pizza Delivery Simulator** [14].
- There will be used some assets, the most useful and main asset for the map construction is the "City Environment Megapack vol 02" taken from the Unreal Engine Marketplace [12] and the City Samples Vehicles from the Unreal Engine Marketplace too [6].
- Sketchfab, and other internet websites alike, will provide car models and various city elements which will be incorporated into the video game.
- For the main character I'll be using the **Control Rig** from Epic Games [8]. And for the meshes the **Stylized Character Kit: Casual 01 by RocketArts** [22]

1.5 Results preview

What is going to be achieved is a game that tries to be realistic both in movement and in environment. The game play and mechanics would feel natural and entertaining. Here we can see some shots of the game and a short game play video.[1]

[1] *Game play Delivery Drift*, https://youtu.be/S7lLfCrk1bQ.

Figure 1.3: Delivery Drift title page

Figure 1.4: Delivery Drift player start

Figure 1.5: Delivery Drift in game photo

Chapter 2

Planning and resources evaluation

Contents

2.1 Planning

The main part of the game would be the delivery missions that take place in the bustling city. As a delivery guy on a bicycle, the player's objective is to deliver packages to different parts of the city within a limited time. Here's what different parts of the game will consist of:

- City Exploration: The player would have the freedom to take any route they desire to deliver the packages. The city would be filled with realistic buildings and vehicles.
- Delivery Missions: The player would have a wide range of delivery missions to complete within a limited time frame. Each mission would be different and offer varying levels of challenge. The player would need to navigate through the city, avoiding traffic to reach their destination and deliver the package on time.
- Traffic Management: The game's AI would manage the circulation of vehicles in the city, making it more challenging for the player to complete their missions. The player would need to use their knowledge of traffic rules to navigate through the city safely and efficiently.

- Rewards: When the player finishes one deliver mission he receives 1 "point", and when he/she has 5 he would be able to drive a car instead of a bike.
- Challenging Scenarios: As the player progresses through the game, they would encounter some challenging scenarios, specifically rush hour traffic. However this scenarios would appear randomly (by chance) along the game play. The player would need to learn how to adapt their strategy to complete the missions successfully.

2.2 Resource Evaluation

From the objectives described in the previous section (see Section 2.1) the next tasks can be deduced to accomplish the requirements (see Figure 2.1)

And from the information in the task table the next Gantt chart is done (see Figure 2.2)

2.2.1 Tools

For the development of this particular game, there will be used a variety of tools and technologies to create a dynamic and immersive game play experience. 2.1 , 2.2, 2.3 , 2.4 and 2.5. The games has been developed in my personal PC with the next specifications:

- OS: Windows 11 Home.
- Processor: Intel i712700
- GPU: Nvidia RTX 3060 Ti
- RAM: 32 GB DDR4

Type:	Engine
Specific software name:	Unreal Engine 5
It offers a range of tools and features that make it easy to develop complex game play mechanics and realistic graphics. Some of the systems that will be used will be Lumen, Behaviour Trees and Environment Queries	

Table 2.1: Software programs «TOOL1. UE5»

Type:	Development Environment
Specific software name:	JetBrains Rider
Efficient and powerful development environment that can help streamline the process of creating a complex and immersive video game. Used to write and debug code for the game's mechanics, AI, and other features.	

Table 2.2: Software programs «TOOL2. Rider»

Type:	3D Modeling
Specific software name:	Blender
It will be used if necessary, just in case there isn't a specific 3D model in free assets.	

Table 2.3: Software programs «TOOL4. Blender»

Type:	Image Editor
Specific software name:	Photoshop
For menus and UI. Textures too if I can't find them in free assets or want to touch up some of them.	

Table 2.4: Software programs «TOOL5. Photoshop»

Type:	Project Management Tools
Specific software name:	Trello / Excel
Keep track of tasks, timelines, and goals for each aspect of the game's development. These tools will help ensure the project stays on schedule and on budget.	

Table 2.5: Software programs «TOOL6. Trello»

Type:	Project report
Specific software name:	Overleaf - LaTeX
Represent all the work done using LaTeX	

Table 2.6: Software programs «TOOL6. Overleaf»

Task	Estimated Time (Hours)	
Learn Unreal Engine 5	50	h
Base project	25	h
Fundamental behavior of vehicles	20	h
Acceleration and deceleration	10	h
One-way roads	14	h
Non grid roads	14	h
Traffic lights	20	h
More than one vehicle	23	h
More than one lane roads	20	h
Cyclist (player)	20	h
Car (player)	5	h
Package delivery	4	h
Map Creation	20	h
Final touches	5	h
Tutoring sessions	6	h
FDG report	35	h
Evaluation presentation	9	h
Total time:	300	h

Figure 2.1: Planning of the tasks (made with Excel)

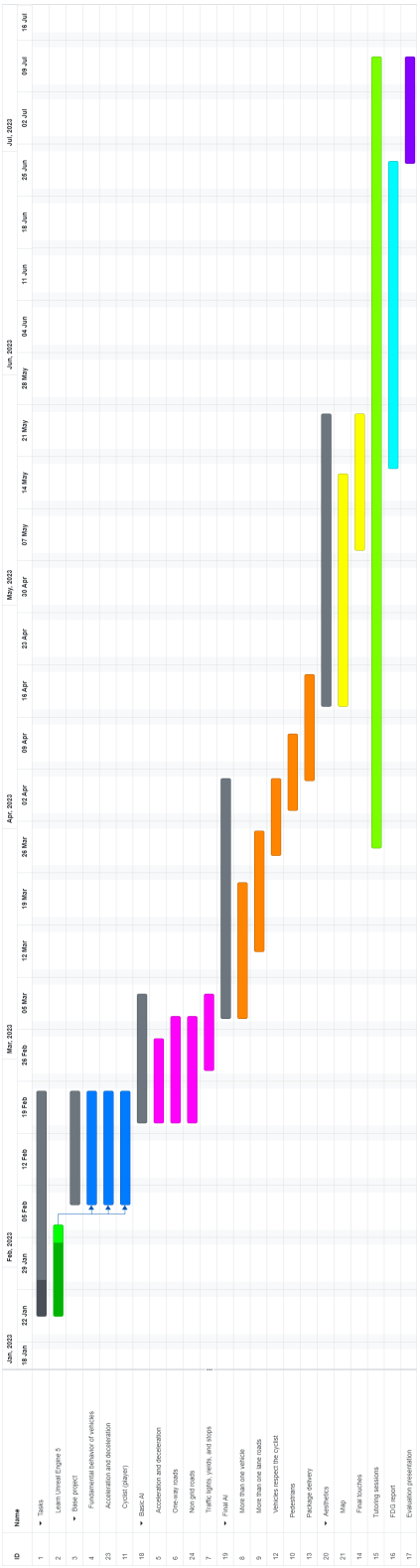


Figure 2.2: Planning of the tasks (made with OnlineGantt.com)

CHAPTER 3

Game Design Document

Contents

This chapter presents the requirements analysis, design and architecture of the proposed work, as well as, its interface design.

3.1 Conceptualization

General details:

Title: Delivery Drift

Genre: Simulation, Casual.

Platforms: Windows.

Age: +6

3.1.1 Game summary

Step into the shoes of a determined Delivery Guy in this thrilling game that puts you in charge of making time-sensitive deliveries. As you embark on your virtual adventure, the game's world unfurls before you, bustling with life and excitement. Your objective? To navigate through the vibrant cityscape, ensuring each package reaches its destination promptly.

As you explore the dynamic game environment, you'll notice buildings adorned with streetlights. There will be a beacon visible from all the parts of the world. This beacon serve as vital clue, revealing the locations where your deliveries must be made. They guide you with a general direction, acting as a beacon of hope amidst the chaos of the city streets.

However, punctuality alone won't guarantee your success. The city is teeming with bustling traffic, with vehicles zipping through the streets at breakneck speeds. It's not just a race against time but also a battle for survival. Each moment demands your unwavering attention as you maneuver your way through the maze of cars.

You must exhibit lightning-fast reflexes and swift decision-making skills to dodge any potential collision. A split-second delay, a minor misstep, and your journey comes to an abrupt end. No mercy is granted, as even the slightest impact will instantly trigger game over.

The stakes are high, and the pressure is intense. Will you emerge as the ultimate Delivery Guy, conquering the city's intricate web of streets and delivering each package with precision? It's a test of skill, nerve, and determination, as you navigate the treacherous urban jungle while racing against time.

Immerse yourself in this adrenaline-fueled delivery adventure, where every decision counts and each successful delivery brings you closer to triumph. Get ready to experience the thrill of the chase and the satisfaction of a job well done in this high-stakes, high-speed race against the clock.

3.1.2 Narrative context

Nowadays, the world is rapidly advancing towards a future that is both exciting and daunting. With the advent of new technologies and the rise of mega-corporations, our cities are evolving at an unprecedented pace. Self-driving cars, drones, and advanced AI systems are becoming a common sight on our streets, transforming the way we live and work. The streets are congested with all sorts of vehicles, from sleek and self-driving cars to massive trucks, all competing for space on the road.

It is in this context that Delivery Drift takes place, presenting a world that is both familiar and alien. The game's cyberpunk-inspired setting presents a vision of the future that is both captivating and terrifying, a world where corporations rule with an iron fist and the streets are congested with all manner of vehicles.

The player takes on the role of a delivery guy, working for a small courier company that is trying to make a name for itself in the cutthroat industry. The company has only one rule: deliver the package on time, no matter what. The player must navigate

through the busy streets of the city, weaving in and out of traffic, dodging obstacles, and avoiding accidents to reach the destination on time.

But at the heart of this world is the humble delivery guy, the unsung hero of the city who keeps the wheels of commerce turning. In Delivery Drift, the player takes on the role of this courier, navigating the busy streets of the city and facing the many challenges that come with the job.

Delivery Drift is more than just a game; it is a reflection of the world we live in, a world that is constantly evolving and changing. With its realistic vehicles movements, advanced AI-based traffic management, and engaging game play, the game promises to provide an exciting and immersive experience for players of all ages.

3.2 Demography

Age. The minimum age is 6 based on the minimum level of skills needed to play the game. There is no violent content (there is no blood when the player crashes) or any other legal restriction for which a child could not play the game.

Genre. The genres selected are a good choice based on the most played genres on steam, as Simulation gets 14% of all released games occupying the 3rd place and casual another 14% occupying the 4th place [18].

Cultural references. The cultural references are taken from the modern Japan style in buildings and city construction. Moreover, the overall environment features a cyberpunk style.

Types of players. The main type of player that the game will attract is The Explorer, due to the city exploration characteristic of the game, and secondly The Achiever, which due to the fact that after completing some deliveries the player will be awarded with the option to drive trough the city with a car.

3.3 Requirement Analysis

The requirements needed can be derived from the different tasks to accomplish, so to make it here is a description of what some of the task should do:

- Base project: Basic assets and initial road grid.
- Fundamental behavior of vehicles: Move the vehicle to desired destination.
- Acceleration and deceleration: Acceleration and deceleration in start, breaks and curves.
- One-way roads: Roads in which vehicles can't go and come back through the same.
- Two-Way roads: Add a variable to vehicles that could be changed to make them able to change lane.
- Non grid roads: Change the grid pattern in the roads to a more realistic one.

- Traffic lights: Adapt the velocity, stop and/or continue in traffic lights
- Automated destinations: System that assigns destinations to the vehicle.
- More than one vehicle: Interaction between vehicles and with Traffic Lights.
- Cyclist (player): Player movement on the Bicycle.
- Car (player): Player movement in the Car.
- Package delivery: Bicycle destination system.
- Map Creation: Road and city design, building placement and background decoration.
- Final touches: Improve assets, post-processing in camera, menu, scoring and saving system.

3.3.1 Functional Requirements

The subsequent functional requirements delineate the essential features and behaviors to be implemented within the project. These requirements encompass a diverse array of functionalities, encompassing aspects such as game entry, vehicle interactions, and player movements.

Moreover, it is imperative that the system engenders an immersive experience for the player. The functional requirements are comprehensively depicted in the following tables: 3.1, 3.2, 3.3, 3.4 and 3.5.

Actor:	Player
Preconditions:	Be on the Main Menu
Steps:	

- Open the Game.
- Select "Play".
- Wait the level to charge.

Table 3.1: Functional requirement «Start Game»

Actor:	Player
Preconditions:	Be on the main level
Steps:	

- Use "W" to accelerate.
- Use "D" to steer right.
- Use "A" to steer left.
- Use "S" to brake if moving forward.
- Use "S" to move backwards if not moving.
- Use the mouse to look around

Table 3.2: Functional requirement «Cyclist movement»

Actor:	Player
Preconditions:	Be on the main level
Steps:	

- Look around.
- Locate the beacon in the city.
- Drive to the destination.
- Dodge the cars.

Table 3.3: Functional requirement «Deliver package to destination»

Actor:	Car
Preconditions:	Be on a road
Steps:	

- Get the reference to the nearest road (that is a spline).
- Get the closest point in the spline.
- Once there get the point 3 meters forward.
- Drive to the point.
- If arrived to an intersection, go to the start of the selected road.
- If not in the end of the road go to step 3.

Table 3.4: Functional requirement «Drive through spline»

Actor:	Car
Preconditions:	Be on the main level
Steps:	

- Drive following the spline
- Look if the traffic light is green
- Receive the intersection overlap event.
- Get the information of the roads in the intersection.
- Pick a road to drive trough.

Table 3.5: Functional requirement «Cross Intersection»

3.3.2 Non-functional Requirements

- Fluency and efficiency in the game play of the video game, both mechanically and FPS.
- Realistic and interesting game look.
- The aesthetic will remind a Japanese cyberpunk style.
- The UI will be minimalist and simple.

3.4 System Design

- Case use diagrams (see Figure 3.1).
- Class diagrams (see Figure 3.2).
- Activities diagrams (see Figure 3.3).

3.5 System Architecture

Given that I don't have the economic means to test the game on a wide range of computer builds I have no exact method to test this. A first approach will set the minimum requirements with:

- OS: Windows 10 / Windows 11
- Processor: Intel i5 10400
- GPU: Nvidia RTX 2060 Ti
- RAM: 10 GB DDR4

3.6 Interface Design

The two HUD that will take place are the Main menu, the pause and the in-game HUD. The design will aim to be really simple and futuristic/cyberpunk. Its interesting to highlight that the pause menu will have a subtle effect of background blur while on the pause. Here we can see the mock-ups for the Main Menu (see Figure 3.4), Pause (see Figure 3.5) and HUD (see Figure 3.6).

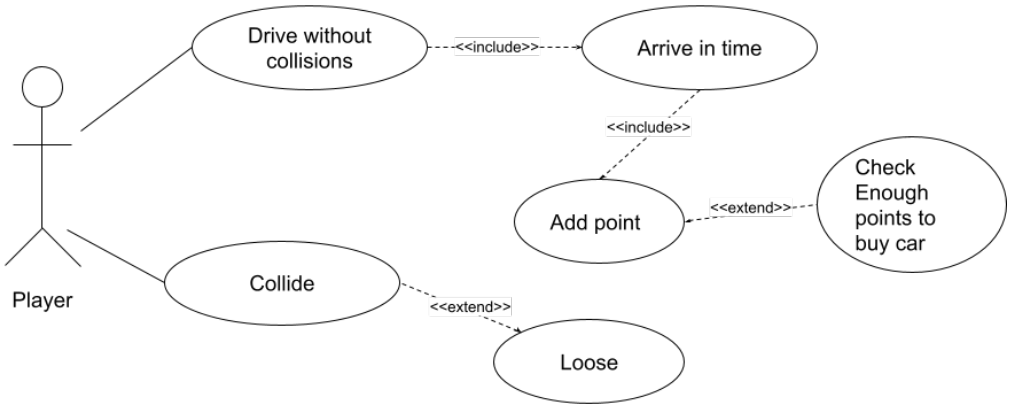


Figure 3.1: Use Cases

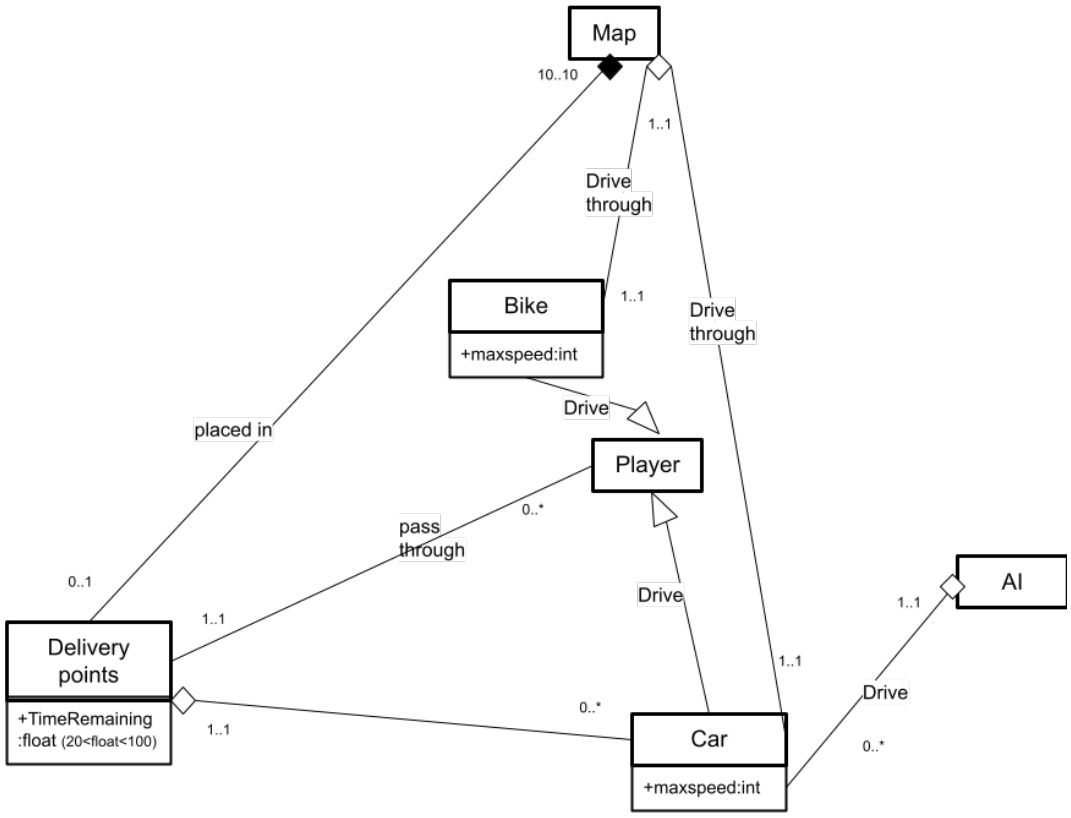


Figure 3.2: Class diagram

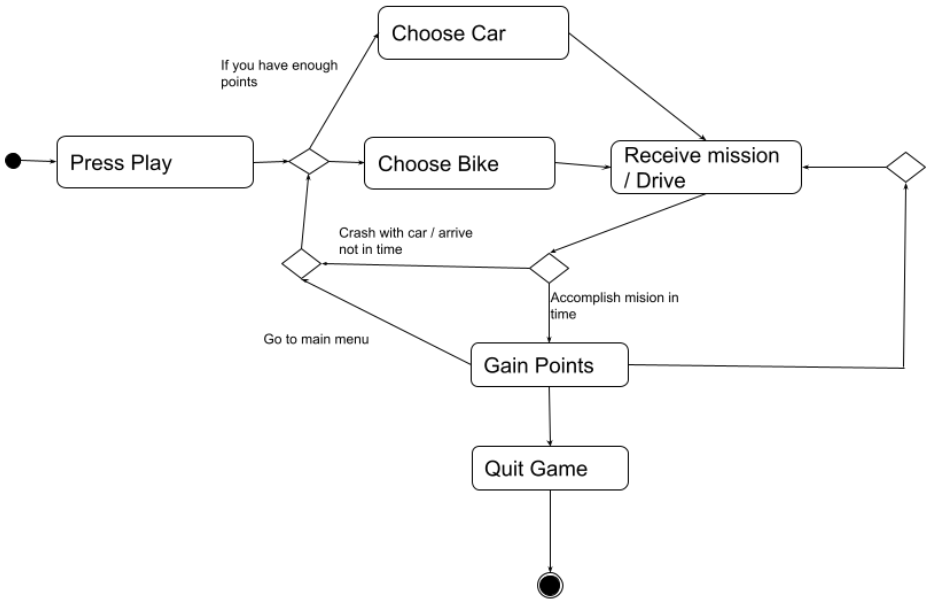


Figure 3.3: Activity diagram

Figure 3.4: Main Menu concept

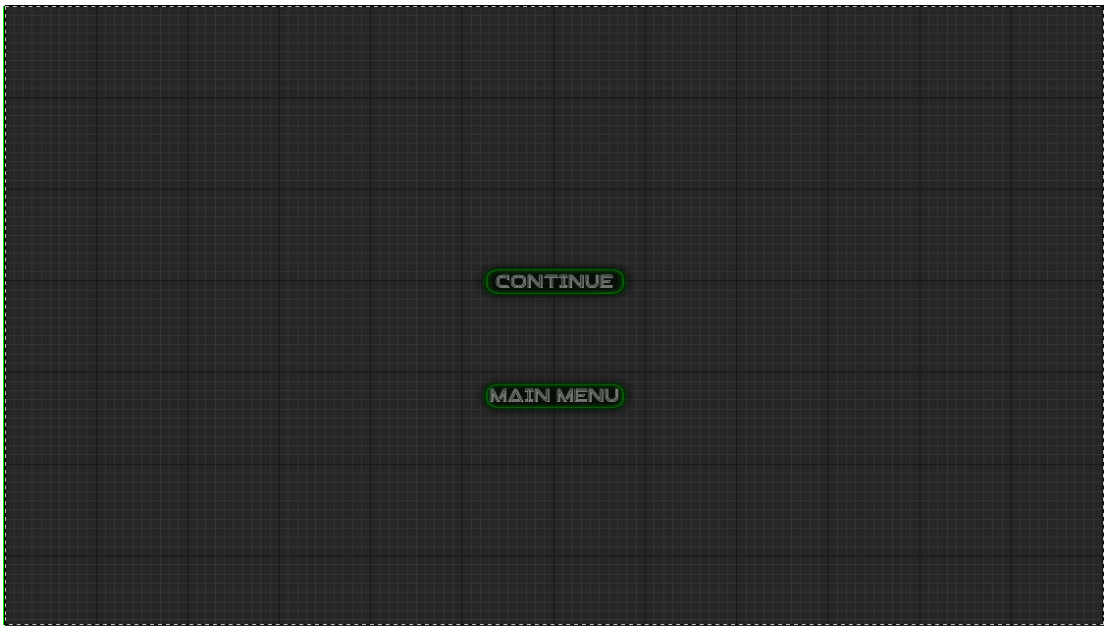


Figure 3.5: Pause Menu concept

Figure 3.6: HUD concept

CHAPTER 4

Work Development and Results

Contents

The developed work and the obtained results will be made explicit in this chapter. All possible deviations from the initial planning are detailed and justified in the results section (see section 4.2).

4.1 Work Development

The programming was done mostly on blueprint, this decision was made because the level of prototyping and testing new mechanics is faster. That's why for these project it is more suitable to be done this way. Although some parts needed to be done in c++, and because the interaction that unreal has between blueprint and c++ is really good this was the best approach.

The development consisted of 6 main parts: the Road System, Car AI, Player, Traffic Lights, Map building and Deliver System. Each part will integrate various of the tasks described in Chapter 3 (see chapter 3):

- Road System (see section 4.1.1): One-way roads, Non grid roads, More than one lane roads
- Car AI (see section 4.1.2): Fundamental behavior of vehicles, Acceleration and deceleration, More than one vehicle
- Player (see section 4.1.3): Cyclist, Car

- Traffic Lights (see section 4.1.4): Traffic lights
- Map building (see section 4.1.5): Map creation, Final touches
- Deliver System (see section 4.1.6): Automated destinations, Package delivery

4.1.1 Road system

Firstly the approach that needed to be defined is the road system, the two main options were having a **NavMesh** [10] or having a spline that defined the road. A **NavMesh** would provide a lot of flexibility but the number of possible bugs that could appear is big. The option of having a spline is less flexible and would require to be more careful with the road placement, but on the other side it would give more control on the possible routes that the car cloud follow.

Figure 4.1: Road intersection

The final decision was to have a spline to be the road that the car would follow (see Figure 4.2). But because there was the need to have bifurcated paths and splines could handle that situations the intersections were included (see Figure 4.1). Their function is to have references to the roads that he is in contact with, and when a car enters they'll provide that information to the car to make it able to go throw bifurcations in splines Car at intersections - YouTube. The roads needed to have defined an start, an end and a boolean that tells if they are a one way road. That information is read by the intersection and stored by the construction script so that it is not needed to compile extra code at run-time.

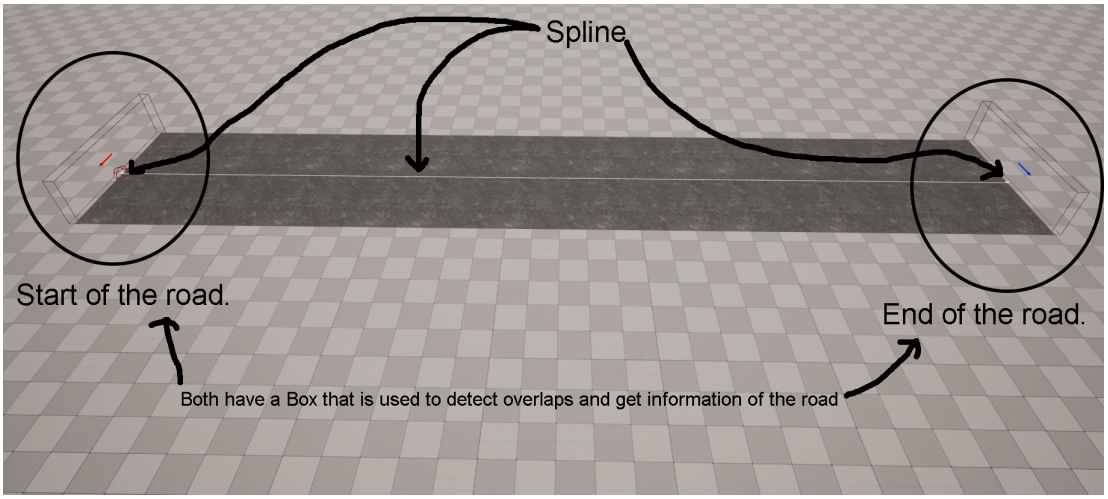


Figure 4.2: Road diagram

4.1.2 Car AI

The assets and basic configuration will be taken form the **City Sample Vehicles** from Epic Games [6] (See image 4.3).

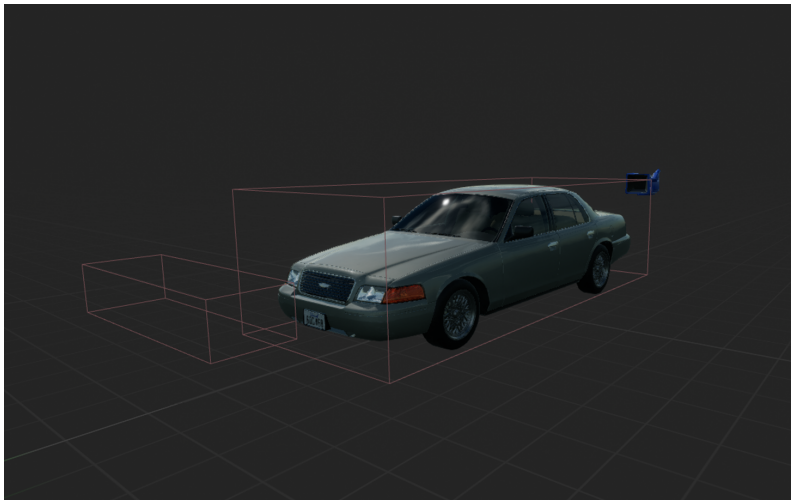


Figure 4.3: Car model

Moving to the car movement, it will be done using the **Chaos Vehicles plugin** from unreal [7]. The plugin enable us to easily control different aspects of vehicles. But in exchange you need to configure more things. The main thing that will be needed are a physics asset that will be important to define the size and form of the wheels, and the car collision with the environment. After that you need to specify different

parameters as shown on the image (See image 4.4), the most important ones are the wheels (which have their own configuration, see image 4.5), the torque, RPM and mass. After configuring all that the acceleration, brake and steer is managed by the Chaos Vehicles System. Then the work done is divided into: make the car follow the spline, the speed control, the target point (instant next destination) and predict the path (smooth transitions).

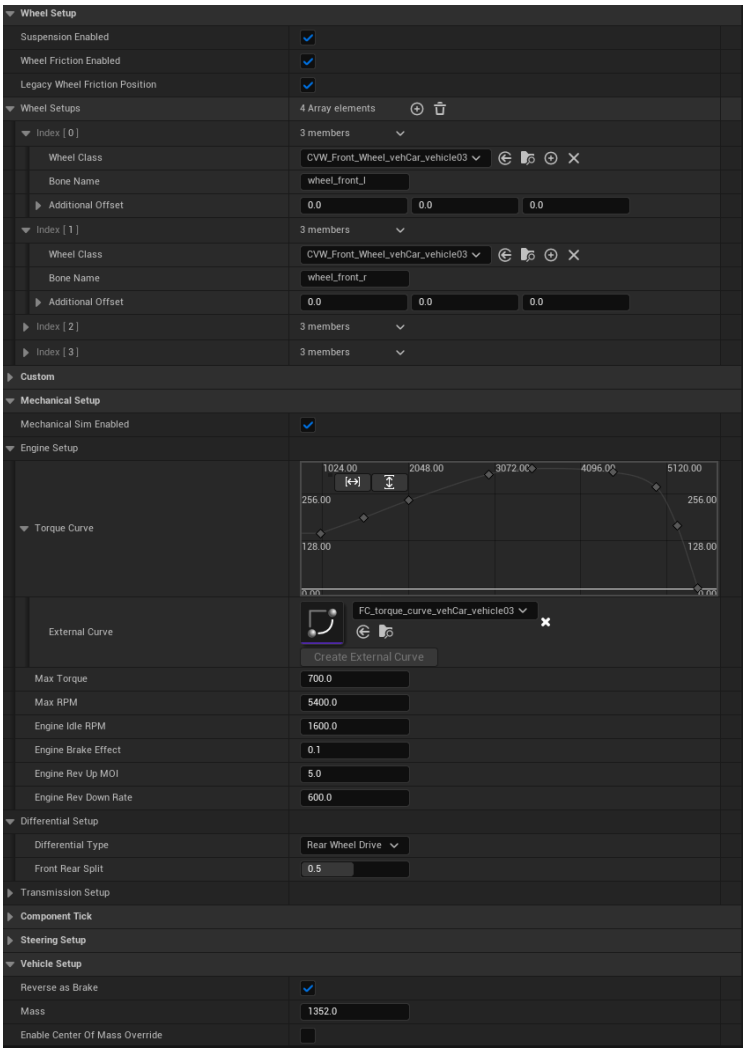


Figure 4.4: Chaos Vehicle System parameters

To make the car follow a spline is fairly simple, we mainly need the reference to the spline and a function that returns the coordinates of the spline at a given distance of the spline. With that we can set the coordinates as the destination. Then we need to make this target point adjust to the position of the car along the spline. When we are too close to the target position we'll want to look for the next position along the spline

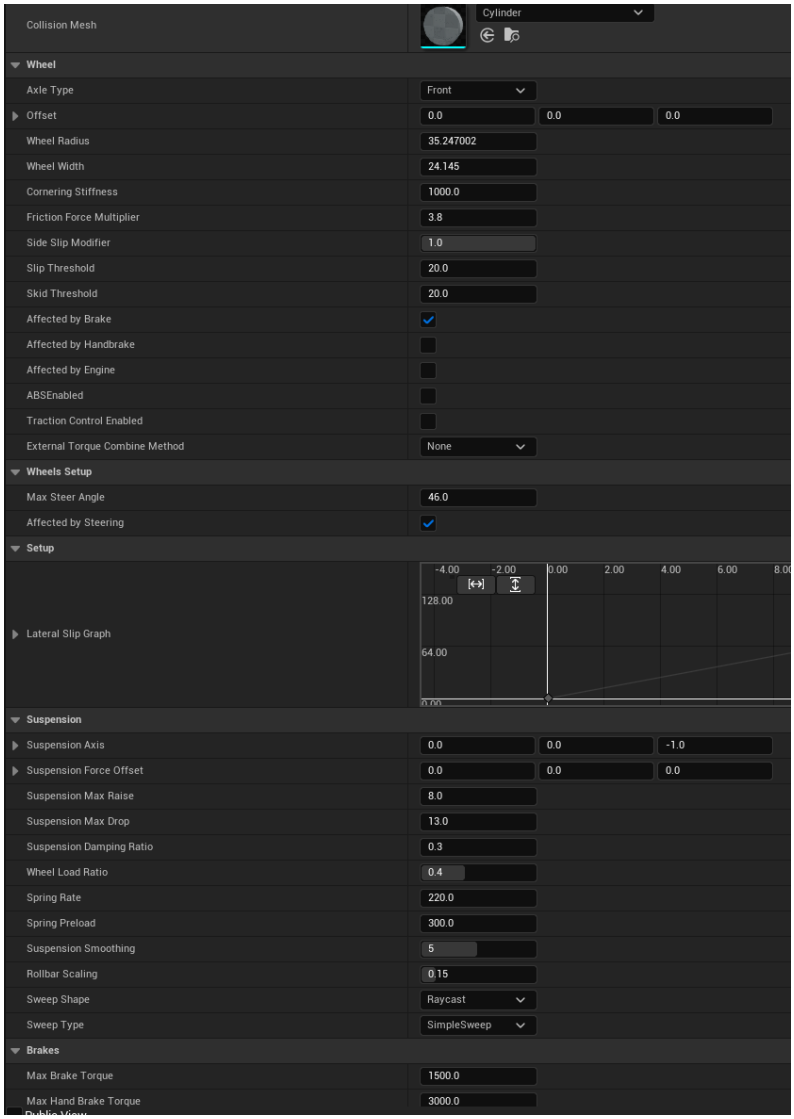


Figure 4.5: Chaos Vehicle Wheels parameters

Car following spline - YouTube. And because we are having two way roads we want the increment on the distance to be aware of the direction in which the car is moving to return the appropriate distance along the spline and the appropriate horizontal offset of the road for driving always at right. This is also relevant when crossing an intersection to take into account the direction of the road to know were the right is. The speed control is necessary for various reasons, we need it to brake on traffic lights, when going to collide, to set the maximum speed, and to slow the speed for curves.

4.1.3 Player

At the beginning, the idea was for the player's movement to be controlled by a C++ class that receives the player's input and includes all the necessary functions. Additionally, the animations were intended to be obtained from **Mixamo** [1] or other websites. However, after incorporating these animations and programming the C++ class, it was decided to change the plan mainly because the bicycle did not take into account the fact that when you accelerate and pedal, the force is applied to the rear wheel and the rotation comes from the front wheel. Moreover, when you stop accelerating, the wheels continue to rotate while the pedals remain still. The animations were not fully compatible with this type of behavior, leading to the conclusion that implementing the entire bicycle movement through code was not feasible as it would require too much additional work and would result in patching upon patches.

It was necessary to change the base player, and for this purpose, the decision was made to use the **Chaos Vehicle plugin** from Unreal Engine [7]. This vehicle has different requirements than the previously used cars (see section 4.1.2), and it was necessary to adapt all the necessary parameters to a two-wheeled vehicle. The physics asset of the bicycle were created to ensure compatibility, the wheels were configured, and particular attention was given to stabilization which is a completely different aspect compared to a four-wheeled vehicle. The bicycle requires a different center of mass since it needs to be able to return to the center in a stable manner when it leans to the sides, simulating how a person balances a bicycle when it tilts to the sides. Similarly, when the bicycle naturally tilts to one side, the wheel turns to balance the bicycle. Although we cannot physically turn the wheel in this case since it is controlled by the player, it is compensated by having a center of mass and a distribution in a way that keeps the bicycle always balanced.

To control the bicycle, the throttle, brake, and steer functions provided by the Chaos Vehicles plugin of Unreal Engine were used. To handle these controls, the **enhanced input system** of Unreal Engine [9]. was utilized, replacing the previous player input control system. Despite having a higher learning curve, the enhanced input system simplifies development for larger games and makes it more suitable for controllers, allowing separate management of the type of action performed by an input and the associated key inputs.

Once the bicycle is functional and moving, the next step is to create animations for this new bicycle with its unique controls. To do this, we began by separating the pedals from the entire bicycle mesh in Blender. This allows us to import them into Unreal Engine and place several sockets that will serve as reference points for the feet. These sockets will be used in the creation of procedural animations that will be generated based on the locations of specific points on the bicycle. To accomplish this, we will need a skeleton for the main character. The default skeleton provided by Unreal Engine is quite helpful, as it already has a pre-formed and well-structured bone hierarchy. Additionally, we will utilize the **Control Rig plugin** from Unreal Engine [8]. This plugin allows us to create handlers that can move specific bones in a natural and organic manner, similar to inverse kinematics. Setting up a Control Rig can be a complex task, especially for

humanoid forms. Therefore, we used the control rig project available on the Unreal Engine Marketplace, although it is specifically designed for Unreal Engine version 4.26. To use it in the version that is used for this project (Unreal Engine 5.1), it requires downloading the 4.26 version and migrating the relevant parts to the new version. After adapting the Control Rig code to Unreal Engine 5.1 (See image 4.6), we add setters to multiple points that can be modified and used for various purposes, such as hands, feet, torso, and head. These points will be used in the animation blueprint, which will control the position of the Control Rig. This animation will make use of two functions of the character. One of the functions is used to retrieve key points of the bicycle, such as the seat, handlebars, pedals, spine tilt, and head position, based on the current state of the bike. After obtaining this information, we set the hand positions in the Control Rig to the specified points. Due to the configuration of the Control Rig itself, it will naturally move towards these points.

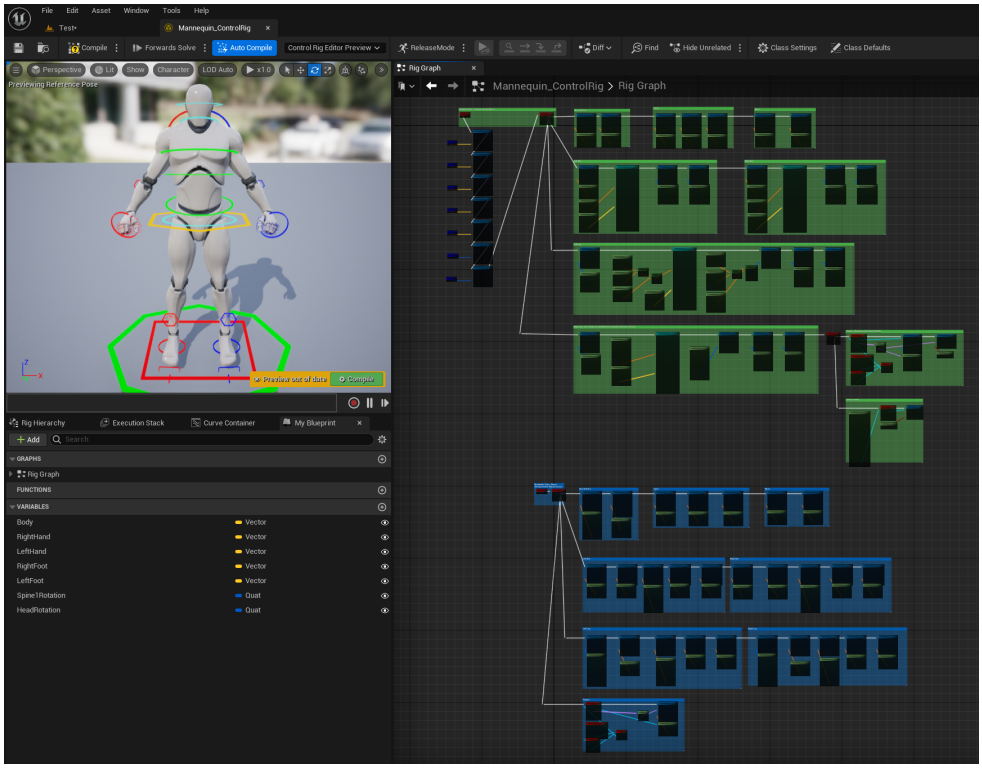


Figure 4.6: Control Rig Blueprint

We configure an idle position in which the player leans forward and supports the foot on the pedal. To achieve this, we use a player function that returns the velocity in km/h, and if it is close to zero, we obtain a different spine and head position from the function that returns the key points of the bike, to create a more natural appearance. By using a linear interpolation function to gradually transition between points based on the speed, the character can switch between idle (See image 4.7) and cycling smoothly

(See image 4.8). As the animation is procedural, the foot that rests on the pedal will stay in place, regardless of the pedal's current position.

Figure 4.7: Cyclist Idle

Figure 4.8: Cyclist Moving

With all these elements in place and the pedals and wheels configured to rotate at the appropriate speed, along with the bicycle's handlebars responding to player input, we now have a player with a functioning bicycle. The Chaos Vehicles system and procedural animations, utilizing the Control Rig and animation blueprint of Unreal Engine, bring the bicycle to life Character Movement - YouTube.

Finally, a character mesh was added to substitute the unreal mannequin base mesh as shown in image 4.9.

Figure 4.9: Comparison between default mesh and final mesh

4.1.4 Traffic lights

To ensure efficient traffic flow, the coordination of traffic lights with the timings of other lights in the intersection is crucial. Given the substantial number of intersections to configure, an automated approach was adopted. This approach involves the automatic spawning of traffic lights at the appropriate positions, rotations, and with adjusted delays between them. Consequently, whenever an intersection is added, the necessary traffic lights are spawned automatically at the required locations. This streamlined process simplifies intersection setup and facilitates the seamless integration of traffic light systems.

4.1.5 Map building

To create the map, a significant portion of the assets were sourced from the **City Megapack** [12]. The map's design was inspired by an image from Crosshead Studios, which was originally intended as a D&D City map (See image 4.10) [13]. Meticulous manual placement of roads, sidewalks, and buildings was undertaken, involving repetitive work and considerable time investment (See image 4.11 and image 4.12).

One notable advantage of having the same roads and same buildings but in different positions is that by the way it is configured, the material and textures sent to the GPU is less, therefore the VRAM consumption is minimized and prevents data congestion on the communication bus between the CPU and GPU. As a result, the system operates more efficiently, requiring fewer resources and ensuring smooth data flow between the central and graphical processing units. Despite these optimizations, the detailed nature of the assets still demands mid/high PC specifications to run the game effectively.

Figure 4.10: Crosshead Studios D&D Map

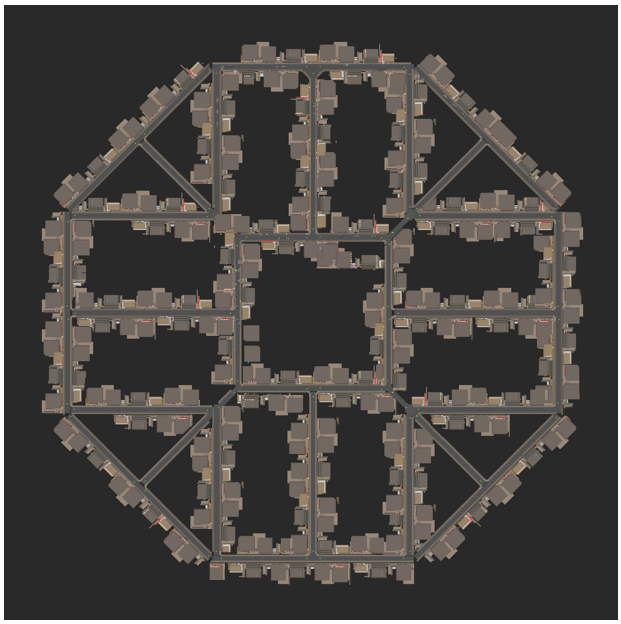


Figure 4.11: Delivery Drift map top view

Figure 4.12: Delivery Drift map side view

4.1.6 Deliver system

The deliver system is really simple and consists of mainly of choosing a random point and showing a beacon (See image 4.15) pointing to the sky to show the position and once arrived there the beacon disappears and a new random location is chosen. To make it more challenging there is a timer to deliver the package. To be able to see the beacon from every point a custom material was needed to highlight the beacon. The material domain needed to be of type post process so that with calculations taking into account the view size, the screen position and the size of the object, the material is able to highlight the beacon. In the player side, it was needed to Visualize the packages delivered properly. But not only in the HUD, but also physically the cyclist is wearing a Package that disappears just when you deliver the package and a new one is spawned after a few seconds. There is another version of the game where you are able to choose a car to drive though the city instead of the bike. This is unlocked just after you're able to deliver 5 packages in time. This will be chosen from the main menu. Also, the game over happens when you crash into a car. When that happens you loose the control of the player and after a few seconds the game restarts.

4.2 Results

Exclusions from initial ideas:

The work archived differs from the initial approach in the GDD in two main things, the not inclusion of pedestrians and the environment conditions like day/night cycle or weather. The main reason to not include them was due to time, but choosing them among other things to not include them is because they are mainly fancy elements and not functional mechanics that affect the game play in a significant way. Even though

Figure 4.13: Beacon on Map

they could become interesting mechanics the amount of work needed to achieve the point where they become mechanics and not just extra elements that add life or fancy visuals.

Traffic system:

Shifting our focus to the accomplishments, the primary goal of establishing a functional traffic system has been successfully attained. This can be observed in the game play video [1], where the vehicles demonstrate the capability to navigate seamlessly across all roadways.

Player:

The cyclist has exceeded initial expectations by incorporating animations from the renowned platform **Mixamo** [1] and employing a simplified movement system. The resulting implementation showcases notable improvements, as the animation system encompasses a broader spectrum of movements, rendering a heightened sense of realism. Moreover, the integration of the **Chaos Vehicle plugin** [7] elevates the cycling experience by simulating friction, mass, wheel dynamics, and other physical attributes. Furthermore, the objective of introducing an additional vehicle upon completing deliveries has been achieved. The intention behind this addition was to create a mode in which the game does not restart upon collisions, resulting in a different game play experience characterized by pure enjoyment, as demonstrated in this video showcasing the car Car showcase - YouTube - YouTube.

[1] *Delivery Drift - YouTube*, https://youtu.be/S7lLfCrk1bQ.

Figure 4.14: Initial View of the game

Traffic Lights:

The initial idea of incorporating additional elements such as yield signs and crosswalks to extend the traffic control system beyond just traffic lights was considered. However, since pedestrian safety measures were not taken into account, there was no valid reason to implement this idea. Overall, the city has successfully achieved a realistic and detailed appearance, offering an authentic urban experience.

Figure 4.15: Intersection view, traffic lights

Deliver System:
The delivery system has been successfully executed in line with initial expectations, presenting a functional and minimalist approach. Notably, the system effectively communicates the destination to the player without the need for textual information or any additional user interface elements. Furthermore, the player is visually informed of successful deliveries through the disappearance of the beacon and packages, followed by the appearance of a new beacon and package, indicating continuous progress.

Map Building:
Finally, the main menu has achieved a cyberpunk minimal style and it used two creative commons assets, the **Stacker Font** [2] and a neon **squircle rectangle** [23] from **freesvg** [20], and for the HUD it was used tree more creative commons images, the **Speedometer** [4], a **timer icon** [21] and the Package points icon [17]. So the main menu with its minimalist approach achieved this final result (See image 4.16 as well as the HUD (See image 4.17.

All together, these elements combine to create a captivating casual/simulation video game as we can see on the video Game play Video: Delivery Drift - YouTube

Figure 4.16: Final Main Menu

Figure 4.17: HUD in game

Chapter 5

Conclusions and Future Work

Contents

5.1 Conclusions

During the development of this game, I had both professional and personal experiences that greatly impacted my growth and learning. One notable aspect was the opportunity to work extensively with Unreal Engine, which I discovered it was more powerful game development tool than I thought. Before this project, my knowledge of Unreal Engine was minimal, but through the development process, I gained a deep understanding of its functionalities and capabilities.

Professionally, this project allowed me to enhance my skills in game design, level creation, and AI programming. Building a bustling city environment with realistic buildings and vehicles was a challenging yet rewarding experience. It required careful attention to detail and some optimization to create an immersive and believable world.

Personally, this project allowed me to explore my creativity and problem-solving skills. Designing the city of making all the different systems work all together without problems offered the opportunity to challenge the knowledge I have.

In terms of the relation between this work and my degree, the development of this game aligns closely with the concepts and skills I acquired during my coursework. The game development process encompasses various aspects, including software engineering, computer graphics, animation, and artificial intelligence, which are core components of

my degree. Through this project, I was able to apply the theoretical knowledge I learned in my degree program to a practical, hands-on project.

Overall, the development of this game has been an enriching experience. I have not only expanded my understanding of Unreal Engine and game development but also honed my problem-solving abilities and creative thinking. It has been a valuable opportunity to bridge the gap between theoretical knowledge and practical application, and I am confident that the skills and experiences gained from this project will benefit me in future endeavors within the game development industry.

5.2 Future work

Looking ahead to the future development of this game, I am excited to incorporate the new tool introduced in Unreal Engine 5.2 called the **Procedural Content Generation Framework** [11]. This framework will be fundamental in creating a much larger and more expansive city environment for the game.

Traditionally, building a city by hand is a time-consuming and repetitive task. It involves manually designing and placing each building, road, and pedestrian, which can be a daunting process. However, with the Procedural Content Generation Framework, we can automate and streamline this process by using algorithms and rules to generate the city procedurally.

This framework allows us to define various parameters and rules for generating the city. We can specify the density and distribution of buildings, the layout of the road network, and the placement of pedestrians and vehicles. By leveraging the power of procedural generation, we can create a city that feels organic and realistic, while also saving a significant amount of time and effort in the development process.

Bibliography

[1] Adobe. Mixamo. https://www.mixamo.com/. Accessed: 2023-06-12.

[2] Almarkhatype. Stacker font. https://www.fontspace.com/stacker-font-f82510. Accessed: 2023-06-12.

[3] RHT Apps. Delivery truck simulator. https://play.google.com/store/apps/details?id=com.RedHorn.TrukS Accessed: 2023-06-12.

[4] berkut123. Speedometer icon or sign with arrow vector image. https://www.vectorstock.com/royalty-free-vector/speedometer-icon-or-sign-with-arrow-vector-23887416. Accessed: 2023-06-12.

[5] Sergio Cejas. Este sería el asombroso apartado visual de gta iv si fuese remasterizado en 4k y con ray-tracing. https://www.vidaextra.com/accion/este-seria-asombroso-apartado-visual-gta-iv-fuese-remasterizado-4k-ray-tracing. Accessed: 2023-06-12.

[6] Epic Games Epic Content. City sample vehicles. https://www.unrealengine.com/marketplace/en-US/product/city-sample-vehicles. Accessed: 2023-06-12.

[7] Unreal Engine. Chaos vehicles. https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Physics/ChaosPhysics/ChaosVehicles/. Accessed: 2023-06-12.

[8] Unreal Engine. Control rig. https://docs.unrealengine.com/5.0/en-US/control-rig-in-unreal-engine/. Accessed: 2023-06-12.

[9] Unreal Engine. Enhanced input. https://docs.unrealengine.com/5.0/en-US/enhanced-input-in-unreal-engine/. Accessed: 2023-06-12.

[10] Unreal Engine. Nav mesh. https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/NavMesh/. Accessed: 2023-06-12.

[11] Unreal Engine. Procedural content generation framework. https://docs.unrealengine.com/5.2/en-US/procedural-content-generation–framework-in-unreal-engine/. Accessed: 2023-06-12.

[12] Kyrylo Sibiriakov Environments. City environment megapack vol 02. https://www.unrealengine.com/marketplace/en-US/product/city-environment-megapack-vol. Accessed: 2023-06-12.

[13] Play With Games. Crosshead studios. https://crossheadstudios.com/runeport/. Accessed: 2023-06-12.

[14] Play With Games. Pizza delivery: Simulador de c. https://play.google.com/store/apps/details?id=com.playwithgames.pizza.delivery.parkinghl=es$_4$19 $US. Accessed : 2023 - 06 - 12$.

[15] RockStar Games. Grand theft auto iv. https://www.rockstargames.com/es/games/IV. Accessed: 2023-06-12.

[16] RockStar Games. Grand theft auto v. https://www.rockstargames.com/es/gta-v. Accessed: 2023-06-12.

[17] HansAchterbahn. Distance. https://freesvg.org/distance. Accessed: 2023-06-12.

[18] Karl Kontus. Video game insights report: First half of 2022 on steam. https://www.gamedeveloper.com/blogs/video-game-insights-report-first-half-of-2022-on-steam. Accessed: 2023-06-12.

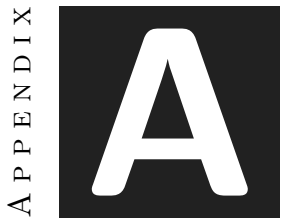
[19] Tom Looman. Professional game development in c++ and unreal engine. https://courses.tomlooman.com/p/unrealengine-cpp. Accessed: 2023-06-12.

[20] OpenClipart. Neon numerals-backgrounds. https://freesvg.org/rwwgub-neon-numerals-backgrounds-5. Accessed: 2023-06-12.

[21] OpenClipart-Vectors. Icon stopwatch clock time black. https://pixabay.com/vectors/icon-stopwatch-clock-time-black-157350/. Accessed: 2023-06-12.

[22] RocketArts. Stylized character kit: Casual 01. https://www.unrealengine.com/marketplace/en-US/product/stylized-male-character-kit-casual. Accessed: 2023-06-12.

[23] Wikipedia. Squircle. https://es.wikipedia.org/wiki/Squircle. Accessed: 2023-06-12.



Source code

All the code can be accessed trough GitHub: TFG-DeliveryDrift - GitHub