Relatório de Estágio

Francisco Silva Gonçalves

Curso Técnico Superior Profissional em
Desenvolvimento de Aplicações Informáticas

set | 2022

GUARDA
POLI
TÉCNICO

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

CAPGEMINI ENGINEERING

RELATÓRIO DE ESTÁGIO
PARA OBTENÇÃO DO DIPLOMA DE TÉCNICO SUPERIOR PROFISSIONAL EM DESENVOLVIMENTO DE APLICAÇÕES INFORMÁTICAS

Francisco Silva Gonçalves

Setembro de 2022

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

CAPGEMINI ENGINEERING

RELATÓRIO DE ESTÁGIO
PARA OBTENÇÃO DO DIPLOMA DE TÉCNICO SUPERIOR PROFISSIONAL EM DESENVOLVIMENTO DE APLICAÇÕES INFORMÁTICAS

Professor orientador: Luís Filipe Figueiredo

Francisco Silva Gonçalves

Setembro de 2022

Ficha de Identificação

Nome: Francisco Silva Gonçalves

Número de Estudante: 1705126

CTESP: Desenvolvimento de Aplicações Informáticas

Estabelecimento de Ensino: Escola Superior de Tecnologia e Gestão

Direção do Curso: Celestino Pereira Gonçalves

Docente Orientador: Luís Filipe Figueiredo

Instituição de Estágio: Capgemini Engineering

Morada: Centro de Negócios e Serviços, Praça Amália Rodrigues, 6230-421 Fundão

Website: https://www.capgemini.com/

Supervisor na Instituição: Leonardo Nascimento

Data de Início de Estágio: 11 de março de 2022

Data de Conclusão de Estágio: 26 de julho de 2022

Duração: 750 horas

Ano Letivo: 2021/2022

Agradecimentos

Agradecer, em primeiro lugar, à entidade de estágio acolhedora pela oportunidade que me proporcionou.

Agradecer a todos os membros da empresa que me acompanharam durante o meu período de estágio, e um especial obrigado ao Leonardo Nascimento, Frederico Júnior e Daniel Neves que sempre me ajudaram quando precisei.

Agradecer a todos os meus professores por ao longo destes dois anos me terem ajudado a crescer como pessoa e um especial agradecimento ao meu professor orientador de estágio, Luís Filipe Figueiredo, pela ajuda que me forneceu durante a realização deste relatório.

Agradecer aos meus colegas pelo apoio que me ofereceram durante este período.

Agradecer ao meu cunhado, Guilherme Afonso, por me ter ajudado a ultrapassar alguns obstáculos e assim conseguir progredir e fazer melhor o meu trabalho.

Agradecer às pessoas mais importante da minha vida, Miguela Gonçalves e Beatriz Gonçalves, que sempre me incentivaram a fazer mais e melhor e me ajudaram em tudo o que eu precisei ao longo deste período.

Agradecer por fim ao meu anjo da guarda, o meu pai, que estará sempre a olhar por mim e certamente estará muito orgulhoso de mim.

Plano de estágio

O plano de trabalho consistiu em desenvolver simulações de cenários de V2X.

V2X é um sistema complexo de simulações de cenários de condução autónoma que está a evoluir rapidamente.

Os sistemas automóveis de hoje utilizam os Sistemas Avançados de Assistência ao Condutor (ADAS) para potenciar a segurança dos seus veículos.

Desta forma, o sistema de simulações de cenários V2X é necessário para tornar as experiências de condução mais previsíveis, indo assim ao encontro do objetivo dos ADAS, potenciar a segurança dos veículos. As simulações de cenários foram desenvolvidas utilizando o simulador “CARLA Simulator” e linguagem de programação Python.

A estratégia de *ramp-up* consistiu na revisão dos conceitos Python, SOLID e Design Patterns.

Resumo

O estágio referente ao término do Curso de Técnico Superior Profissional em Desenvolvimento de Aplicações Informáticas teve início no dia 11 de março de 2022, com uma duração total de 750 horas, o qual terminou no dia 26 de julho de 2022.

O objetivo do estágio foi o desenvolvimento de simulações de cenários V2X num simulador denominado CARLA *Simulator*. De forma a atingir esse objetivo, fui inserido na iniciativa/projeto da Capgemini Engineering denominada "V2X initiative at the Embedded and Software Critical Systems unit".

Numa fase inicial, para poder ser inserido neste projeto, foram-me atribuídas variadas formações, nomeadamente sobre C++ e Python, com o objetivo de me familiarizar com as linguagens de programação a utilizar e para também conhecer as boas práticas a serem aplicadas durante o desenvolvimento do código. Posteriormente, a única linguagem utilizada foi Python.

Estas simulações de cenários V2X tiveram como foco a deteção de veículos que estariam no campo de visão de outro, recolhendo assim todos os dados desse veículo, como velocidade, travagem, alteração da trajetória, entre outros parâmetros.

Durante o estágio pude ser confrontado com problema reais que ocorrem nesta área de trabalho e por consequência, aprender a melhor estratégia para a sua resolução.

Este estágio permitiu também evoluir o meu pensamento em relação à minha área de trabalho, e assim ficar mais claro de que programação é o que quero realizar durante a minha carreira profissional.

Overview

The internship for the completion of the Professional Higher Technician Course in Software Application Development started on March 11, 2022, with a total duration of 750 hours, which ended on July 26, 2022.

The scope of the internship was the development of V2X scenario simulations in a simulator called CARLA Simulator. In order to achieve this, I was inserted in Capgemini Engineering's initiative /project called "V2X initiative at the Embedded and *Software* Critical Systems unit".

At an early stage, in order to be included in this project, I was given diverse courses namely on C++ and Python so that I could be familiar with the programming languages I would be using and to learn good practices to be applied during the development of the code. Later, the only language used was Python.

These V2X scenario simulations had the objective of detecting which vehicles would be in the field of view of another one, thus collecting all the vehicle's data, such as speed, braking, changes of path, among other parameters.

During this internship, I was able to be confronted with real problems that occur in this area of work and in turn learn the best strategy for solving them.

This internship also allowed to evolve my thinking in relation to my area of work and thus become clearer that programming is what I want to accomplish during my professional career.

“Não é a linguagem de programação que define o programador, mas sim a sua lógica.”

David Ribeiro Guilherme

Índice

Índice de figuras

Glossário de sílabas e abreviaturas

ADAS – Advanced Driver Assistance Systems

CMD – Command Prompt

CPU – Central Processing Unit

GNSS - Global Navigation Satellite Systems

ID – Identity

IDE - Integrated Development Environment

LIDAR - Light Detection and Ranging

MQTT - Message Queuing Telemetry Transport

URL – Uniform Resource Locator

V2X - Vehicle to Everything

Capítulo I

Introdução

1. Contexto do Relatório

Este relatório pretende descrever o estágio realizado na empresa Capgemini Engineering, com a finalidade de concluir o Curso Técnico Superior Profissional em Desenvolvimento de Aplicações Informáticas realizado na Escola Superior de Tecnologia e Gestão no Politécnico da Guarda.

Ao longo do relatório serão descritas as atividades e o projeto principal onde fui inserido, algumas informações sobre a empresa acolhedora, ferramentas utilizadas e o trabalho desenvolvido ao longo desse período de tempo.

2. Fundamentação da escolha do estágio

Com a finalidade da realização do estágio e por sua vez da finalização do curso, foram tidas em consideração empresas relacionadas com a área do desenvolvimento de *software*, visto ser a área de maior interesse para o meu percurso profissional.

Com a decisão de escolha de empresas neste ramo, o estágio foi realizado na empresa Capgemini Engineering, tendo sido inserido na área de trabalho desejada, desenvolvimento de *software*.

3. Organização do documento

Este documento divide-se em seis capítulos: “Introdução”, “Empresa”, “Estágio”, “Ferramentas utilizadas”, “Trabalho desenvolvido” e “Conclusão”.

No primeiro capítulo, Introdução, é feito um sumário do relatório, descrevendo o conteúdo do mesmo e fundamentando a escolha do estágio.

No segundo capítulo, Empresa, encontra-se uma breve apresentação da Capgemini Engineering, a sua atividade, história, ambições e valores e respetivo espaço físico.

No terceiro capítulo, Estágio, encontra-se uma breve descrição sobre o objetivo do estágio e o processo de integração no projeto.

No quarto capítulo, Ferramentas utilizadas, é possível encontrar a lista de ferramentas utilizadas ao longo do estágio e uma breve descrição da utilização das mesmas.

No quinto capítulo, Trabalho desenvolvido, encontra-se descrito o trabalho realizado ao longo do estágio, trabalho não dedicado ao projeto e trabalho para o projeto.

No último capítulo, Conclusão, encontram-se as considerações finais sobre o estágio, o impacto no meu futuro profissional e uma opinião final.

Capítulo II

Empresa

1. A Empresa

A Capgemini é uma empresa multicultural que trabalha com clientes de renome mundial por três submarcas especializadas, Capgemini Invent, Sogeti e Capgemini Engineering.

As suas principais linhas de serviço são: Engenharia de Sistemas de Produtos, *Software* Digital e Operações Industriais.

Os seus setores de desenvolvimentos são: Aeronáutica, Espaço, Defesa, Naval, Automóvel, Ferroviário, Infraestruturas e Transportes, Energia, Utilidades e Químicos, Ciência da Vida, Comunicação, Semicondutores e Eletrónica, Industrial e Consumidor e *Software* e Internet.

A empresa conta com mais de cinquenta e dois mil engenheiros e cientistas por todo o mundo, mais de trezentos e quarenta mil funcionários em mais de cinquenta países e encontra-se no top um da investigação e desenvolvimento há mais de trinta anos.

O grupo reportou em 2021 receitas globais de mais de dezoito mil milhões de euros.

Figura 1 - Logotipo Capgemini Engineering.

2. Ambições, Valores e Objetivo

A Capgemini Engineering tem como ambição tornar-se o líder global da indústria de produtos inteligentes e tem como valores a honestidade, audácia, confiança, espírito de equipa, liberdade, divertimento e modéstia.

A Capgemini Engineering tem também como objetivo libertar a energia humana através da tecnologia para um futuro inclusivo e sustentável.

3. Caracterização do espaço físico

A Capgemini Engineering conta com quatro espaços físicos, em Lisboa, Porto, Fundão e Tunísia.

O espaço acedido durante o presente estágio foi o escritório do Fundão, constituído por mais de seis mil metros quadrados, seis salas de reuniões, mais de trinta salas para projetos e conta com mais de quinhentos espaços de trabalho individuais.

4. Horário de trabalho

A Capgemini Engineering dá aos seus funcionários uma grande flexibilidade em termos de horário, sendo que os horários podem variar bastante, mantendo o cumprimento das oito horas de trabalho diárias e o cumprimento dos prazos estipulados.

O horário de trabalho realizado durante estágio era compreendido entre as nove da manhã até às seis da tarde, com uma hora de pausa para almoço.

Capítulo III

Estágio

1. Objetivo

Como já referido, o estágio decorreu sobre o projeto/iniciativa "V2X initiative at the Embedded and Software Critical Systems unit", sendo o foco da mesma o desenvolvimento de simulações de cenários V2X.

O objetivo inicial do estágio foi desenvolver, através do simulador CARLA e da linguagem de programação Python, simulações de cenários onde o carro era o integrante principal e tudo à sua volta o integrante secundário.

O objetivo final do projeto seria ter uma aplicação que executasse a biblioteca Pygame, onde fosse possível escolher o trajeto de dois carros e assim fazer a comparação dos valores gerados no simulador pelos mesmos.

O objetivo final passou também pela documentação dos *softwares* criados, de forma a obter repositórios no GitHub que retratassem todos os passos de utilização do *software* desenvolvido e algumas demonstrações dos mesmos.

2. Integração do Estagiário

A Capgemini Engineering conta com um processo de integração para todos os novos membros que consiste numa apresentação inicial onde é mostrado de modo geral os principais elementos da empresa, atividade da empresa, os seus valores, ambições e objetivos numa primeira fase. Numa segunda fase da apresentação são apresentadas todas as ferramentas utilizadas internamente, de forma a registar as horas de trabalho, marcação de férias, resolver problemas experienciados com alguma aplicação, entre outras ferramentas. Por fim, é apresentado o escritório, fornecido um kit inicial com alguns objetos de interesse e é também fornecido um portátil de trabalho.

3. Ramp-up

Para integração no projeto, foi, inicialmente, necessário recorrer a um *ramp-up* que consistiu em diversas formações acerca das linguagens C++ e Python.

Posteriormente foi também necessário a leitura de documentação acerca do simulador que iria ser utilizado durante o estágio, CARLA simulator.

4. Cronograma

O estágio teve data de início a 11 de março de 2022 e terminou a 26 de julho de 2022, completando assim um total de 750 horas.

Capítulo IV

Ferramentas e Linguagens utilizadas

1. Ferramentas utilizadas

Neste tópico encontram-se listadas todas as ferramentas utilizadas ao longo do período de estágio, contendo uma breve descrição das mesmas.

1.1. Microsoft Teams

O Microsoft Teams foi a aplicação utilizada para a comunicação entre todos os membros do projeto, agendar reuniões entre a equipa e onde todos os dias era realizada uma reunião diária de atribuição de tarefas a serem realizadas ao longo do dia.

Figura 2 - Logotipo Microsoft Teams.

1.2. Visual Studio Code

O Visual Studio Code foi o Integrated Development Environment (IDE) utilizado para realizar o desenvolvimento do código necessário para o projeto. Este IDE permite a utilização de diversas linguagens de programação, no entanto a linguagem maioritariamente utilizada foi Python.

Figura 3 - Visual Studio Code.

1.3. Git

O Git é um dos sistemas de controlo de versão mais utilizados no mundo do desenvolvimento de *software*, sendo um projeto de código aberto desenvolvido em 2005 por Linus Torvalds e é utilizado para controlar o histórico de alterações nos ficheiros.

Durante o estágio foi utilizado o Git para criação e alteração de *branches*, clonagem de repositórios e efetuar *commits*, *pushs* e *pulls*, tanto para o GitLab como para o GitHub. Estas são algumas das opções que o Git disponibiliza e que foram mais utilizadas no estágio.

Figura 4 - Logotipo Git.

1.4. GitLab

O GitLab é uma plataforma de hospedagem de código-fonte, permitindo que os desenvolvedores contribuam em projetos privados ou públicos. Nesta plataforma, cada projeto contém um repositório, permitindo a divisão do código de desenvolvimento, por exemplo, entre *front-end* e *back-end*.

A utilização desta ferramenta durante o estágio serviu para desenvolver o código já existente do projeto e salvar novo código desenvolvido.

Figura 5 - Logotipo GitLab.

1.5. GitHub

O GitHub é uma plataforma idêntica ao GitLab, funcionando da mesma forma e praticamente com os mesmos recursos. Esta plataforma esteve envolvida no estágio, no sentido em que foi utilizada para a criação dos repositórios finais sobre o trabalho desenvolvido no simulador, tópico que será discutido posteriormente.

Figura 6 - Logotipo GitHub.

1.6. Pluralsight

O Pluralsight é uma plataforma que fornece formações online sobre diferentes tipos de linguagens de programação, fornecendo bastante documentação e tutoriais acerca da linguagem pretendida, ajudando assim os novos membros das equipas na integração no projeto.

A utilização desta plataforma durante o estágio esteve presente nos primeiros dias, fornecendo toda a informação sobre as linguagens que iriam ser utilizadas no projeto.

Figura 7 - Logotipo Pluralsight.

1.7. Burp Suite

O Burp Suite é um conjunto de ferramentas usadas para a realização de testes de penetração em aplicações web, fornecendo oito ferramentas destintas: Aranha, Proxy, Intruso, Repetidor, Sequenciador, Descodificador, Extensor e Scanner, tendo cada uma delas uma função especifica.

No caso do presente estágio, esta ferramenta não foi utilizada no projeto. Esta ferramenta foi apenas apresentada com o objetivo de a conhecer através de demonstrações de algumas das suas utilizações.

Figura 8 - Logotipo Burp Suite.

1.8. Ubuntu

O Ubuntu é um sistema operativo construído a partir do núcleo Linux, que possibilita a execução de programas num computador. É um sistema de código aberto baseado nas normas de *software* livre.

A utilização do mesmo no contexto do estágio, teve como função a execução do simulador CARLA e respetiva execução dos exemplos criados para o mesmo, que irão ser abordados posteriormente.

Figura 9 - Logotipo Ubuntu.

1.9. Pygame

O Pygame é uma biblioteca de jogos multiplataforma, criada para ser utilizada em conjunto com a linguagem de programação Python, fornecendo acesso a áudio, teclado, comandos, rato e *hardware* gráfico via OpenGL e Direct3D.

Figura 10 - Logotipo Pygame.

2. Linguagens utilizadas

Neste tópico encontram-se listadas as linguagens de programação utilizadas ao longo do período de estágio, contendo uma breve descrição das mesmas.

2.1. Python

Python é uma linguagem de programação de alto nível, ou seja, uma linguagem com uma sintaxe mais simplificada e que se aproxima da linguagem humana, sendo utilizada para diversas aplicações como *desktop*, *web*, servidores e ciência de dados.

No contexto do estágio, foi utilizada esta linguagem de programação para o desenvolvimento das simulações no CARLA simulator, que serão mostrados mais à frente.

Figura 11 - Logotipo Python.

2.2. C++

C++ é uma linguagem de programação orientada a objetos, utilizada para o desenvolvimento de *embedded systems*, bibliotecas gráficas, jogos, sistemas operativos, entre outros, sendo baseada na linguagem C e por essa razão, é uma extensão dessa linguagem.

Esta linguagem permite que os sistemas desenvolvidos nela sejam de alto desempenho, estáveis e seguros. No entanto, esta linguagem apenas foi utilizada para as formações iniciais de integração.

Figura 12 - Logotipo C++.

3. Verificadores da qualidade do código

Ao longo do desenvolvimento de código em Python foi possível detetar alguns *bugs*, erros de lógica e erros de sintaxe, e para ser possível resolver esses problemas foram utilizadas duas ferramentas de verificação de código, Pylint e Pyflakes.

3.1. Pylint

Pylint é um verificador de *bugs* e verificador da qualidade do código fonte para a linguagem de programação Python.

A sua utilização foi feita ao longo do estágio, visto que o projeto a ser desenvolvido utiliza a linguem de programação Python.

De entre diversas verificações que o Pylint executa, as mais utilizadas foram a verificação do nome de classes e variáveis, verificação do comprimento de uma linha de código e verificação de possíveis *bugs* futuros devido a incoerências no código.

3.2. Pyflakes

Pyflakes é uma ferramenta idêntica ao Pylint que também procede à verificação de erros de lógica e de sintaxe na linguagem de programação Python.

Capítulo V

Trabalho Desenvolvido

1. Formações Iniciais

Inicialmente foram atribuídas algumas formações a serem realizadas sobre as linguagens de programação C++ e Python, através da plataforma Pluralsight, que serviriam como base para o projeto a ser desenvolvido.

1.1. Formação em C++

Nas formações sobre C++ foi possível ter uma visão global de como funciona a linguagem, começando pelas sintaxes básicas de escrita de variáveis, utilização de operadores, envio de mensagens e compilação dos ficheiros em Linux e Windows através da linha de comandos.

Posteriormente, foram realizadas formações acerca de Templates, Ponteiros e Referências e Gestão de memória, sendo estes os tópicos que levaram inicialmente mais tempo a compreender.

1.2. Formação em Python

Nas formações sobre Python, foram apenas realizadas as formações iniciais sobre como funciona a linguagem e a sua forma de escrita, visto ser uma linguagem de alto nível e ter uma forma de escrita diferente do habitual.

Após a formação inicial, foram propostos alguns desafios para serem resolvidos, que serão apresentados nos tópicos seguintes.

1.2.1. Desafio em Python – Calculadora

O primeiro desafio proposto foi a criação de uma calculadora que, à partida, parece um exercício bastante simples, mas à medida que vamos pensando em mais pormenores, reparamos que é um exercício bastante desafiador.

Inicialmente, o foco deste desafio era a criação de uma calculadora simples: quatro funções para efetuar os quatro principais cálculos, soma, subtração, multiplicação e divisão. Assim, foi feito o desenvolvimento de uma calculadora onde era pedido ao utilizador que inserisse o primeiro número, de seguida um dos quatro operadores e por fim, o segundo número.

Todo este processo contou com verificações, para casos onde por exemplo, é solicitado um número e o utilizador insere uma letra, casos onde o utilizador não insere um operador válido ou casos onde o utilizador tenta efetuar uma divisão por zero.

Na Figura 13, é apresentado o código do ficheiro principal, que chama as funções de fazer o pedido dos números ao utilizador, o operador do cálculo e apresentar o resultado.

```
def main():

    #Request numbers and operator
    n1 = GetNums("Insert the first number:")
    op = GetOP()
    n2 = GetNums("Insert the second number:")

    #Execute func for calc
    if op == '+':
        print("Result = " + str(sum(n1,n2))
    elif op == '-':
        print("Result = " + str(sub(n1,n2))
    elif op == 'x':
        print("Result = " + str(mult(n1,n2))
    elif op == '/':
        print("Result = " + str(div(n1,n2))

if __name__ == "__main__":
    main()
```

Figura 13 - Ficheiro Principal.

Na Figura 14, Figura 15 e Figura 16, é apresentado o código presente no ficheiro que tem todas as funções a serem utilizadas na calculadora, nomeadamente a função dos cálculos, a função para pedir os números ao utilizador e a função para pedir o operador.

```
#functions to run the calculation
def sum(num1,num2):
    return num1 + num2

def sub(num1,num2):
    return num1 - num2

def mult(num1,num2):
    return num1 * num2

def div(num1,num2):
    if num2 == 0:
        print("Impossible to divide a number by zero!")
    else:
        return num1 / num2
```

Figura 14 – Funções dos cálculos.

```
def GetNums(question):
    num = None
    while num == None:
        try:
            num = int(input(question))
        except ValueError:
            print("Please insert a valid number!")
    return num
```

Figura 15 - Funções para pedir primeiro e segundo número

```
def GetOp():
    while True:
        op = input("Choose the operation(type ? to see operations):")
        if op == '?':
            print("Operations that you can do: + , - , x , /")
        elif op == '+' or op == '-' or op == 'x' or op == '/':
            return op
        print("Choose a correct operation!")
```

Figura 16 – Função para pedir operador.

Na Figura 17, é apresentada uma pequena demostração do funcionamento do programa.

```
Insert the first number:b
Please insert a valid number!
Insert the first number:1
Choose the operation(type ? to see operations):b
Choose a correct operation!
Choose the operation(type ? to see operations):?
Operations that you can do: + , - , x , /
Choose the operation(type ? to see operations):+
Insert the second number:2
Result = 3
```

Figura 17 - Demonstração do funcionamento do programa.

1.2.2. Desafio em Python – Lista ordenada para ficheiro texto

O segundo desafio proposto consistia em pedir ao utilizador para inserir números compreendidos entre 0 e 1000, até ser introduzida a palavra “stop” pelo utilizador, que no caso era a palavra definida para terminar o programa. Após isso, eram criados dois ficheiros texto, sendo que o primeiro continha todos os números pela ordem que foram introduzidos pelo utilizador e o segundo continha todos os números ordenados por ordem crescente.

Todo este processo contou com validações, para casos onde por exemplo, o utilizador não introduzir números entre 0 e 1000, ou no caso de o utilizador não introduzir um número válido, como por exemplo a introdução de alguma letra ou caracter especial.

Este desafio foi também dividido em dois ficheiros como no primeiro desafio que foi realizado, sendo um o ficheiro principal onde eram criados os ficheiros texto, onde era chamada a função de introdução dos números e feita a escrita dos números para o ficheiro texto, e o segundo ficheiro que contem a função que realiza a recolha dos números introduzidos e respetivas validações e a função de organização dos números por ordem crescente.

Na Figura 18, é apresentado todo o código presente no ficheiro principal do programa.

```python
from CreateTXTFunc import *

def main():

    #create files
    txtNumList = open("NumList.txt","w+")
    txtNumOrdList = open("NumOrdList.txt","w+")

    #Get the return list values
    num = Allcode()

    #write list on file and close file
    txtNumList.write(str(num) + "\n")
    txtNumList.close()

    #write list in ascending order and close file
    txtNumOrdList.write(str(sorted_francisco(num)) + "\n")
    txtNumOrdList.close()

if __name__ == "__main__":
    main()
```

Figura 18 - Ficheiro Principal.

Na Figura 19 e Figura 20, é apresentado todo o código nos ficheiros que contêm todas as funções utilizadas no programa.

```python
def Allcode():

    #start var values
    num = []
    insert = ''
    cont = 1

    #asks for numbers as long as the word stop is not inserted
    while insert != "stop":
        try:
            insert = input("Insert the " + str(cont) + "° number: ")
            #if user type stop, end
            if insert == "stop":
                break
            #if number not between 0 and 1000
            elif int(insert) < 0 or int(insert) > 1000:
                print("The numbers can only be between the values 0 and 1000")
            else:
                #if is a valid number
                try:
                    num.append(int(insert))
                    cont += 1
                    #if last position of list is stop, delete
                    if num[-1] == "stop":
                        num.pop()
                #not valid number
                except ValueError:
                    print("Insert a valid number! If you want to stop type stop")
        except:
            print("Insert a valid number! If you want to stop type stop")
    return num
```

Figura 19 – Função para pedir números e executar validação.

```
def sorted_francisco(num):

    for i in range(len(num)):
        for j in range(len(num)):
            if num[i] < num[j]:
                controlVal = num[i]
                num[i] = num[j]
                num[j] = controlVal

    return num
```

Figura 20 - Função para ordenar números.

A linguagem Python já conta com uma função de ordenação de números denominada sort(), no entanto, foi criada uma própria função sort() para ser possível entender o funcionamento da mesma.

Na Figura 21, é apresentada uma demostração do funcionamento do programa.

```
Insert the 1º number: b
Insert a valid number! If you want to stop type stop
Insert the 1º number: 10000
The numbers can only be between the values 0 and 1000
Insert the 1º number: 1
Insert the 2º number: -1
The numbers can only be between the values 0 and 1000
Insert the 2º number: 20
Insert the 3º number: 13
Insert the 4º number: 2
Insert the 5º number: 1004
The numbers can only be between the values 0 and 1000
Insert the 5º number: 1000
Insert the 6º number: stop
```

Figura 21 - Demonstração do funcionamento do programa.

Na Figura 22, é apresentado o resultado da execução do programa que cria os dois ficheiros texto, sendo o primeiro ficheiro o que contém os números pela ordem que foram introduzidos e o segundo ficheiro o que contém os números ordenados.

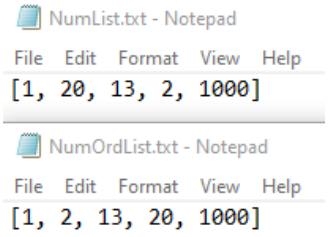

Figura 22 - Ficheiros texto que são criados.

2. Protocolo MQTT

Ao longo do estágio, além de trabalho prático, também foram abordados alguns temas apenas teóricos e o Protocolo *Message Queuing Telemetry Transport* (MQTT) foi um desses tópicos.

O Protocolo MQTT é um protocolo de mensagens criado com o objetivo de oferecer um baixo consumo de rede, de banda e dos demais recursos de *software*, sendo o seu formato de utilização Cliente/Servidor.

2.1. Como funciona o Protocolo MQTT

O funcionamento do Protocolo MQTT utiliza um modelo *Publish*/*Subscribe* que permite ao cliente enviar e/ou receber informações enquanto o servidor administra essa informação. Sendo assim, neste protocolo existirá um ou mais publicadores que são os responsáveis por publicar as mensagens num determinado tópico, e um ou mais subscritores que irão receber os dados atualizados desse tópico.

Como não existe uma ligação direta entre o subscritor e o publicador, é necessário um gerenciador de mensagens denominado *Broker*.

2.2. Termos do Protocolo MQTT

Subscriber – Pessoa que está inscrita no tópico e que tem o papel de recetor.

Publisher – Pessoa que irá enviar os dados para um tópico.

Broker – Ponte de comunicação entre o *Publisher* e o *Subscriber*, responsável pela receção e envio das mensagens.

Tópico – Endereço pelo qual as mensagens serão enviadas.

Cliente – Elemento que tem a capacidade de interagir com o *Broker*.

Mensagem – Pacote de dados que são trocados entre Cliente e *Broker*.

Unsubscribe – Permite remover a assinatura a um determinado tópico.

Payload – Conteúdo da mensagem a ser enviada.

3. Burp Suite

Burp Suite foi uma ferramenta que também como o Protocolo MQTT, apenas foi abordado teoricamente, e como já referir num tópico anterior, o Burp Suite é um conjunto de ferramentas usadas para a realização de testes de penetração em aplicações *web*.

Em relação ao Burp Suite, foi abordada a sua utilização, de forma a ver e modificar o conteúdo dos pedidos e respostas enquanto estão em circulação e para a descodificação de *Uniform Resource Locator* (URLs).

4. Design Patterns

Design Patterns são soluções comuns para problemas que ocorrem normalmente no desenvolvimento de *software*. São como plantas pré-fabricadas que se podem personalizar para resolver um problema que é recorrente no código.

Entre um vasto número de *design patterns* diferentes, durante o meu estágio abordei dois tipos, sendo eles o *Singleton* e o *Factory Method*.

4.1. Singleton

Singleton é um *design pattern* que permite assegurar que uma classe tenha apenas uma instância, ao mesmo tempo que proporciona um ponto de acesso global a esta instância.

As vantagens deste *design pattern* passam por ser possível ter a certeza de que uma classe tem apenas uma única instância, e é adquirido um ponto de acesso global a essa instância e o objeto *Singleton* apenas é inicializado quando é chamando pela primeira vez.

Na Figura 23, é apresentado um pequeno exemplo de uma classe *Singleton*, em Python, que foi criada durante a abordagem a este *design pattern*.

```
class Singleton:
   __instance = None
   @staticmethod
   def getInstance():
      """ Static access method. """
      if Singleton.__instance == None:
         Singleton()
      return Singleton.__instance
   def __init__(self):
      """ Virtually private constructor. """
      if Singleton.__instance != None:
         raise Exception("This class is a singleton!")
      else:
         Singleton.__instance = self
s = Singleton()
print (s)

s = Singleton.getInstance()
print (s)
```

Figura 23 - Exemplo do Design Pattern Singleton.

Na Figura 24, é apresentada a estrutura do *design pattern*.

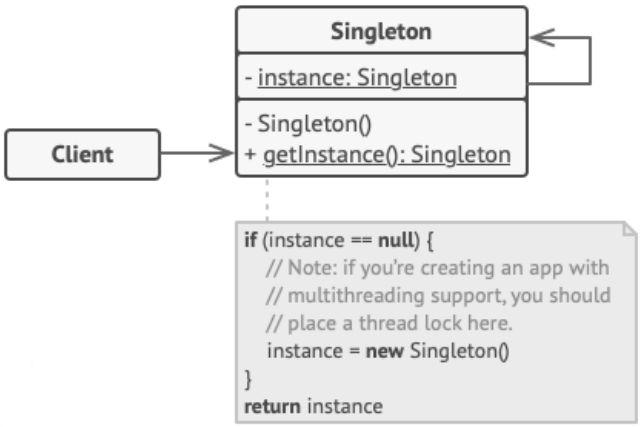

Figura 24 - Estrutura Singleton.

4.2. Factory Method

Factory Method é um *design pattern* que fornece uma interface para a criação de objetos numa superclasse, mas permite às subclasses alterar o tipo de objetos que serão criados.

Na Figura 25, é apresentada estrutura do *design pattern*.

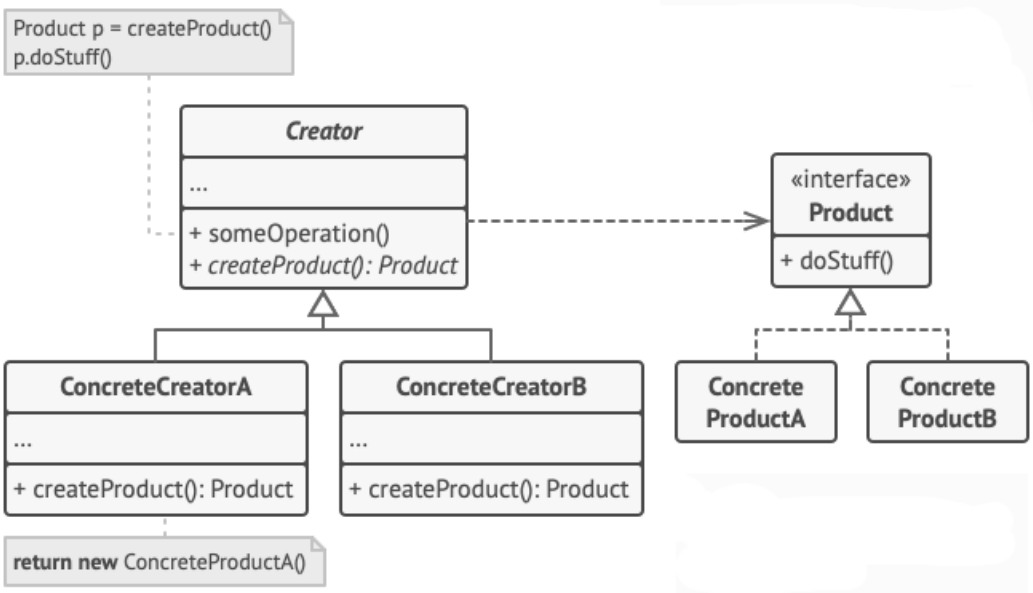

Figura 25 - Estrutura *Factory Method*.

4. SOLID

SOLID foi mais uma abordagem teórica que decorreu ao longo do estágio, sendo que SOLID é um acrônimo para os cinco princípios da programação orientada a objetos, que surgiu para facilitar o desenvolvimento de *software*. Cada letra do termo corresponde a um princípio e tem uma designação própria.

Nos tópicos seguintes é apresentada uma breve explicação de como é utilizado e para que serve a utilização de cada princípio.

4.1. Letra S – Single Responsibility Principle

O princípio da responsabilidade única diz que cada classe deve ter um, e somente um motivo para mudar, declarando que uma classe deve ser feita apenas para um único assunto e possuir apenas uma responsabilidade dentro do *software*.

4.2. Letra O – Open-Closed Principle

O princípio aberto-fechado diz que objetos ou entidades devem estar disponíveis para uma extensão, mas fechados para modificações, ou seja, quando é necessário adicionar novos recursos no *software*, devemos poder acrescentar código sem a alteração do código original.

4.3. Letra L – Liskov Substitution Principle

O princípio da substituição de Liskov diz que uma classe derivada deve ser possível de ser substituída pela classe principal sem que seja necessário a alteração das propriedades do *software*.

4.4. Letra I – Interface Segregation Principle

O princípio da segregação da interface diz que uma classe não deve ser forçada a implementar interfaces e métodos que não serão utilizados, ou seja, é preferível a criação de uma interface mais específica do que uma interface genérica.

4.5. Letra D – Dependency Inversion Principle

O princípio da inversão de dependências possui duas definições.

A primeira definição diz que módulos de alto nível não devem depender de módulos de baixo nível e sim, ambos devem depender de abstrações.

A segunda definição diz que abstrações não devem depender de detalhes, mas sim que os detalhes é que devem depender de abstrações.

Em resumo, se tivermos uma classe que é chamada dentro de outra classe devemos optar pela criação de um método em vez de uma classe, respeitando assim este princípio.

5. Notação BigO

Numa fase inicial do estágio, devido à criação de código bastante complexo, foi apresentada a notação BigO.

A notação BigO é uma notação matemática que descreve a complexidade do código, baseada no tempo e espaço.

Na Figura 26, é apresentado um gráfico que mostra algumas notações de BigO, consoante o seu tempo e tamanho.

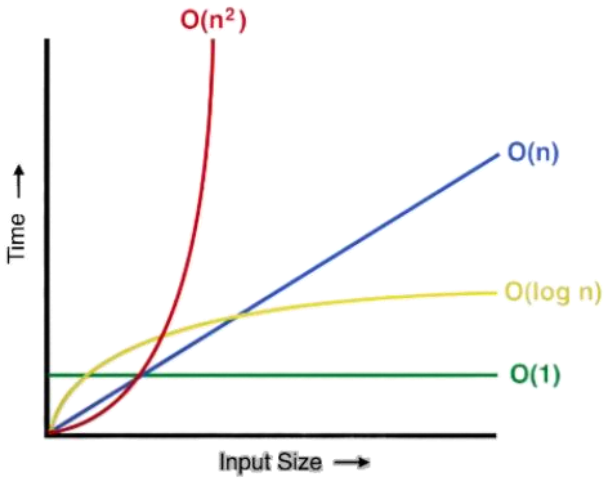

Figura 26 - Notação BigO.

6. Event Driven

O *Event Driven* é um ciclo de ações que está sempre à espera de receber um novo evento para tratamento. Este modelo foi aplicado no projeto e utilizado através da biblioteca Pygame, sendo que a própria já conta com o seu *event driven* que foi utilizado no *software* "CARLA Ego Vehicle - 2D Path's", que será abordado posteriormente, para receber os eventos de teclas pressionadas pelo utilizador.

7. Thread

As *Threads* são entidades executadas no *Central Processing Unit* (CPU) juntamente com outras *Threads*, que permitem que várias execuções possam ocorrer no mesmo software, sendo elas independentes umas das outras. Com isto, as *thread* tornam mais simples o modelo de programação à medida que vamos decompondo o nosso *software* por diferentes *threads* que podem ser executadas quase ao mesmo tempo dependendo do número de CPU's.

8. Callback

O *Callback* é uma função que é executada quando algum evento acontece ou quando o *software* chega a um estado desejado, sendo que o *Callback* cria regras dentro de outras funções para que sejam utilizadas posteriormente.

O *Callback* é muito comum na linguagem de programação JavaScript, por exemplo, durante a criação de animações, isto porque permite ao programador especificar o que deve ocorrer quando um estado é alcançado.

9. GitHub action

O GitHub *action* é uma ferramenta que ajuda na automatização do fluxo de trabalho dentro do GitHub. Com isso, é possível a criação de uma *action* para ser realizada sempre que existe um *commit* ou um *push* e executar uma certa ação, tendo essa ação de ser configurada no repositório do GitHub.

No caso prático que foi utilizado durante o estágio, foi criada uma *action* para fazer a instalação de todas as bibliotecas necessárias para o *software* criado, sempre que um novo *push* para a *branch* era efetuado e também uma verificação utilizando o Pylint.

Na Figura 27, é apresentada a lista de todos os passos que eram executados pela *action* criada.

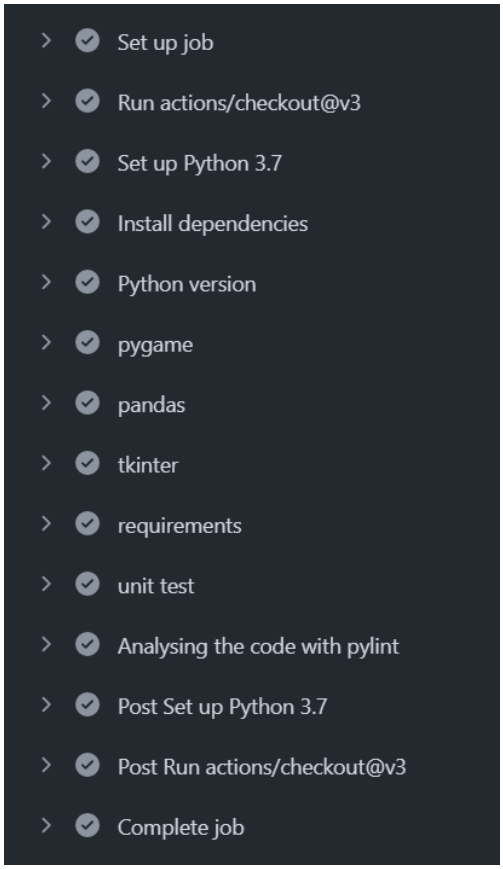

Figura 27 - GitHub Action.

10. Projeto V2X

Durante o estágio fui inserido no projeto V2X, como já referido, onde foi realizado o trabalho principal proposto como plano de estágio. Para tal, foi utilizada a linguagem de programação Python e também o simulador CARLA.

Todos os tópicos abordados anteriormente serviram como base de preparação para este projeto.

Ao longo deste tópico será descrito tudo o que foi realizado no projeto, desde documentação do simulador até ao projeto final.

10.1. CARLA Simulator

O CARLA é um simulador 3D que foi desenvolvido para apoio no desenvolvimento, formação e validação de sistemas de condução autónomos. O CARLA fornece diferentes designs de meios urbanos, edifícios e veículos, que foram criados para serem utilizados de forma livre nas simulações.

O CARLA fornece ainda um conjunto de diversas ferramentas que podem ser implementadas, como diferentes sensores, sensor *Light Detection and Ranging* (LIDAR), sensor de Colisão, sensor *Global Navigation Satellite Systems* (GNSS), entre outros, assim como diferentes condições ambientais, controlo de todos os veículos, criação de mapas personalizados, e muito mais, mas sendo estes os principais a serem utilizados durante o projeto.

Todos os exemplos correm dentro de um *loop* infinito para que vários comandos possam ser executados durante a duração da simulação.

10.2. Requirements

Os softwares criados necessitam de diferentes bibliotecas para que o utilizador possa executar o *software* sem problemas, e para tal existe um ficheiro denominado "Requirements.txt" que será utilizado para a instalação de todas as bibliotecas necessárias. Para tal, é necessário correr o comando "pip install -r requirements.txt" no *Command Prompt* (CMD), dentro do diretório onde se encontram todas as simulações.

10.3. CARLA Ego Vehicle - Simulation to Excel

Ao longo da criação dos diferentes tipos de exemplos foi determinado que seria procedida à criação de um projeto no GitHub, onde seria inserido todo o código criado para uma única simulação em questão. Sendo assim foi criado o projeto "CARLA Ego Vehicle – Simulation to Excel".

Este projeto tem como objetivos o controlo manual de um veículo através do teclado, visualização de uma caixa delineadora sobre todos os outros veículos na visão do ator principal, identificação do Identity (ID) do veículo na visão do ator principal, cálculo da distância entre os outros veículos e velocidade dos veículos. No final da execução da simulação eram gerados ficheiros Excel, consoante o número de veículos que estavam na simulação, com todas as informações dos veículos: *Speed*, *Throttle*, *Brake*, *Steer*, *Reverse*, *Handbrake*, *Manual*, *Gear*, valores de posição x, y e z, *tick* e também gráficos com o trajeto dos veículos.

10.3.1. Pygame

Todos os exemplos que o CARLA proporciona e que foram criados para o projeto, são exemplos que necessitam da biblioteca Pygame, visto serem exemplos em que o utilizador pode conduzir o veículo ou visualizar uma simulação a ser executada, e para isso, o Pygame fornece uma reprodução em 2D do exemplo a ser executado, visto que o simulador CARLA apenas fornecer imagens em 3D.

Na Figura 28, é apresentada uma representação da execução de um exemplo do CARLA a ser executado pelo Pygame.

Figura 28 - Exemplo executado com o Pygame.

10.3.2. Spawn actor

No CARLA, qualquer veículo, objeto, sensor ou edifício são considerados atores, e todos os atores necessitam de ser criados. Para isso, o CARLA fornece uma função denominada "spawn_actor()", que recebe dois parâmetros, o parâmetro *blueprint* e o parâmetro *transform*.

O parâmetro *blueprint* é o que vai definir o ator que irá ser criado, por exemplo, se o *blueprint* for de um Tesla Model 3, irá ser criado um Tesla para o ator em questão e o parâmetro *transform* é a localização, x, y e z, juntamente com a rotação no veículo em relação ao mapa.

Na Figura 29, pode ser observada a criação de um ator que foi utilizada no projeto. Neste caso, para a criação de um segundo veículo foi utilizado o *blueprint* de um Tesla Model 3 e um *transform* que foi criado através da posição de um ator já existente com a condição de a posição ser sempre atrás do veículo principal.

```
blueprint = self.world.get_blueprint_library().find('vehicle.tesla.model3')
player2_location = self.player.get_transform().location
player2_direction = self.player.get_transform().get_forward_vector()
player2_rotation = self.player.get_transform().rotation
new_location = self.player2_location + 10*self.player2_direction
new_player2_location = carla.Location(self.new_location.x, self.new_location.y, self.new_location.z + 2)
player2 = self.world.spawn_actor(blueprint, carla.Transform(self.new_player2_location, self.player2_rotation))
```

Figura 29 - Spawn actor.

10.3.3. Distância entre os veículos

Para ser efetuado o cálculo da distância entre dois veículos, primeiramente é necessário obter as coordenas x e y e de seguida efetuar o cálculo matemático.

Para tal procedeu-se à criação de uma função que irá receber os valores x e y dos dois veículos e efetuar esse cálculo.

Na Figura 30, é apresentada a forma de como eram recolhidas as coordenadas dos veículos.

```
valx1 = world.player.get_location().x
valy1 = world.player.get_location().y
valx2 = vehicleidl.get_location().x
valy2 = vehicleidl.get_location().y
```

Figura 30 - Recolha das coordenadas x e y.

Na Figura 31, é apresentada a função que calcula a distância entre os veículos, através dos valores representados na figura anterior, Figura 30.

```
def distancia2d(valx1,valy1,valx2,valy2):
    """calc distance between cars"""
    valx = valx2 - valx1
    valy = valy2 - valy1
    return round(math.sqrt(math.pow(valx, 2) + math.pow(valy, 2)),1)
```

Figura 31 - Função do cálculo da distância.

No caso prático que estava a ser utilizado, essa distância era enviada para um *array* que guardava todas as distâncias de todos os veículos em relação ao ator principal. Na Figura 32, é mostrado como os valores da distância estavam a ser salvos.

```
distanciacheck[i] = distancia2d(valx1,valy1,valx2,valy2)
```

Figura 32 - Utilização dos valores de calculo da distância.

10.3.4. Velocidades dos veículos

Para efetuar o cálculo da velocidade de um veículo, era recolhido o parâmetro velocidade dos veículos, que contém a velocidade em x, velocidade em y e velocidade em z, e a partir da fórmula matemática que o CARLA fornecia, era possível converter essa velocidade para km/h.

Para tal, foi criada uma função que recebe esse parâmetro de velocidade e converte para km/h.

Na Figura 33, é mostrada a função utilizada.

```
def velocidade(vel):
    """Calc vel car"""
    kmh = int(3.6 * math.sqrt(vel.x**2 + vel.y**2 + vel.z**2))
    return kmh
```

Figura 33 - Função que converte a velocidade em km/h.

Na Figura 34, é apresentada a forma como era recolhida a velocidade dos veículos através da função “get_velocity()” que o simulador CARLA fornecia.

```
vel = str(velocidade(vehicleid1.get_velocity()))
```

Figura 34 - Recolha do parâmetro velocidade de um veículo.

No caso prático que estava a ser utilizado, este valor iria ser salvo para um dicionário, que depois irá ser utilizado para a criação do ficheiro Excel com todas as informações do veículo, que serão abordadas posteriormente.

10.3.5. Caixa nos veículos na visão do veículo principal

No campo de visão do veículo principal era possível visualizar uma caixa delimitadora de todos os outros veículos presentes na simulação.

Na Figura 35, é apresentado o código criado para a criação da caixa delimitadora.

```
debug.draw_box(carla.BoundingBox(actorV.get_transform().location,carla.Vector3D(0.5,0.5,2)),
                actorV.get_transform().rotation, 0.05, carla.Color(255,0,0,0),0.1)
```

Figura 35 - Criação da caixa delimitadora.

10.3.6. ID do veículo na visão do veículo principal e no simulador CARLA

No campo de visão do veículo principal e no próprio simulador do CARLA, era também possível visualizar o ID de todos os outros veículos na simulação em cima do veículo.

Na Figura 36, é apresentado o código criado para a criação do texto em cima do veículo.

```
debug.draw_string(actorV.get_transform().location, str(text),
                  False, carla.Color(255,0,0,0), -1)
display.blit(textsurface, (new_x,new_y))
```

Figura 36 - Texto ID em cima dos veículos.

Na Figura 37, é apresentado o resultado na visão do veículo principal, com a caixa e o ID.

Figura 37 - Visão do veículo principal.

Como o Pygame é uma biblioteca 2D e o CARLA é um simulador 3D foi impossível colocar o texto com ID na visão do Pygame na posição correta, ficando apenas na posição correta dentro do simulador.

10.3.7. Criação do ficheiro Excel

Para a criação do *software* final foi necessário salvar todos os dados que poderiam ser obtidos de cada veículo através deste *software*, para um ficheiro Excel separadamente, visto ser o *software* que corria a simulação no CARLA.

Para tal, foi criado um dicionário que guardava as informações *Speed*, *Throttle*, *Brake*, *Steer*, *Reverse*, *Handbrake*, *Manual*, *Gear*, posição *X*, *Y* e *Z* e *Tick*, de todos os veículos que estavam na simulação, sendo que normalmente eram sempre dois.

Na Figura 38, é apresentado o código que realiza o processo de recolha dos dados dos veículos a cada *tick*, sendo que como o *software* funciona num *loop* infinito foi criado o valor *tick* para ser possível distinguir o momento em que os dados foram recolhidos.

```
for actorID in VehicleID_list:
    for i in range(len(VehicleID_list)):
        try:
            if actorID.id == VehicleID_list[i].id:

                #Recolha de informação dos veiculos
                vel = str(velocidade(VehicleID_list[i].get_velocity()))
                x = VehicleID_list[i].get_location().x
                y = VehicleID_list[i].get_location().y
                z = VehicleID_list[i].get_location().z
                throttle = ControlCar[i].throttle
                braking = ControlCar[i].brake
                Steer = ControlCar[i].steer
                Reverse = ControlCar[i].reverse
                Handbrake = ControlCar[i].hand_brake
                Manual = ControlCar[i].manual_gear_shift
                Gear = ControlCar[i].gear

                #Enviar dados para o dicionario
                DicDados[str(actorID.id)]["Speed"].append(vel)
                DicDados[str(actorID.id)]["Throttle"].append(round(throttle,3))
                DicDados[str(actorID.id)]["Brake"].append(round(braking,3))
                DicDados[str(actorID.id)]["Steer"].append(Steer)
                DicDados[str(actorID.id)]["Reverse"].append(Reverse)
                DicDados[str(actorID.id)]["Handbrake"].append(Handbrake)
                DicDados[str(actorID.id)]["Manual"].append(Manual)
                DicDados[str(actorID.id)]["Gear"].append(Gear)
                DicDados[str(actorID.id)]["Position_X"].append(round(x,3))
                DicDados[str(actorID.id)]["Position_Y"].append(round(y,3))
                DicDados[str(actorID.id)]["Position_Z"].append(round(z,3))
                DicDados[str(actorID.id)]["Tick"].append(tick)
        except:
            print("VehicleID_list not ready")
        if deixar <= 1:
            time.sleep(0.2)
tick += 1
```

Figura 38 – Criação do dicionário com dados dos veículos.

O código fornece ainda uma verificação da existência dos atores, visto que as simulações levam algum tempo até serem executadas completamente, se não existisse essa verificação, poderiam ser criados dados falsos ou até mesmo levar à paragem do *software*.

Posteriormente, o dicionário de dados de cada veículo será enviado para um novo Excel. Na Figura 39, é apresentado código que procede à criação do ficheiro Excel.

```
for i in DicDados:
    df = pd.DataFrame(DicDados[i])
    df.to_excel(f"EXCEL/DadosCarro{i}.xlsx")
```

Figura 39 - Criação dos ficheiros Excel.

Na Figura 40, é mostrado o resultado no Excel dos dados obtidos dos veículos, sendo que a quantidade de valores obtidos depende do tempo de execução do *software*.

Speed	Throttle	Brake	Steer	Reverse	Handbrake	Manual	Gear	Position_X	Position_Y	Position_Z	Tick
0	0	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	1
0	0	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	2
0	0	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	3
0	0	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	4
0	0	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	5
0	0,01	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	6
0	0,01	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	7
0	0,01	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	8
0	0,04	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	9
0	0,04	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	10
0	0,04	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	11
0	0,04	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	12
0	0,08	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	13
0	0,08	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	14
0	0,08	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,001	15
0	0,12	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	16
0	0,12	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	17
0	0,12	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	18
0	0,12	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	19
0	0,15	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	20
0	0,15	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	21
0	0,15	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	22
0	0,18	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	23
0	0,18	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	24
0	0,18	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	25
0	0,18	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	26
0	0,22	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	27
0	0,22	0	0	FALSE	FALSE	FALSE	0	4,926	40,579	0,002	28

Figura 40 - Dados em Excel.

10.4. CARLA Ego Vehicle - 2D Path's

O *software* final denominado por "CARLA Ego Vehicle – 2D Path's" consiste na leitura dos ficheiros Excel criados pelo *software* anterior, para ser possível a visualização dos trajetos efetuados pelos veículos no decorrer da simulação, visualização de todas as informações presentes no ficheiro Excel, *Speed*, *Brake*, *Gear*, *Steer* e *Throttle*, dependendo do *tick* em que nos encontramos, e percorrer o valor do *tick* para podermos visualizar os dados de ambos os veículos no mesmo momento.

Este *software* foi criado com o propósito de qualquer pessoa conseguir correr uma simulação do CARLA, visto ser um simulador bastante pesado e ser necessário uma máquina razoável para poder correr as simulações, mostrando assim um exemplo do que seria visível no simulador.

A sua estrutura encontra-se dividida em 3 ficheiros: Ficheiro principal, Ficheiro com todas as funções e Ficheiro com a leitura inicial do ZIP. O ficheiro principal contém a criação de toda a estrutura do *software*, desde design do *software* até ao tratamento dos eventos de pressionar uma tecla; o ficheiro das funções continha todas as funções a serem utilizadas pelo ficheiro principal; e o ficheiro que fazia a leitura e validação do ZIP, que continha os ficheiros Excel a serem utilizados.

10.4.1. Escolher ficheiro ZIP

Normalmente, na execução do *software* anterior eram sempre utilizados dois veículos, gerando assim sempre dois ficheiros Excel. Para a simples utilização deste *software*, foi delineado que seria mais simples se o utilizador carregasse logo um ficheiro ZIP e assim teria os dois carros que estiveram na mesma simulação. Assim, sempre que este *software* era executado, era aberto o file explorer na pasta que continha todos os ZIP's gerados.

10.4.2. Modo Normal

O modo normal do *software* era a primeira janela apresentada ao utilizador, podendo visualizar o trajeto dos dois veículos, cada um em metade da tela e onde também podia optar por mudar para o modo de edição através da tecla “seta para cima”.

Na Figura 41, é apresentada a representação da primeira janela do software.

Figura 41 - Janela Modo normal.

Neste modo era também possível, utilizando o rato, clicar em dois pontos de um trajeto e era devolvida a distância em metros entre eles.

Na Figura 42, é mostrada a representação da utilização da distância entre dois pontos.

Figura 42 - Distância entre dois pontos.

10.4.3. Modo de edição

No modo de edição o utilizador conseguia controlar a posição dos veículos ao longo do valor do *tick* e assim, visualizar em cada *tick* os respetivos valores de *Speed, Brake, Throttle, Steer* e *Gear* de cada veículo.

Para percorrer os diferentes valores do *tick*, o utilizador utilizava as teclas "seta para a direita" e "seta para a esquerda" para avançar e recuar o valor do *tick*.

Era também possível alterar o multiplicador do *tick*, para assim poder ser mais rápido percorrer o valor do *tick*, sendo que para isso utilizar a tecla "1" para o multiplicador ser de 1 vez, "2" para o multiplicador ser de 2 vezes, e "4" para o multiplicador ser de 4 vezes.

Neste modo era também possível, através da tecla "P" ativar o reprodutor automático, isto é, o *software* iria correr todos os valores do *tick* automaticamente, onde também era possível alterar a velocidade de reprodução.

Na Figura 43, é apresentada a representação do modo de edição.

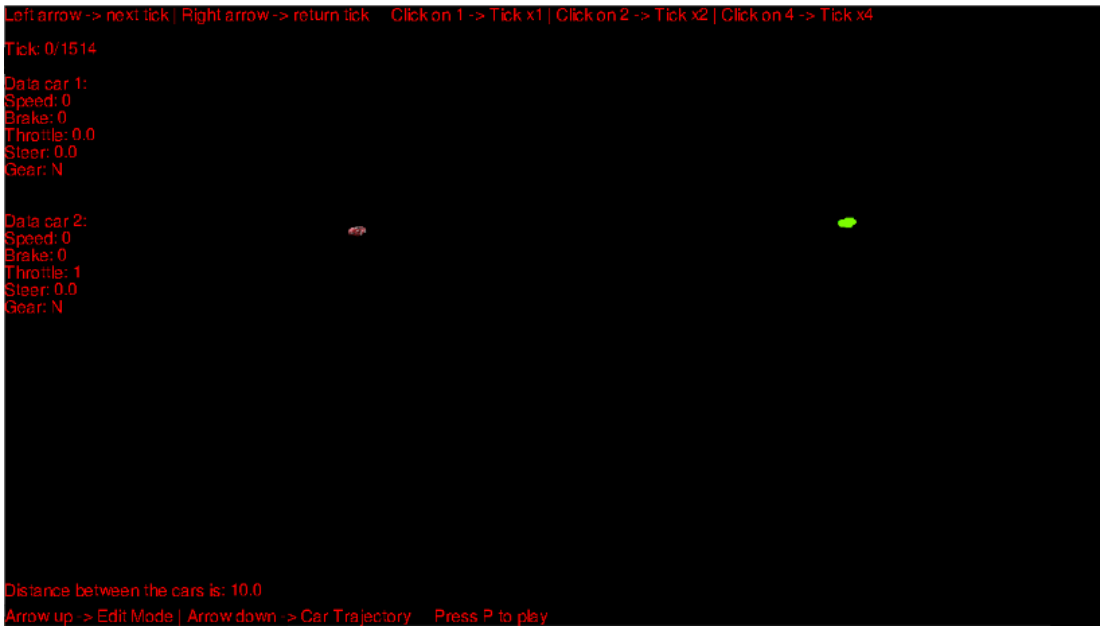

Figura 43 – Janela Modo edição.

10.4.4. Implementação futura

Com o término do estágio não foi possível melhorar o *software*, sendo que o objetivo seria a implementação de uma imagem aérea do simulador e assim os trajetos dos veículos serem vistos no mapa.

Na Figura 44, é mostrado um exemplo de como seria a vista aérea do mapa, sendo que o objetivo era ter essa visão aérea por trás dos trajetos do *software*.

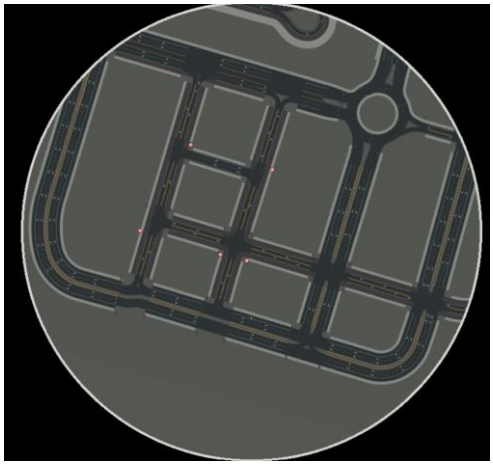

Figura 44 - Visão aérea mapa CARLA.

10.5. Unit Tests

Ao longo do decorrer do projeto foram também criados testes unitários para a validação dos ficheiros Excel a serem carregados pelo *software* “CARLA Ego Vehicle - 2D Path's”.

No caso prático, eram carregados dois ficheiros Excel e posteriormente era verificado se os ficheiros continham informação.

Na Figura 45, é apresentado um exemplo que foi criado para essa validação.

```
def setUp(self) -> None:
        self.data1input, self.data2input = verifyfiles(['filestotest/DadosCarro86_2022-07-15_09:41:32.xlsx',
                                                        'filestotest/DadosCarro87_2022-07-15_09:41:32.xlsx'])

    def testdic1(self):
        """Test"""
        if len(self.data1input) == 0:
            resultdic = False
        else:
            resultdic = True
        self.assertEqual(resultdic, True)

    def testdic2(self):
        """Test"""
        if len(self.data2input) == 0:
            resultdic = False
        else:
            resultdic = True
        self.assertEqual(resultdic, True)
```

Figura 45 - Unit tests.

Capítulo VI

Conclusão

Com este relatório é pretendido mostrar a todas as pessoas que não puderam acompanhar o meu percurso como estagiário na empresa Capgemini Engineering, a empresa e em que área de negócio se encontra, todas as ferramentas e as linguagens de programação utilizadas, o objetivo inicial do estágio e todo o trabalho realizado ao longo do mesmo.

Este relatório exigiu um grande esforço de forma a retratar todo o trabalho realizado ao longo do estágio, para que qualquer pessoa que nunca tenha estado perante as tecnologias utilizadas possa compreender.

O objetivo do estágio foi o desenvolvimento de simulações de cenários V2X num simulador denominado CARLA *Simulator*. De forma a atingir esse objetivo, fui inserido na iniciativa/projeto da Capgemini Engineering denominada "V2X initiative at the Embedded and Software Critical Systems unit".

Tendo em conta este objetivo, durante o decorrer do estágio, através do simulador CARLA e da linguagem de programação Python, foram desenvolvidas simulações de cenários entre dois veículos. Foi também desenvolvida uma aplicação que executa a biblioteca Pygame, onde foi possível escolher o trajeto de dois carros e assim comparar os valores gerados no simulador pelos mesmos. Por fim foi criada diversa documentação dos *softwares* criados.

Todos os tópicos referidos anteriormente foram executados e desenvolvidos da forma pretendida.

Foi um relatório que me deu gosto realizar visto ser acerca do que eu gosto mais de fazer e ser o último objetivo a ser cumprido para o término do curso.

Este estágio foi mais uma nova experiência que acrescentou ainda mais experiência à minha carreira e formou ainda melhor que é na área de desenvolvimento de *software* que quero seguir a minha carreira profissional.

Como perspetiva futura este projeto podia ter continuação através da melhoria do *software* “CARLA Ego Vehicle - 2D Path's”, com a implementação de uma imagem aérea do simulador e assim os trajetos dos veículos serem visto no mapa.

Num futuro próximo pretendo continuar a utilizar os conhecimentos adquiridos ao longo do estágio e também continuar a progredir a minha carreira através de estudos, nomeadamente uma licenciatura em Engenharia Informática.

Webgrafia

https://capgemini-engineering.com/pt/pt-pt/ - 14 de setembro 2022

https://capgemini-engineering.com/nl/en/about-us/overview/ - 14 de setembro 2022

https://www.microsoft.com/pt-pt/microsoft-teams/group-chat-software/ - 14 de setembro 2022

https://code.visualstudio.com/ - 14 de setembro 2022

https://blog.betrybe.com/git/ - 14 de setembro 2022

https://git-scm.com/ - 14 de setembro 2022

https://www.zup.com.br/blog/git-github-e-gitlab - 14 de setembro 2022

https://about.gitlab.com/ - 14 de setembro 2022

https://github.com/ - 14 de setembro 2022

https://www.pygame.org/news - 14 de setembro 2022

https://www.pluralsight.com/ - 14 de setembro 2022

https://portswigger.net/burp - 14 de setembro 2022

https://acervolima.com/o-que-e-burp-suite/ - 14 de setembro 2022

https://ubuntu.com/ - 14 de setembro 2022

https://www.python.org/ - 14 de setembro 2022

https://cplusplus.com/ - 14 de setembro 2022

https://www.automacaoindustrial.info/mqtt/ - 15 de setembro 2022

https://refactoring.guru/ - 15 de setembro 2022

https://pypi.org/project/pyflakes/ - 15 de setembro 2022

https://pypi.org/project/pylint/ - 15 de setembro 2022

https://medium.com/desenvolvendo-com-paixao/ - 16 de setembro 2022

https://carla.org/ - 16 de setembro 2022

https://www.pygame.org/docs/ref/event.html - 18 de setembro 2022

https://github.com/FranciscoG001/CARLA-2DCarsPath - 18 de setembro 2022

https://github.com/FranciscoG001/CARLA-EgoVehicle-Simulation2Xlsx – 18 de setembro 2022

https://canaltech.com.br/produtos/o-que-e-thread/ - 18 de setembro 2022

https://www.devmedia.com.br/programacao-com-threads/6152 - 18 de setembro 2022

https://www.hostgator.com.br/blog/metodo-de-callback-o-que-e/ - 18 de setembro 2022