Projeto:

Lane Change Test

Faculdade de Ciências e Exatas e da Engenharia

Data: 9-6-2023

Docente: Filipe Quintal

Discente: Tiago Augusto Noite Meneses
Número de aluno: 2076520

Conteúdo

1.Introdução

O projeto foi baseado no Lane Change Test descrito neste artigos [1], [2], [3] e [6], este é uma simulação de condução que permite verificar os níveis de atenção dos condutores, e o tempo de reação destes em ambientes controlados com tarefas que têm de ser feitas durante a condução. Isto permite tirar conclusões, na forma como as interfaces têm de ser desenhadas e o que pode ser melhorado nestas. O objetivo da criação do programa LCT é para evitar pôr em risco a vida das pessoas que fossem conduzir os veículos pois como estes tinham de realizar pequenas tarefas poderia acabar havendo acidentes, magoando alguém desnecessariamente.

O problema do Lane Change Test é que esta simulação está muito desatualizada para usar sendo difícil de configurar, instalar e fornece poucas configurações durante o uso deste, para além disso, mesmo não sendo o foco do problema, esta aplicação não é apelativa visualmente.

O objetivo do projeto criado é melhorar os aspetos negativos do programa de simulação do Lane Change Test anterior e para tal foi criado uma versão nova em Unity, uma ferramenta portável e acessível a qualquer sistema operativo, que facilita na instalação, setup e leitura e analise de dados.

2.Solução

A solução apresentada neste projeto, tem como objetivo melhorar os aspetos negativos do programa utilizado.

Os principais objetivos são melhorar a portabilidade do sistema, simplificar as configurações de instalação e o setup, oferecer opções adicionais aos analisadores antes de criar o teste e facilitar a coleta de informações.

Este programa desenvolvido em Unity torna a instalação e utilização mais fácil para qualquer pessoa, mesmo que possua um computador antigo e permite que o sistema seja integrado em outros ambientes. O Unity é conhecido pela sua portabilidade e facilidade de uso. Quanto às opções disponíveis para os analistas durante o teste, estas estão relacionadas à capacidade de ajustar a velocidade do carro, a distância da pista e até mesmo a distância entre sinais, assim oferecendo uma maior flexibilidade para realizar testes personalizados caso necessário.

Por fim, os dados são armazenados no final de cada volta em uma folha de Excel, o que permite uma visualização mais fácil dos resultados através de gráficos assim permitindo uma visualização melhor dos dados coletados durante as repetições

Este programa tenta recriar o melhor possível o Lane Change Test de forma que obedeça a todos os parâmetros que foram referidos em alguns artigos que irei mencionar, [1], [2], [3] e [6].

3.Revisão de Literatura

Nesta secção irei explicar o porque que a ferramenta escolhida foi o Unity enquanto descrevo esta e outras ferramentas como o CryEngine, o Godot e uma das maiores competidoras da Unity o Unreal engine.

3.1 Unity

O Unity é um dos motores de desenvolvimento de videojogos mais conhecidos. O termo motor de desenvolvimento ou motor de jogo, refere-se a um tipo específico de software que possui uma série de rotinas de programação que permitem a projeção, criação e a operação de um ambiente interativo, ou seja, de um videojogo, experiência digital ou filme/animação. Entre as funcionalidades típicas de um motor de jogo, estão as seguintes:

- Motor gráfico para renderização de gráficos 2D e 3D;
- Motor de física que simula as interações entre objetos;
- Sistema de iluminação;
- Animações;
- Sons;
- Programação de inteligência artificial;
- Programação por scripting;
- Simulação de partículas;

Neste projeto foi usado o Unity que é um engine de criação de jogos bastante popular, após seus criadores terem libertado uma edição gratuita (com algumas limitações) em 2009.

Esta permite criar jogos para smartphones, tablets, consolas, browsers e computadores e funciona tanto no Windows quanto no MacOS.

O Unity dá aos utilizadores a hipótese de criar jogos em 2D e 3D suportando as várias APIs como: Direct3D no Windows e na Xbox 360, OpenGL no MacOS, Linux e OpenGL ES no Android e iOS, WebGL na Internet.

Este usa o MonoDevelop que é um IDE integrado para a criação e debug dos scripts, este pode usar C#, ou Boo (que tem uma sintaxe inspirada no Python), em 2015, a Unity retirou o suporte a linguagem Boo e em 2017, a Unity declarou que iria fechar o suporte da linguagem UnityScript, que estava ligado ao Unity desde sua criação.

Nos jogos 2D, esta autoriza a importação de sprites e um renderizador de mundo 2D, já para jogos 3D, o Unity consente a especificação de compressão de textura, mipmaps e configurações de resolução para cada plataforma suportada pelo mecanismo de jogo, e fornece suporte para os mapeamentos de relevo, de reflexão, de paralaxe, oclusão de ambiente de espaço de tela, sombras usando mapas de sombras, efeitos de pós-processamento de renderização para textura e tela inteira.

O Unity também oferece outros serviços para os desenvolvedores como o Unity Ads e outros, e suporta a criação de vértices, fragmentos personalizados, tesselação, shaders de computação.

As características principais que o Unity tem são:

- Suporte ao uso de shaders;
- Programação em C# (principal);

- Suporte ao PhysX, incluindo detetor de colisão, soft body e ragdoll;
- Compatibilidade com os navegadores (via o plugin Unity Web Player);
- Compatibilidade com Blender, 3ds Max, Maya, Cinema 4D, Cheetah 3D, Softimage, modo, ZBrush, Lightwave, Photoshop, Fireworks e Substance.

De facto, existem diversos motores de criação de jogos no mercado atualmente como Unity, Godot Engine, CryEngine e outros sendo que estes poderiam ser usados para criar projetos semelhantes, mas sem dúvidas os que mais se destacam entre os diversos engines são a Unreal e a Unity pois ambas disponibilizarem uma grande diversidade de recursos e ferramentas para seus utilizadores, facilitando a criação de jogos, estas são as mais versáteis, servindo tanto para criar jogos 2D quanto 3D.

As empresas maiores preferem usar a Unreal para criar projetos como melhores gráficos, enquanto os desenvolvedores Indie preferem usar a Unity pelas suas capacidades para criar jogos 2d e 3d e versatilidade

3.2 Unreal

A Unreal Engine está por trás de grandes jogos como Mortal Kombat X, Batman, Borderlands, Tekken 7 e BioShock 2. Esta foi desenvolvida pela EPIC Games. Esta usa C++, o que possibilita uma boa portabilidade para diversas consolas e sistemas operativos.

Este engine como dito é bastante bom para criar jogos de alta qualidade gráfica, mas é preciso saber quando escolher este pois dependendo do tipo de projeto e das necessidades deste, temos de saber qual a melhor ferramenta a escolher. Algumas características desta são:

- Suporte de scripts em python para automatizar os fluxos de trabalho;
- A Unreal inclui o Unreal Editor, um ambiente de desenvolvimento integrado disponível em Linux, MacOS e Windows para criação de conteúdo e desenvolvimento;
- Permite criar terrenos inteiros com o sistema Landscape;
- O Unreal oferece ferramentas como geração automática de nível de detalhe, a ferramenta Proxy Geometry para otimizar o uso dos recursos disponíveis;
- Permite ter acesso ao código-fonte C++ completo, permitindo personalizar, estender e depurar toda a Unreal Engine;
- Fornece protótipos físicos para a criação de um jogo;

As desvantagens de usar o Unreal Engine é que ao produzir jogos de grande escala pode acabar sendo atingido o limite e fazer um jogo demasiado aberto ou grande [4].

Em suma o Unreal engine é uma boa ferramenta para criar jogos de alta qualidade fornecendo protótipos para ajudar os criadores e muitas outras ferramentas, mas no caso deste projeto a simulação a ser feita não tinha de ser em grande escala nem de ter bons gráficos sendo mais importante a funcionalidade e simplicidade do projeto assim sendo melhor optar pelo Unity para tal.

3.3 CryEngine

A CryEngine [5] foi criada para fazer diferentes tipos de jogos e desenvolvido pela Crytech usando as linguagens de programação C++, C# e Lua. Foi lançado pela em 2002 e sua versão estável foi lançada em 2020. Pode ser executado em diferentes plataformas como Windows, MacOS, Linux etc. É usado por muitas organizações para fazer jogos

comerciais, é popular devido à sua simplicidade de usar interface e a sua qualidade visual. Algumas das características que este motor tem são:

- Esta tem uma renderização e iluminação de alta qualidade resultando em gráficos de alta qualidade
- Fornece uma variedade de ferramentas para manipular objetos e ambientes
- Possui um motor de física de objetos preciso e realista.
- Permite criar ambientes vastos dando suporte a estes nos detalhes
- Esta engine é compatível com várias plataformas como computadores, dispositivos móveis e consolas

A partir destas características pode-se deduzir que as suas vantagens principais são:

- o facto de permitir uma renderização de alta qualidade assim tendo gráficos bastante bons
- permite o uso de ferramentas bastante úteis para a criação de jogos assim criando jogos ricos em detalhes
- esta engine é especializada em criar mundos abertos.

As desvantagens deste engine são:

- Necessita de muito tempo e esforço para aprender para os iniciantes tendo uma curva de aprendizagem grande;
- Requer um hardware poderoso para suportar os gráficos;
- Principalmente este tem uma documentação limitada tornando moroso o processo de aprendizagem;

Com base nestas desvantagens podemos concluir que apesar de ser uma boa ferramenta para uso esta não seria a mais indicada para este projeto pelo tempo que iria levar a aprender e para além disso pelo facto de ser necessário um computador poderoso para aguentar a qualidade dos gráficos.

3.4 Godot

A Godot foi desenvolvida para que os seus jogos fossem de código aberto e é gratuita. Esta é fácil de usar, fornece recursos avançados e tem uma comunidade ativa, algumas das suas características e vantagens são:

- Esta é suportada por várias plataformas;
- Permite criar jogos sem escrever muito código;
- Esta usa uma linguagem semelhante a python chamada GDScript sendo esta também fácil de usar;
- Esta dá suporte a recursos avançados como físicas 2d e 3d, shaders personalizados e efeitos visuais;
- Esta tem uma curva de aprendizagem suave facilitando a criação de jogos
- Este é uma ferramenta bastante leve pesando só 60mb sendo só o pacote de exportação para Mac da Unity pesa 2gb
- Permite fazer alterações nos scripts enquanto o jogo está a ser testado ao contrário do Unity
- o editor de código está embutido na engine

quanto às suas desvantagens temos:

- Este tem recurso 3d limitados, ou seja, não tem tanto suporte para a criação de jogos 3d como para os jogos 2d;

- Este também não tem muitas bibliotecas que podem ser usadas comparado com outras engine, sendo necessário um esforço maior para criar certos elementos;
- É mais difícil de escrever código bom na linguagem usada

Podemos concluir que esta poderia ser uma boa ferramenta a ser usada, mas tendo algumas desvantagens podendo estas interferir no projeto final.

Em suma a melhor ferramenta a usar para este projeto seria ou o Unity ou o Godot pois ambas são simples para os iniciantes usarem, mas como a Godot pode não ter os modelos necessários para criar o veículo então a melhor opção foi o Unity pois esta tem imenso suporte para a criação de jogos e dúvidas que possa haver e uma grande comunidade para ajudar.

4.Implementação

Como referido anteriormente o que levou a criação deste programa foi o facto de o antigo ter alguns problemas a nível de compatibilidade e de configurações, a solução foi criar um novo que conseguisse resolver estes problemas e melhorar em outros aspetos, mas mantendo os standards usados nos testes do programa mais antigo, para tal foi necessário criar alguns requisitos básicos para criação deste programa, também explicado o código criado.

4.1 Requisitos

4.1.1 requisitos funcionais

- Deve ser usado o Unity para a criação do projeto;
- O sistema deve seguir o padrão de 60 metros para visualização dos sinais;
- O sistema deve seguir o padrão da pista ter 3 mil metros de comprimento;
- O sistema deve seguir o padrão de o carro andar a uma velocidade de 60 km\h;
- O sistema deve seguir o padrão da distância entre os sinais ser de 150 metros;
- O sistema deve permitir ao cliente inserir quantas voltas irá repetir antes de começar os testes;
- O sistema deve pedir ao cliente qual seu nome ou ID para começar o jogo;
- O sistema deve guardar os dados numa folha Excel;
- Os dados guardados no Excel devem permitir criar um gráfico no mesmo para entendimento
- Os dados a serem guardados devem ser as coordenadas de X, Z do veículo a ser conduzido pelo utilizador.
- O sistema deve guardar os dados das posições onde o veículo do condutor devia estar, ou seja, o percurso onde este deve estar em cada momento
- O sistema deve gerar as pistas de forma aleatória consoante a posição do carro
- O sistema deve seguir o padrão de que em cada pista só deve haver três vezes cada tipo de curva.

4.1.2 requisitos não funcionais

- O sistema deve avisar ao condutor uma mensagem quando este estiver a sair fora da pista
- O sistema fornece a capacidade de o investigador escolher qual a velocidade máxima do carro;
- O sistema fornece a capacidade de o investigador escolher o número de repetições a ser feitas na simulação;
- O sistema deve fornecer a capacidade de o investigador escolher a distância entre os sinais e a distância a que conseguimos visualizar estes;
- O sistema deve fornecer a capacidade de o investigador definir qual a distancia da pista;
- O sistema permite que quem esteja conduzindo o carro possa ter um pequeno descanso entre testes caso seja necessário.

4.2 Ferramentas

As ferramentas usadas para a criação do programa que simula o Lane Change Test forma o Unity como motor e base para a criação do projeto como já foi referido, o Visual Studio Code para criação do código necessário, o Paint para desenho de sprites necessários como sinais e a estrada e a Unity Assets Store para procurar e descarregar o modelo do carro.

4.3 Componentes de código

Nesta parte irei explicar o código criado, o que este faz sendo que irei primeiro proceder a nomeação e rápida explicação de cada componente e depois a uma explicação mais aprofundada.

Foram criados 15 componentes sendo eles:

- CarMovement: trata da parte de conseguir mover, virar e travar o veículo.
- finish_line: este trata de mover tanto a placa que avisa que a pista acabou como outros objetos que estejam ligados a esta, para o fim da pista.
- strat_line: este tem como objetivo mover a placa que avisa ao condutor que o teste começa.
- Form_Menu: aqui o player tem de inserir o Id de jogo e o número de tentativas que ira realizar sendo que se não puser o Id não consegue começar a simulação.
- Scene_changing: Esta contem as funções para escrever no ficheiro Excel e mudar de cena quando o condutor clicar no botão “r”.
- tranform: transformar o tamanho do cubo para o pedido nas opções ou para o standard que tem no LCT que é de 3 mil metros
- reset_level: este serve para quando um player chega ao fim da pista ao clicar no “r” o player é mandado para outro ecrã e para de guardar os valores para poderem ser escritos.
- Main_Menu: O Main_Menu é onde o player pode escolher ir para o menu de opções ou prosseguir para a área de teste
- Options_menu: este serve para fornecer a capacidade de personalizar de certa forma o teste sendo que podem alterar a velocidade do carro a distância entre sinais e outras partes do teste caso desejem, mas se nada for alterado tudo irá se manter dentro do standard do LCT
- Show_Warning: mostra a mensagem de aviso para voltar para a pista caso o player decida sair da pista por alguma razão
- signal_position: cria e posiciona os sinais com base no comprimento da pista e na distância entre cada sinal sendo que pode ser mudado nas opções estas duas características.

- spawn_images: este posiciona os sinais para dizer ao condutor para onde este deve se dirigir com base na sua posição nas faixas rodoviárias.
- Store_values: busca e armazena os valores das posições do carro e onde o carro deveria estar, de x em x milissegundos

Agora irei explicar mais detalhadamente cada componente do código criado:

4.3.1 Main_Menu

Este script como o nome indica está ligado ao Main Menu, este usa o "UnityEngine.SceneManagement" para fazer a ligação entre cenas pois este tem o método PlayGame() que ao ser chamado passa para a próxima cena.

```
public void PlayGame ()
{
    SceneManager. LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

depois caso seja para sair da aplicação então o Main Menu fornece outro método que tem essa função sendo ele o QuitGame().

```
public void QuitGame ()
{
    Debug.Log("QUIT!");
    Application.Quit ();
}
```

4.3.2 Options_menu

O options menu encontra se na mesma cena que o Main Menu e este serve para redefinir os standards do teste caso seja necessário, ou seja, este script é o que controla o quão rápido o carro vai ir, a distância entre os sinais, o quão distante o condutor consegue ver e quão comprida é a pista sendo que todas as variáveis que este tem são partilhadas com scripts em outros lugares.

Esta busca o valor posto num dos campos de input e passa o valor lá posto para a variável partilhada entre os outros scripts, por exemplo:

```
public static int Car_speed = 60;

public static int SignaDistance =150;

public static int RoadLenght = 1000;

public static float ViewField= 60;

public void Car_velocity(string Car_velocity){
    try{
        Car_speed = int.Parse(Car_velocity);
        if(Car_speed > 200){
            Car_speed=150;
        }
        if(Car_speed < 10){
            Car_speed=10;
        }

    }
    catch(Exception e)
    {
        print("Error "+e);
    }
}
```

Como se pode notar esta função recebe um parâmetro e este depois é verificado a ver se o valor não é muito pequeno ou muito grande para não ocorrer problemas com o teste e então o valor é passado para a variável global, também é verificado por erros caso o valor escrito no campo de input não possa ser convertido para inteiro.

4.3.3 Form_Menu

Este tem duas variáveis globais que também são partilhadas com outros scripts sendo elas o Person_ID que é o identificador que todas as pessoas têm de pôr para começarem a fazer a simulação e o rep que é o número de repetições que as pessoas vão ter de fazer durante a simulação sendo que esta também usa o “UnityEngine.SceneManagement” junto com “UnityEngine.UI” para mudar de cena e facilitar o desenvolvimento de interfaces.
Este tem funções do gênero do Options Menu, em que recebe uma variável e esta é passada para a variável global.

```
public void String_reader_1(string ID){
    Person_ID = ID;

}
```

Caso o campo do Id não tenha sido preenchido então não passa para a simulação se não entra nesta isto na função Play game.

```
public void PlayGame()
{
    if( Person_ID !=null){

        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
```

O número de repetições é feito no mesmo gênero em que recebe um argumento e passa esse valor para a variável rep caso não seja muito grande, caso não seja posto nada então o teste é repetido duas vezes, caso o valor seja menor que um então este é reposto a um.

Por fim, a última função que tem aqui é o QuitGame que volta uma cena atrás, ou seja, ao menu principal caso seja necessário.

```
public void QuitGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
}
```

4.3.4 CarMovement

As variáveis que este contem são os WheelCollider que tem um para cada roda, tem um raio de 0.8 unidades no Unity e estes servem para suportar o corpo e peso do carro e fazer o carro andar e não passar do chão, o transform que tem um para cada pneu do carro para fazer com que estes rodem consoante a direção que o carro está a andar, a acceleration que diz qual a velocidade de aceleração do carro sendo esta definida a 500 unidades, breakingForce que define quão rápido o carro consegue parar depois de estar a andar sendo que esta definida com 300 unidade, de seguida temos o currentAcceleration e currentBreakForce que define a aceleração a que o carro está a ir no momento e a força a que o carro esta a travar no momento, temos position_z e position_x que tratam de ir recolher a posição no eixo do x e z do carro para estes serem guardados mais tarde, o maxVelocity define a velocidade máxima a qual o condutor consegue ir, a CurrentSpeed diz quão rápido o jogador esta a ir durante a pista sendo que quando chega a uma velocidade igual ou superior a maxVelocity para de acelerar, temos uma variável com o nome de Entrou que verifica as variáveis que definem onde o carro deve se posicionar no inicio da pista e depois é definida como falsa, por fim temos o LowestSteerSpeed, lowSpeedSteerAngel e heightSpeedSteeringel que definem quanto o carro consegue virar a baixas e altas velocidades.

```
[SerializeField] WheelCollider front_right;
[SerializeField] WheelCollider front_left;
[SerializeField] WheelCollider back_right;
[SerializeField] WheelCollider back_left;

[SerializeField] Transform front_right_tranform;
[SerializeField] Transform front_left_tranform;
[SerializeField] Transform back_right_tranform;
[SerializeField] Transform back_left_tranform;

public float acceleration = 500f;
public float breakingForce = 300f;

private float currentAcceleration= 0f;
private float currentBreakForce = 0f;
public static float position_z=0;
public static float position_x=0;

public int maxVelocity;
public float CurrentSpeed=0;

private float LowestSteerSpeed =47;
private float lowSpeedSteerAngel = 37;
private float heightSpeedSteeringel = 1;
public bool entrou=true;
```

Os métodos chamados depois das variáveis globais são o 'Start, que é um método criado quando há um novo script a ser chamado e este é chamado quando o script é ativado e antes de qualquer método Update ou FixedUpdate, neste o único objetivo é ir buscar a velocidade máxima definida no menu de opções e passar esse valor para maxVelocity.

```
private void Start() {
    maxVelocity = Options_menu.Car_speed;

}
```

O FixedUpdate que chama a função controlar e trata da condução do veículo, sendo que dentro desta é calculada a velocidade atual do carro com base nos inputs do player junto com a direção para onde o carro deve virar e o grau de viragem, ou seja, quanto o carro consegue virar com base na velocidade a que está a ir, por fim a posição do carro no eixo do X e Z é passado para as variáveis position_z e position_x.

```
private void FixedUpdate() {

    controlar();
}
```

```
private void controlar() {
    //limite de velocidade
    CurrentSpeed = 2 * Mathf.PI * back_right.radius * back_right.rpm * 60 / 1000;
    CurrentSpeed= Mathf.Round(CurrentSpeed);


     // Get forward/reverse acceleration from the vertical axis (W and S keys)
     if(CurrentSpeed > maxVelocity){
        currentAcceleration= 0;
     }else{
        currentAcceleration= acceleration * Input.GetAxis("Vertical");

     }


    var speedFactor= GetComponent<Rigidbody>().velocity.magnitude/LowestSteerSpeed;
    var currentSteerAngle =Mathf.Lerp(lowSpeedSteerAngel, heightSpeedSteeringel,speedFactor );
    currentSteerAngle *=Input.GetAxis("Horizontal");

    if(Input.GetKey(KeyCode.Space)){
        currentBreakForce = breakingForce;
    }
    else{
        currentBreakForce = 0f;
    }

    front_right.motorTorque = currentAcceleration;
    front_left.motorTorque = currentAcceleration;

    front_right.brakeTorque = currentBreakForce;
    front_left.brakeTorque = currentBreakForce;
    back_right.brakeTorque = currentBreakForce;
    back_left.brakeTorque = currentBreakForce;

    //virar o carro

    front_left.steerAngle = currentSteerAngle;
    front_right.steerAngle = currentSteerAngle;


    position_z = transform.position.z;
    position_x=transform.position.x;

}
```

O último método chamado neste script é o Update() que verifica a variável Entrou e se esta for falsa não faz nada senão este busca a posição e comprimento da estrada e calcula a posição do carro com base nesses valores, depois põe a variável que verificou a falso e logo depois disso cada vez que é chamado usa as variáveis do tipo Transform, para fazer a textura das rodas do carro rodarem com base na velocidade do carro.

```
private void Update() {
    if(entrou){
        float x = tranform.lenght;
        float start_position =  GameObject.FindGameObjectWithTag("road").transform.position.x;
        start_position -= x;

        transform.position = new Vector3(start_position + 10, 1.5f , 500f);
        entrou =false;
    }

    front_right_tranform.Rotate( front_right.rpm/60 * 360 * Time.deltaTime, 0 ,0);
    front_left_tranform.Rotate( front_left.rpm/60 * 360 * Time.deltaTime, 0 ,0);
    back_right_tranform.Rotate( back_right.rpm/60 * 360 * Time.deltaTime, 0 ,0);
    back_left_tranform.Rotate( back_left.rpm/60 * 360 * Time.deltaTime, 0 ,0);

}
```

4.3.5 finish_line

Neste script é o usado o método Update para definir a posição da linha de chegada junto com os objetos filhos ligados a esta, sendo que este busca a posição do cubo com a tag road busca o cumprimento deste e define a posição da linha de chegada com base nesses valores com um offset de algumas unidades para que o carro nao passe para a relva.

```
private float position=0;
private float cube_lenght=0;

void Update()
{
    position =  GameObject.FindGameObjectWithTag("road").transform.position.x;
    cube_lenght= tranform.lenght;

    transform.position = new Vector3 (position + cube_lenght -20 , 8 , 501);
}
```

4.3.6 start_line

Este tem o mesmo objetivo que o finish_line que é definir a posição de partida do carro sendo que funciona da mesma forma que este com a exceção de que em vez de somar a variável cube lenght ele subtrai e adiciona um offset de 30 para começar em cima do cubo

4.3.7 Show_Warnings

O objetivo deste script é avisar o condutor caso este esteja a sair da estrada sendo que é mostrada uma mensagem a avisar que é a para retornar ao percurso antes que chegue a borda do mapa. basicamente este é composto por cubos ao redor da estrada que quando o condutor “colide” com estes ou entrar nestes é ativado um canvas, que fica escondido no início da simulação, a avisar para retornar para a estrada, depois de sair destes então a mensagem volta a desativar e ficar escondida para caso o condutor volte a fazê-lo.

```
public class Show_Warning : MonoBehaviour
{
   public GameObject Object;

    void Start()
    {
        Object. SetActive (false);
    }


    void OnTriggerEnter (){
        Object.SetActive(true);
    }


    void OnTriggerExit()
    {
    Object.SetActive(false);
    }
}
```

4.3.8 tranform

Neste são criadas variáveis globais que são a base para criar a estrada sendo que estas definem a largura, comprimento, e espessura da estrada, a última é a variável que irá ser partilhada com os outros scripts.

```
public float X_axis = 1000f;
private float Y_axis = 0.8f;
private float Z_axis = 25f;

public static float lenght = 0;
```

O único método usado neste script é o start que começa por ir buscar o comprimento da estrada ao menu de opções depois usa o tranform.localScale para mudar o tamanho da estrada e por fim usa o valor guardado no X_axis, divide por dois, e passa este para o lenght para que assim se obtenha o meio da estrada.

```
void Start()
{

  float sum=0f;
    if(Options_menu.RoadLenght!=0){
        sum=Options_menu.RoadLenght;

    }
  sum = X_axis;

  transform.localScale = new Vector3(sum ,Y_axis, Z_axis);

  lenght= sum/2;

}
```

4.3.9 signal_position

Este script pertence a um objeto vazio no Unity que é filho do start para que assim seja posicionado no início junto com o carro e com a linha de partida. Este é o que cria e posiciona o lugar onde os sinais irão aparecer.

As variáveis que tem são dois gameObjects um para criar os placares onde aparece os sinais e outro para buscar um empty object que tem o script para posicionar os sinais no placar, também tem uma variável do tipo Quaternion em que o objetivo é guardar a orientação que os gameobjects vão ter de ter para que fiquem virados para lugar certo, este tem também uma variável para saber a que distância do centro cada sinal terá de ficar, guarda uma variável para não posicionar vários sinais uns em cima dos outros que é booleana e guarda um conjunto de variáveis para começar a posicionar os sinais e definir a distância entre cada um. Os métodos que este usa são o start que busca a distância entre sinais que foi definida nas opções sendo que se não tiver nada é definido com o padrão dos testes. O ultimo método que este usa é o Update em que é verificado a variável booleana e se esta estiver a falso entra, busca o cumprimento do cubo e a sua posição calcula o número de sinais que podem ser colocados desde o início até ao fim e por fim entra num ciclo for que cria a três vetores em sendo dois deles para os sinais um para o lado esquerdo e outro para o lado direito e o terceiro vetor é para colocar o gameObject com o objeto vazio no centro, depois cada um deles é instanciado e criado usando os vetores e a variável de rotação referida acima.

```
void Update()
{
    if(enterd){
        cube_lenght=tranform.lenght;

        start_position = GameObject.FindGameObjectWithTag("road").transform.position.x;

        start_position -= cube_lenght -200;

        cube_lenght = cube_lenght * 2;


        float number_of_signal= (cube_lenght + start_position)/ dist_betwen_sig;


        for(int i=0; i< number_of_signal -1; i++){

            int aleatorio = Random.Range(30, 60);

            Vector3 position_left = new Vector3 (start_position+ i * dist_betwen_sig + aleatorio,      6f,   501 + road_tickness/2);
            Vector3 position_right = new Vector3 (start_position + i * dist_betwen_sig + aleatorio,    6f,   501 - road_tickness/2);
            Vector3 position_center = new Vector3 (start_position + i * dist_betwen_sig + aleatorio,   6f,   501);

            Instantiate(plane_prefab_, position_left, Z_90);
            Instantiate(plane_prefab_, position_right, Z_90);
            Instantiate(empty_prefab_script, position_center, Z_90);

        }

        enterd=false;
```

4.3.10 Store_values

Este como o nome indica serve para guardar os valores da posição do carro e dois sinais, este tem 3 listas onde os valores são guardados, tem dois offsets para x e z como o carro não começa nas coordenadas 0 em x nem em 0 em z então estas são necessárias para ter isso em conta, por fim este tem uma variável booleana para dizer quando deve parar ou começar a guardar os valores e tem mais três variáveis que buscam os valores das posições necessárias.

Os métodos que usa são o Start que busca StartCoroutine take_valeus e busca a posição e comprimento da estrada para calcular o offset x, tem o FixedUpdate que busca as posições do carro já retirando o offset e busca o booleano para parar ou continuar a guardar os valores e caso este seja verdadeiro então para de guardar os valores, Por fim a Coroutine verifica se pode ou não continuar a buscar os valores e armazena los nas listas e depois faz uma pausa de meio segundo

```
void Start()
{
    StartCoroutine(take_values());
    float cube_lenght=tranform.lenght;
    float start_position =  GameObject.FindGameObjectWithTag("road").transform.position.x;
    offset_x=start_position - cube_lenght;
}


void FixedUpdate()
{
    position_x = CarMovement.position_x - offset_x;
    position_z = CarMovement.position_z - offset_z;
    Real_position= spawn_images.real_position;

    store_continues = reset_level.stop_storage;

}

IEnumerator take_values()
{
    while( !store_continues){
        print("real_position: "+ Real_position);
        positions_z.Add(position_z);
        positions_x.Add(position_x);
        real_positions.Add(Real_position);
        yield return new WaitForSeconds(0.5f);

    }
}
```

4.3.11 reset_levels

Este script serve para quando o carro chegar ao fim da pista e quando o jogador clicar numa tecla em específico, ele mudar para outra cena onde os dados são escritos no ficheiro Excel. Este basicamente busca o gameObject do carro, busca as variáveis que contém o número de repetições que faltam fazer e por fim tem as variáveis para mudar de cena e parar de guardar os valores.

Este script usa o start que busca define as variáveis para guardar os valores e mudar de cena como falsas, busca os valores do número de repetições que já tem feitos e busca o carro pela sua tag.

```
void Start()
{
  sceneChanging_in_between=false;
  num_Of_Reps=Form_Menu.rep;
  repetition=Scene_changing.repetition;
  stop_storage=false;

  player = GameObject.FindGameObjectWithTag("Player");

}
```

Este também usa o método OnCollisionEnter para verificar se o carro já chegou ao fim e se sim então muda o valor das variáveis para true.

```
void OnCollisionEnter(Collision collision)
  {

      if (collision.gameObject.tag == "Player")
      {
          sceneChanging_in_between=true;
          Enterd=true;
          stop_storage=true;
      }
  }
}
```

Por fim o último método que usa é o Update que verifica se o carro já chegou ao fim e se sim então coloca a velocidade e aceleração a zeros e verifica se o player clica na letra “r” para mudar de cena.

```
void Update()
{

  if (Enterd) {//stops car

    player.GetComponent<Rigidbody>().velocity = Vector3.zero;
    player.GetComponent<Rigidbody>().angularVelocity = Vector3.zero;

    if (Input.GetKeyDown("r")){
      stop_storage=false;
      repetition++;
      print("repetition "+repetition);

      SceneManager.LoadScene("in_between");


    }
  }
}
```

4.3.12 spawn_images

Neste script é definido o conjunto de variáveis e métodos que irão criar e posicionar os sinais consoante a posição do player. Este tem uma variável que é a distância mínima que indica a que distância os sinais vão aparecer para o player junto com o um par de variáveis que indicam qual o percurso que este deve fazer Este contem tres GameObjects sendo dois deles são para guardar os sinais que permitem ou não a passagem e o outro busca o player, também contém uma lista que guarda o percurso que o carro deveria fazer quando está a mudar de faixa, tem um conjunto de variáveis que guardam quantas vezes o carro fez cada tipo de mudança de faixa sendo que cada uma delas é inicializada com 3, depois contém também os valores em que se o carro passar desses já é considerado este estar noutra faixa rodoviária ou seja contém o limite de cada faixa de rodagem, contém uma variável do tipo Quaternion que indica a direção que cada sinal irá aparecer junto com o offset que o caro tem de ter poi este não aparece no lugar (0,0,0) e por fim te uma variável que o seu objetivo é

verificar a posição do carro para saber em que faixa de rodagem este se encontra.

```
public float distanciaMin = 60f;
public static float real_curve=0.0f; // passa -1 se tiver na direita , passa 0 se tiver no centro , passa 1 se tiver na esquerda
public static float real_position=0.0f;

private GameObject player;

public GameObject passar_prefab;
public GameObject Nao_passar_prefab;

public Quaternion Z_90 =new Quaternion( 0.0f, 0.0f, 1.0f, 1.0f);

private float distancia;
private float offset= 500.01f;
private float car_position=0;
private bool enterd=false;

public float layer_2 = -4.166666667f;
public float layer_3 = 4.166666667f;

public static  int right_left=3;
public static int right_center=3;

public static int left_right=3;
public static int left_center=3;

public static int center_left=3;
public static int center_right=3;

public static List <float> curves_array= new List <float>();
private bool changed_scene;
```

Os métodos que este contém são o Start, o FixedUpdate, o verify_variables, o Instat_NaoPassar, o Instat_Passar, o perfectCurve, a coroutine Wait e o Spawn_signal irei descrever o que cada um faz abaixo:

- **Start**: Neste, primeiramente é procurado pelo carro e passado para o gameObject player, depois esta busca o valor que foi posto na distância para os sinais aparecerem e por fim a variável real_position é posta a zero

```
void Start()
{
    player = GameObject.FindGameObjectWithTag ("Player");
     if(Options_menu.ViewField!=0){
         distanciaMin = Options_menu.ViewField;
      }
    real_position=0.0f;
}
```

- **verify_variables**: este método basicamente verifica se já está a repetir a simulação outra vez e caso tal esteja a acontecer então volta a resetar as variáveis que dizem quantas vezes é que o carro pode mudar de faixa para cada lado.

- **Instat_Passar**: Este cria e posiciona os sinais que avisam o local onde o carro vai passar em ambos os lados da estrada

```
void Instat_Passar( Vector3 Side1, Vector3 Side2){
    Instantiate(passar_prefab, Side2, Z_90);
    Instantiate(passar_prefab, Side1, Z_90);
}
```

- **Instat_NaoPassar**: Este cria e posiciona os sinais que avisam o local onde o carro não deve passar, em ambos os lados da estrada
- **FixedUpdate**: Esta busca o busca o valor que reseta as variáveis que avisam quantas vezes o player pode mudar de faixa para depois ser usado, chama os métodos verify variables e Spawn_signal.
- **Wait**: Como dito este é uma coroutine e esta é o que armazena os valores intermédios quando o carro faz a transição de uma faixa para a outra, sendo que este aumenta pouco a pouco o valor até que chegue a um valor para então parar, os valores são todos adicionados numa lista e há uma pausa de 0.3 segundos entre cada valor.

```
IEnumerator Wait( int add_sub)
{
    real_curve += 0.5f * add_sub;

    if(real_curve<=1 ){
        if(real_curve>=-1) {
            curves_array.Add(real_curve);

        }
    }

    yield return new WaitForSeconds(0.3f);
}
```

- **perfectCurve**: este serve para calcular a posição que o carro deveria esta quando está a fazer a mudança de faixa e basicamente este verifica se o valor do lugar em que ele esta é maior ou menor que o valor que ele deveria estar e depois entra num ciclo que pega ni número de faixas que tem que mudar, e vai decrescendo este ate

que chegue a zero e em cada iteração é chamado o método Wait.

```
void perfectCurve(float real_position, float nmb_lane){
   if(real_position > real_curve){
      while(nmb_lane>0){

         nmb_lane-= 0.50f;
         StartCoroutine(Wait(1));
      }

   }else{

     while(nmb_lane>0){

         nmb_lane -= 0.50f;
         StartCoroutine(Wait(-1));
      }
   }
}
```

- **Spawn_signal**: Este é o método principal que faz o serviço mais importante que é posicionar os sinais e garantir que no máximo, caso não aconteça nenhum inconveniente, cada tipo de mudança de faixa só acontece 3 vezes.
  Este começa por buscar a distância a que o carro se encontra e caso esta seja igual ou menor a distância definida nas definições então entra e retira o offset a posição do carro e cria os vetores onde a posição dos sinais irão ser criadas depois é verificado em que faixa de rodagem é que o carro se encontra, tendo em conta que se ele encontrasse fora da faixa de rodagem para qualquer um dos lados este é tido em conta e os sinais aparecem para o centro ou lado oposto, depois é gerado um número aleatório e com base nesse número e valores que as variáveis para mudar de faixas têm é definido para onde o player tem de ir sendo que tem três opções sendo as duas primeiras com base no valor aleatório e a terceira é caso o player não obedeça aos sinais dentro de cada uma são chamados só métodos Instat_Passar, Instat_NaoPassar para posicionar os sinais é retirado um a variável do dado tipo de mudança de faixa que o player fez e o real_position é definido consoante a posição do player sendo 0 o centro 1 a esquerda e -1 a direita, isto é repetido para cada uma das faixas de rodagem.

```
void Spawn_signal(){
   distancia = Vector3.Distance(transform.position, player.transform.position);

   if (distancia <= distanciaMin) {

      car_position = CarMovement.position_z - offset;

      if(!enterd){
         int aleatorio =0;

         Vector3 right_side__center = new Vector3 ( transform.position.x - 0.1f, transform.position.y, transform.position.z + 20);
         Vector3 left_side__center= new Vector3 ( transform.position.x - 0.1f, transform.position.y, transform.position.z - 20);

         Vector3 right_side__left = new Vector3 ( transform.position.x - 0.1f, transform.position.y, transform.position.z + 20 +3);
         Vector3 left_side__left= new Vector3 ( transform.position.x - 0.1f, transform.position.y, transform.position.z - 20 +3 );

         Vector3 right_side__right = new Vector3 ( transform.position.x - 0.1f, transform.position.y, transform.position.z + 20 -3 );
         Vector3 left_side__right= new Vector3 ( transform.position.x - 0.1f, transform.position.y, transform.position.z - 20 -3);


         if(car_position > layer_3 ){

            Instat_NaoPassar(right_side__left, left_side__left);

            aleatorio= Random.Range(1,5);

           if(aleatorio<3 && (center_left>0 || center_right>0 )&& left_center>0)
            {
               Instat_Passar(right_side__center,left_side__center);
               Instat_NaoPassar(right_side__right, left_side__right);

               real_position=0.0f;
               left_center--;

               perfectCurve(real_position,1);

            }
            else if((right_left>0 || right_center>0) && left_right>0) // carro vai do lado esquerdo para o direita
            {
               Instat_Passar(right_side__right,left_side__right);
               Instat_NaoPassar(right_side__center, left_side__center);

               real_position=-1.0f;
               left_right--;
```

4.3.13 Scene_changing

Este começa por importar algumas bibliotecas como o “System.IO” e “UnityEngine.SceneManagement” que estes permitem escrever e ler de ficheiros, mudar de cena, as variáveis globais que este tem são repetition, number_of_reps sendo estas floats e values_stored que é um booleano e os métodos que este usa são o Start que busca o número de repetições que foi preenchido no formulário, o Update, o writeCSV e IsFileLocked.

O Update verifica se já escreveu, caso não tenha escrito então ele chama o writeCSV para tal e muda o valor do values_stored para verdadeiro para não voltar a escrever depois este espera até que a tecla “r” seja pressionada para então passar para a próxima cena verificando se já fez o número de repetições necessários ou se ainda falta fazer a simulação mais alguma vez, caso seja necessário repetir o teste então vai para a cena Lane_chance_tes se não vai para o Main_Menu.

O método writeCSV é o que pega na posição X e Z do carro e em qual faixa o carro devia estar a ir para então guardar na folha Excel. Este tem quatro Listas em que duas destas armazenam a posição do carro e as outras duas armazenam o caminho que este tem de fazer entre os sinais

```
List <float> positions_of_z= Store_values.positions_z;
List <float> positions_of_x= Store_values.positions_x;

List <float> Real_position= Store_values.real_positions;

List <float> curves =  spawn_images.curves_array;
```

Depois é criado o caminho do ficheiro que os valores vão ser escritos usando

```
string filePath = Application.dataPath + "/test.csv";
```

e antes de escrever, a lista Real_position é percorrida e cada vez que um valor muda em relação ao anterior, é calculado quantas faixas é que o carro mudou depois a lista curves é lida sempre no primeiro valor e é trocado este até que o valor a colocar seja maior e igual ao valor calculado.

```
for( int i = 0; i < Real_position.Count; i++ ){
    value=Real_position[i];

    if(value!=previous_values){
        float add= ( Math.Abs(value) + Math.Abs(previous_values))*2;// valeu entre 1 e -1 a multiplicar por 2

        for( int j = 0; (j < add ) && ((i + j) < Real_position.Count); j++ ) {

            if ((i + j) < Real_position.Count && curves.Count>0){
                print("curves : "+ curves[0] + "/// Realposition :"+Real_position[i+j]);
                Real_position[i+j]=curves[0];
                curves.RemoveAt(0);
            }
        }
        i+=(int)add;
    }
    previous_values=value;
}
```

Por fim é verificado se o ficheiro existe com o método IsFileLocked e se existir é lido uma das lista e passado os valores destas para o ficheiro pois todas tem o mesmo tamanho, por fim é fechado o ficheiro e as listas são limpas para a próxima iteração.

5.Processo de construção:

Nesta secção irei descrever o processo de construção por partes para um melhor entendimento deste.

5.1 Entendimento do problema

No início da criação do projeto antes de começar a criar o programa no Unity, foram lidos alguns artigos e tiradas algumas dúvidas com o orientador para saber o que era necessário fazer e como era suposto ser feito sendo que os artigos lidos foram [1], [3] e [6] estes ajudaram a ter uma ideia geral do teste que tinha de ser feito e o que estava a ser testado, ao retirar as duvidas com o orientador foi entendido o que tinha de ser feito e quais os objetivos a ser cumpridos.

5.2 Criação do básico

Nesta fase o básico da cena foi criado, ou seja a cubo que é a estrada e o carro, que foi retirado da Unity Assets Store, foi criado o script transform para mudar o tamanho do cubo para este ser de tamanho aleatório e um material para mudar a cor e textura deste, posteriormente o script mudado para ser de tamanho fixo e o material foi mudado para uma textura criada no Paint assim assemelhando-se a uma estrada, o script CarMovement foi criado fazendo o carro andar e virar sem que este apresente uma velocidade, houve algumas dificuldades no início, pois ao criar os wheelcolliders e atribuir estes ao carro houve um erro fazendo um dos wheelcolliders não gerar força de aceleração ou travagem e como consequência o carro virava para um dos lados, só apos uma inspeção mais aprofundada ao problema é que este foi encontrado sendo um problema de atribuição de um dos wheelcolliders.

Os painéis que representam o início e fim do teste foram criados junto com os seus scripts sendo estes a star_line e finish_line, a capacidade de fazer o carro se mover para o início da pista foi adicionada de seguida para que este não aparecesse no meio ou fora da pista.

5.3 Criação dos sinais e terrenos

Os placares onde aparecem os sinais que indicam para onde o condutor se dirigir foram criados de forma estática e independente do tamanho da estrada, havendo o problema de estes poderem aparecer logo de início e passarem do fim pois a distância entre estes não era constante, só depois usar o tamanho da estrada e usar a distância padrão é que estes começaram a aparecer dentro do intervalo entre o início e fim. As imagens dos sinais foram criadas usando o Paint pois não havia nenhuma textura semelhante na Unity Assets Store e estas são colocadas 0.1 unidades de distância a frente dos placares para que não haja problemas na renderização.

Os terrenos à volta da estrada foram criados e adicionadas algumas texturas a este, que foram encontradas na Unity Assets Store, assim não permitindo o carro cair caso sai-se da estrada.

5.5 Criação dos Menus

O Main menu, o menu de opções simples e o script para o Main menu foram criados permitindo escolher se queriam entrar ou sair da aplicação, só depois é que foi criado o formulário intermediário que permitia escolher o número de repetições a fazer e o id que o condutor iria ter.

5.6 Guardar valores

Foi criada a cena intermediária junto com o script scene_changing que verificava o número de repetições que faltava e enviava o condutor para o Main menu caso o teste tivesse acabado ou repetia este caso contrário, também foi adicionada a capacidade de guardar os valores, sendo criado o script store_values que buscava a posição do carro e onde este deveria de estar, para depois enviar e guardava os valores numa folha Excel cada vez que o teste era repetido.

O menu de opções foi modificado para que pudesse ser definido os valores da velocidade do carro, a distancia da pista, a distancia entre sinais e o campo de visualização destes assim permitindo criar testes mais personalizados.

5.7 Padronizar

Foi procurado por informações sobre os standards do LCT para que estes fossem adicionados no teste sendo que foi encontrado no artigo [6] a distancia a que os sinais deviam estar, a velocidade do carro, o cumprimento da estrada e a que distancia os sinais deveriam ser visíveis, estes foram depois partilhados com o orientador para um melhor entendimento das unidades que deveriam ser usadas.

Foram feitas modificações aos scripts que utilizassem algum desses parâmetros de forma a que estes ficassem na forma padrão sendo que uma das principais mudanças foi script que tratava de criar e posicionar os sinais, neste cada tipo de mudança de faixa só pode ser feita 3 vezes e estas são aleatórias, não permitindo com que os condutores tenham algum efeito de aprendizagem da pista, mas gerando alguns problemas pois como os sinais são gerados a partir do mesmo script teve de ser usado variáveis estáticas para que quando uma das variáveis mudasse esta mudança fosse propagada nos outros se não poderia ser selecionado sempre o mesmo lado para fazer a mudança de faixa.

5.8 Criação do caminho perfeito e finalização do projeto

Na etapa final foi mudado o script que criava os sinais e enviava a posição que o carro deveria estar de forma que este pudesse dizer o caminho intermediário quando houvesse mudanças de faixa, ao fazer tal foi necessário também mudar o script que escrevia os valores na folha Excel pois houve algumas dificuldades devido ao uso de coroutines mas o problema acabou sendo solucionado.

6.Discussão e Avaliação

O problema, como referido anteriormente é o facto de que o programa de simulação do Lane Change Test não ser muito portável nem ser fácil de fazer o setup deste pois o programa é antigo, então foi dada a tarefa de criar um programa semelhante no Unity que permitisse uma melhor portabilidade e facilidade no setup pois o Unity é conhecido por poder criar programas que funcionem em praticamente todos os aparelhos.

Podemos então dizer que o problema ficou resolvido pois agora com esta nova versão do LCT, este pode ser usado em qualquer lugar sem muitos problemas, ao mesmo tempo a nova versão criada permite aos utilizadores fazerem testes personalizados podendo alterar algumas definições destes, esta também contém avisos caso os condutores saem fora da pista parecendo uma mensagem a dizer que tem de volta, os valores da simulação são armazenados e guardados numa folha Excel para facilitar a recolha e leitura de dados destes e as pistas geradas são completamente aleatórias pois não usa pistas já feitas, em vez disso os sinais aparecem com base na posição do condutor assim evitando efeitos de aprendizagem durante os testes, o único lado negativo é que o código pode ser otimizado em várias partes para melhorar a experiência e leitura do mesmo.

Em suma este programa novo conseguiu resolver os problemas que o outro continha usando o Unity com algumas margens para melhoria, mas ao mesmo tempo com algumas melhorias que o programa anterior não continha.

7.Conclusão

Com este projeto podemos concluir que apesar de o LCT não ser a melhor ferramenta a usar este permite evitar acidentes que possam acontecer se os testes não fossem uma simulação mas ao mesmo tempo este está a ficar desatualizado sendo assim necessário que haja melhorias neste ou seja criado um novo LCT, sendo que a opção escolhida foi criar um novo programa para fazer o teste obtendo melhorias quanto a sua portabilidade, junto com algumas melhorias como a adição de um menu que permite definir alguns parâmetros do teste, adição de mensagens de aviso para quando o condutor está a sair da pista e a capacidade de escrever os dados obtidos num ficheiro Excel, isto irá permitir aos utilizadores mudar alguns parâmetros dos testes caso necessário e irá facilitar a análise de resultados recolhidos ainda assim este programa criado pode ser melhorado quanto ao código feito podendo ser melhorado as partes de recolha dos dados e as ligações entre os componentes.

Neste projeto, todos os objetivos e requisitos forma alcançados com sucesso. Além disso, a experiência adquirida permitira explorar novas metas no futuro. Ao longo do desenvolvimento, foi possível aprimorar o conhecimento do uso do Unity ena programação, compreendendo melhor as características e funcionamento dessa plataforma.

No fim podemos concluir que foi recriado as condições do LCT o melhor possível para que esta ferramenta possa ser usada, sem problemas, por todos os que necessitam e que no futuro esta possa ser melhorada em outros aspetos sendo necessário fazer mais testes e podendo ser melhorado também a nível gráfico.

Bibliografia

[1] F. Quintal e M. Lima, «HapWheel: in-car infotainment system feedback using haptic and hovering techniques», *IEEE Trans. Haptics*, vol. 15, n.º 1, pp. 121–130, 2021.
[2] S. Mattes, «The Lane-Change-Task as a Tool for Driver Distraction Evaluation».
[3] «ISO Lane Change Test», *Data Acquisition | Test and Measurement Solutions*. https://dewesoft.com/blog/iso-lane-change-test (acedido 8 de junho de 2023).
[4] «Características - Unreal Engine». https://www.unrealengine.com/pt-BR/features (acedido 9 de junho de 2023).
[5] «CRYENGINE | The complete solution for next generation game development by Crytek», *CRYENGINE*. https://www.cryengine.com/ (acedido 9 de junho de 2023).
[6] «Double-Lane Change Maneuver - MATLAB & Simulink». https://www.mathworks.com/help/vdynblks/ug/double-lane-change-maneuver.html (acedido 8 de junho de 2023).