



Instituto Politécnico de Tomar

# **Projeto Final DriveByMind**

2013 / 2014

**13683 Jorge Martins**

**13686 Rafael Costa**



# **Licenciatura em Engenharia Informática**

## **DriveByMind**

**Controlo de jogos através de ondas cerebrais**

**13683 Jorge Martins**

**13686 Rafael Costa**

**Outubro 2014**

**Orientadores:**

**António Manso**

**Gabriel Pires**



«A todos os que primam pelo desenvolvimento tecnológico e que vêm na inovação um horizonte inesgotável»

Jorge Martins, Rafael Costa

# Agradecimentos

Este trabalho foi desenvolvido no âmbito do projecto AMS-HMI2012 – *Assited Mobility by Shared-Control and Advanced Human-Machine Interfaces*, com referência RECI/EEI-AUT/0181/2012, ISR/UC/IPT/APCC, 2013-2015, financiado pela Fundação para Ciência e Tecnologia e programa COMPETE.

Ainda assim, este projeto concretizou-se graças à grande ajuda dos nossos orientadores António Manso e Gabriel Pires, que incansáveis, se mostraram sempre disponíveis para nos ouvir e/ou aconselhar.

Mais do que um projeto final de curso, este também é o resultado de um trabalho em equipa, em que nós, alunos, e respetivos orientadores nos entregámos e alcançámos assim este produto final.

A estes, e aos demais que indiretamente também contribuíram para a concretização do nosso projeto, um forte e sentido obrigado.

.

## Resumo

O desenvolvimento de jogos de vídeo com novas formas de interação teve um crescimento exponencial nos últimos tempos, assim como, um grande impacto na sociedade atual.

É proposto neste projeto o desenvolvimento de um jogo cujo controlo poder ser feito através de atividade cerebral recorrendo a sinais eletroencefalográficos (EEG). Trata-se por isso de um projeto inovador que tem como grande desafio torná-lo controlável com sinais EEG evocados por estímulos visuais.

Esta nossa proposta conjuga dois elementos: o Unity, uma framework para desenvolvimento de jogos e análise de estímulos visuais a frequências constantes que evocam padrões EEG designados por SSVEP (steady-state visual evoked potentials).

O Unity é um motor gráfico para desenvolvimento de jogos em 2D ou 3D, com possibilidade de exportação para diversas plataformas, tais como *Android*, *iOS*, *Windows Phone*, entre outras. A qualidade gráfica é elevada e estão disponíveis uma grande variedade de recursos, (objetos gráficos, tutoriais, fóruns, etc.) que tornam o desenvolvimento de jogos um processo mais leve e intuitivo.

No campo do entretenimento está menos explorado estando a surgir algumas novas apostas nesta área, como é o caso do produto desenvolvido pela NeuroSky, o MindWave.

Numa primeira fase, foi utilizado o dispositivo de aquisição de biossinais *MindWave* da empresa Neurosky, a qual possui um único elétrodo seco. Foi desenvolvida a interface de comunicação e um jogo 2D para comandar a posição de um carro através de ritmos EEG associados a níveis de atenção. Tendo-se verificado que o sistema de aquisição *MindWave* não fornecia qualidade EEG necessário e desejada, este sistema foi abandonado em detrimento de um sistema de aquisição de sinais certificado, nomeadamente o sistema gUSBamp da empresa gtec.

Na segunda fase do projeto foi desenvolvido um jogo num cenário 3D, um helicóptero a sobrevoar uma cidade, e um carro a percorrer as ruas de uma cidade, em que o controlo é com base em estímulos visuais que evocam os padrões SSVEP.

**Palavras Chave:** Unity, jogos de vídeo, 2D, 3D, eletroencefalografia, SSVEP, modelos 3D.

## Glossário

Sigla	Descrição
2D	Objetos com duas dimensões (Largura e comprimento).
3D	Objetos com três dimensões (Largura, comprimento e profundidade).
Android	Sistema operativo móvel desenvolvido pela Google Inc.
BCI	Brain-Computer Interface – Via de comunicação direta entre o cérebro e um dispositivo externo.
Blender	Software para desenvolvimento de modelos 3D e motor de jogo.
C#	Linguagem de programação orientada a objetos, desenvolvida pela Microsoft.
CEO	Director executivo (Chief Executive Officer)
CityEngine	Software para desenvolvimento de modelos 3D.
EEG	Eletroencefalografia – Estudo do registo gráfico das correntes elétricas desenvolvidas no encéfalo.
FPS	Frames Per Second – Frames por segundo – Numero de imagens que são apresentadas, por segundo, na visualização de um vídeo
GameObject	Qualquer objeto pertencente ao projeto
g.USBAmp	Amplificador de sinais biológicos. Dispositivo EEG
iOS	Sistema operativo móvel desenvolvido pela Apple Inc.
Java	Linguagem de programação orientada a objetos, desenvolvida pela Oracle Corporation.

JavaScript	Linguagem de programação implementada, originalmente, para utilização em Web Browsers.
JVM	Java Virtual Machine – programa que carrega e executa os aplicativos Java.
Localhost	Comunicação entre o mesmo dispositivo, como se este fosse um dispositivo remoto.
MindWave	Dispositivo EEG.
Open-Source	Código Aberto – Software de livre utilização, cuja licença não é cobrada, e o respetivo código fonte é disponibilizado pelo autor.
Rendering	Processo de gerar uma imagem a partir de um modelo 2D ou 3D.
SketchUp	Software para desenvolvimento de modelos 3D.
Socket	Tecnologia de sistemas distribuídos que permite que uma aplicação comunique com outra
SSVEP	<i>Steady State Visually Evoked Potential</i> - Sinais de resposta a estímulos visuais a determinadas frequências.
Stream	Fonte de entrada de dados, ou destino para escrita de dados
Strong typing	Tipagem segura. Característica de uma linguagem de programação que impede a conversão automática de tipos quando haja perda de informação.
TCP/IP	Conjunto de protocolos de comunicação entre dispositivos numa rede.
TCP Listener	Escuta as conexões de clientes na rede TCP.
ThinkGear	Programa para encaminhamento de dados entre MindWave e outros dispositivos.
Thread	Sequência de instruções que são executadas de forma assíncrona

numa linha de execução paralela.

# Índice

<b>1</b>	<b><i>Introdução</i></b>	<b>1</b>
<b>1.1</b>	<b>Visão global do projeto</b>	<b>2</b>
<b>1.2</b>	<b>Motivação</b>	<b>3</b>
<b>1.3</b>	<b>Tecnologias utilizadas</b>	<b>3</b>
<b>1.4</b>	<b>Contributos do projeto</b>	<b>4</b>
<b>1.5</b>	<b>Organização do relatório</b>	<b>5</b>
<b>2</b>	<b><i>Enquadramento do Projeto</i></b>	<b>7</b>
<b>2.1</b>	<b>Estado da Arte</b>	<b>7</b>
<b>2.1.1</b>	Dispositivos de captação de sinais EEG	7
<b>2.1.2</b>	Jogos controlados por EEG	11
<b>2.1.3</b>	Jogos em Unity controlados por EEG	13
<b>2.2</b>	<b>Tecnologias utilizadas</b>	<b>15</b>
<b>2.2.1</b>	Unity	15
<b>2.2.2</b>	EEG	16
<b>2.2.3</b>	SSVEP	17
<b>2.2.4</b>	C#	17
<b>2.2.5</b>	Java	18
<b>2.2.6</b>	MindWave	19
<b>2.2.7</b>	ThinkGear	19
<b>2.2.8</b>	g.USBamp	19
<b>2.2.9</b>	CityEngine	20
<b>2.2.10</b>	SketchUp	20
<b>2.2.11</b>	Blender	21
<b>3</b>	<b><i>Especificação do sistema</i></b>	<b>23</b>
<b>3.1</b>	<b>Introdução ao jogo</b>	<b>23</b>
<b>3.2</b>	<b>Menu</b>	<b>24</b>

3.2.1	Jogar .....	25
3.2.2	Opções .....	28
3.2.3	Sair.....	31
<b>3.3</b>	<b>Níveis .....</b>	<b>31</b>
3.3.1	Carros 2D.....	32
3.3.2	Carros 3D.....	45
3.3.3	Helicóptero 3D .....	63
3.3.4	Estímulos .....	79
3.3.5	Setas direcionais .....	81
3.3.6	Animações .....	84
3.3.7	Comunicação TCP/IP entre DriveByMind e MatLab.....	87
3.3.8	Introdução ao Jogo.....	90
<b>4</b>	<b>Testes e resultados experimentais.....</b>	<b>93</b>
<b>4.1</b>	<b>Setup experimental e análise offline dos SSVEP .....</b>	<b>93</b>
4.1.1	Setup experimental e procedimento de montagem .....	93
<b>4.2</b>	<b>Experiência 1 – análise SSVEP .....</b>	<b>94</b>
<b>4.3</b>	<b>Experiência 2 – análise offline.....</b>	<b>97</b>
<b>4.4</b>	<b>Experiencia 3 – Online.....</b>	<b>98</b>
4.4.1	Comando online do jogo.....	99
<b>5</b>	<b>Conclusões.....</b>	<b>103</b>
<b>5.1</b>	<b>Contribuição para a comunidade .....</b>	<b>103</b>
<b>5.2</b>	<b>Desenvolvimento Futuro.....</b>	<b>104</b>
<b>5.3</b>	<b>Apreciação final.....</b>	<b>104</b>
<b>Anexo 1</b>	<b>Enunciado do projeto.....</b>	<b>109</b>
<b>Anexo 2</b>	<b>Atas de reunião de projeto .....</b>	<b>110</b>
<b>Anexo 3</b>	<b>Tutorial Unity 3D .....</b>	<b>121</b>
<b>1</b>	<b>Tutorial Unity3D.....</b>	<b>121</b>
<b>1.1</b>	<b>Criação de um projeto em Unity.....</b>	<b>121</b>

<b>1.2</b>	<b>Componentes .....</b>	<b>122</b>
1.2.1	Física.....	122
1.2.2	Material.....	124
1.2.3	Câmeras .....	124
1.2.4	Luzes.....	125
1.2.5	GameObject.....	126
1.2.6	Prefabs .....	126
<b>1.3</b>	<b>Ambiente Unity .....</b>	<b>127</b>
1.3.1	Hierarchy .....	127
1.3.2	Console .....	128
1.3.3	Inspector .....	128
1.3.4	Project.....	128
1.3.5	Scene.....	131
1.3.6	Game.....	132
1.3.7	Play/Pause/Step .....	133
1.3.8	Barra de estado .....	133
<b>1.4</b>	<b>Criação de um cenário de jogo.....</b>	<b>133</b>
1.4.1	Terrain .....	133
1.4.2	Sky Box .....	134
1.4.3	Scripts .....	135
1.4.4	Fatores a ter em conta no desenvolvimento .....	138

# Índice de figuras

<i>Figura 1 – Setup e cenário de jogo .....</i>	1
<i>Figura 2 – Plataformas compatíveis com o DriveByMind .....</i>	3
<i>Figura 3 – MindWave da NeuroSky .....</i>	8
<i>Figura 4 – Emotiv EPOC .....</i>	9
<i>Figura 5 – Dispositivo BodyWave .....</i>	10
<i>Figura 6 – Dispositivo NIA Game Controller .....</i>	10
<i>Figura 7 – Jogo Mindflex .....</i>	11
<i>Figura 8 - Cenário do jogo controlado pelo BodyWave .....</i>	12
<i>Figura 9 - Jogo e respectivos elementos do Spacecraft .....</i>	12
<i>Figura 10 - Simulador cadeira de rodas .....</i>	13
<i>Figura 11 – Cenário do jogo com respectivos níveis de atenção .....</i>	14
<i>Figura 12 – Dispositivo MindSet.....</i>	14
<i>Figura 13 – Logotipo Unity.....</i>	16
<i>Figura 14 – Pessoa utilizando EEG .....</i>	17
<i>Figura 15 – Microsoft Visual C#.....</i>	18
<i>Figura 16 – Logotipo Java .....</i>	18
<i>Figura 17 – Dispositivo g.USBamp da g-Tec.....</i>	20
<i>Figura 18 – Logotipo CityEngine.....</i>	20
<i>Figura 19 – Logotipo SketchUp .....</i>	21
<i>Figura 20 – Logotipo Blender.....</i>	21
<i>Figura 21 – Logotipo do jogo DriveByMind .....</i>	23
<i>Figura 22 – Introdução do DriveByMind.....</i>	24
<i>Figura 23 – Menu inicial DriviByMind.....</i>	25
<i>Figura 24 – Menu jogar DriveByMind.....</i>	25
<i>Figura 25 – Menu Carros2D.....</i>	26
<i>Figura 26 – Menu Carros3D.....</i>	27
<i>Figura 27 - Menu Helicóptero.....</i>	28
<i>Figura 28 – Menu opções DriveByMind .....</i>	28

<i>Figura 29 – Menu definição manual frequências.....</i>	29
<i>Figura 30 – Menu escolha de estímulos .....</i>	29
<i>Figura 31 – Menu calibração dos estímulos .....</i>	30
<i>Figura 32 – Menu de comandos para comunicação Unity / MatLab .....</i>	31
<i>Figura 33 – Hierarquia do jogo Carros2D .....</i>	33
<i>Figura 34 – Componentes da pista de jogo.....</i>	34
<i>Figura 35 – Componentes do objecto carro 2D.....</i>	36
<i>Figura 36 – Componentes do objecto Camera.....</i>	36
<i>Figura 37 – Janela conexão MindWave .....</i>	39
<i>Figura 38 – Diagrama classes da conexão do MindWave .....</i>	42
<i>Figura 39 – Simulação de jogo (Obstáculo Barreira).....</i>	43
<i>Figura 40 – Simulação de jogo (Obstáculo Bidon).....</i>	43
<i>Figura 41 – Simulação de jogo (Obstáculo Buraco).....</i>	43
<i>Figura 42 – Simulação de jogo (Obstáculo Coelho).....</i>	44
<i>Figura 43 – Simulação de jogo (Obstáculo Pino).....</i>	44
<i>Figura 44 – Simulação de jogo (Vários obstáculos) .....</i>	44
<i>Figura 45 – Simulação de jogo (versão Android) .....</i>	44
<i>Figura 46 – Simulação do jogo versão MindWave .....</i>	45
<i>Figura 47 – Seta orientadora .....</i>	46
<i>Figura 48 – Terreno de construção do cenário de jogo.....</i>	47
<i>Figura 49 – Propriedades de um modelo 3D (prefab) .....</i>	48
<i>Figura 50 – Modelo 3D no cenário de jogo e suas propriedades .....</i>	49
<i>Figura 51 – Componentes do modelo 3D do táxi .....</i>	50
<i>Figura 52 – NavMesh jogo Taxi.....</i>	51
<i>Figura 53 – Destinos jogo Táxi .....</i>	52
<i>Figura 54 – Componentes do modelo 3D Character Controller.....</i>	53
<i>Figura 55 – Destinos NavMesh a volta do táxi para o passageiro .....</i>	53
<i>Figura 56 – Escolha de destinos através do teclado .....</i>	56
<i>Figura 57 - Procedimento da escolha de direção para a versão teclado.....</i>	56
<i>Figura 58 – Escolha de destino através de estímulos visuais.....</i>	57

<i>Figura 59 – Procedimento da escolha de direção para a versão com os estímulos.....</i>	57
<i>Figura 60 – Diagrama de classes que controlam movimento do táxi .....</i>	59
<i>Figura 61 – Menu de Pausa .....</i>	60
<i>Figura 62 – Inicio jogo carros3D .....</i>	61
<i>Figura 63 – Táxi apanha primeiro viajante .....</i>	61
<i>Figura 64 – Destino primeiro viajante.....</i>	61
<i>Figura 65 – Apanha terceiro viajante .....</i>	62
<i>Figura 66 – Estímulos para decisão da direcção.....</i>	62
<i>Figura 67 – Indicação do estímulo detectado .....</i>	63
<i>Figura 68 – Arquitetura original da cidade.....</i>	64
<i>Figura 69 – Arquitetura final da cidade.....</i>	65
<i>Figura 70 – Componentes do helicóptero .....</i>	65
<i>Figura 71 – NavMesh do jogo helicóptero.....</i>	66
<i>Figura 72 – Destinos Navmesh do jogo helicóptero .....</i>	66
<i>Figura 73 – Visualização das moedas do jogo.....</i>	67
<i>Figura 74 – Animação da moeda .....</i>	67
<i>Figura 75 – Anéis do nível II do jogo 3D do helicóptero .....</i>	68
<i>Figura 76 – Escolha dos destinos através de teclado.....</i>	70
<i>Figura 77 - Procedimento da escolha de direção para a versão teclado.....</i>	70
<i>Figura 78 – Escolha dos destinos através do g.USBBamp .....</i>	71
<i>Figura 79 - Procedimento da escolha de direção para a versão g.USBBamp .....</i>	71
<i>Figura 80 – Diagrama de classes que controlam o movimento do helicóptero .....</i>	72
<i>Figura 81 – Diagrama de classes que controlam o movimento do helicóptero no modo livre .....</i>	74
<i>Figura 82 – Fim do jogo do helicóptero .....</i>	75
<i>Figura 83 – Inicio jogo nível I Helicóptero.....</i>	76
<i>Figura 84 – Jogo helicóptero nível I.....</i>	77
<i>Figura 85 – Helicóptero apanha moeda .....</i>	77
<i>Figura 86 – Helicóptero apanha segunda moeda .....</i>	77
<i>Figura 87 – Jogo helicóptero nível II.....</i>	78
<i>Figura 88 – Preparação para entrada do helicóptero no anel .....</i>	78

<i>Figura 89 – Helicóptero entra no anel.....</i>	78
<i>Figura 90 – Estímulos .....</i>	80
<i>Figura 91 – Diagrama de classes que controlam os estímulos visuais.....</i>	81
<i>Figura 92 – Setas direcionais para a versão PC.....</i>	82
<i>Figura 93 – Setas direcionais para a versão móvel .....</i>	83
<i>Figura 94 – Jogo helicóptero nível 2, sem setas direcionais.....</i>	83
<i>Figura 95 – Disposição das setas direcionais para a versão móvel .....</i>	83
<i>Figura 96 - Animação passageiro no jogo Táxi3D .....</i>	84
<i>Figura 97 - Transição relativa à animação “Parado” e “Chamar_Carro” .....</i>	85
<i>Figura 98 – Visão da primeira câmara da introdução do jogo.....</i>	90
<i>Figura 99 – Segunda câmara da introdução, com a apresentação de estímulos .....</i>	91
<i>Figura 100 - Terceira câmara da introdução .....</i>	92
<i>Figura 101 – Quarta câmara da introdução .....</i>	92
<i>Figura 102 – Setup experimental .....</i>	93
<i>Figura 103 – Sistema internacional 10-20 estendido (posicionamento dos elétrodos).....</i>	94
<i>Figura 104 – Espectro de frequência para aproximadamente 50 segundos de estimulação a 6 Hz (azul) e a 9 Hz (vermelho) .....</i>	94
<i>Figura 105 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 6 Hz .....</i>	95
<i>Figura 106 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 9 Hz.....</i>	95
<i>Figura 107 – Espectro de frequência obtido para aproximadamente 50 segundos de estimulação a 7.5 Hz (vermelho) e 11 Hz (azul).....</i>	96
<i>Figura 108 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 7.5 Hz.....</i>	96
<i>Figura 109 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 11 Hz.....</i>	96
<i>Figura 110 – Teste do jogo DriveByMind utilizando o g.USBamp na experiência 2.....</i>	97
<i>Figura 111 – Espectros de frequência da resposta alternada entre estímulo esquerdo (6 Hz) e estímulo direito (9 Hz) .....</i>	98
<i>Figura 112 - Espectros de frequência da resposta alternada entre estímulo esquerdo (7.5 Hz) e estímulo direito (9 Hz).....</i>	98

<i>Figura 113 - Espectros de frequência da resposta alternada entre estímulo esquerdo (7.5 Hz), estímulo direito (9 Hz) e estímulo frente (11Hz).....</i>	99
<i>Figura 114 – Teste dos estímulos utilizando o g.USBamp na experiência 1.....</i>	100
<i>Figura 115 – Controlo jogo do helicóptero modo 1 .....</i>	100
<i>Figura 116 – Controlo jogo táxi.....</i>	100
<i>Figura 117 – Controlo jogo helicóptero modo 2.....</i>	101
<i>Figura 118 – Janela inicial para a criação de um projecto em Unity .....</i>	122
<i>Figura 119 – Janela das definições do componente Rigidbody .....</i>	123
<i>Figura 120 – Janela das definições da Box Collider .....</i>	123
<i>Figura 121 – Componente Character Controller(Esquerda) e respectiva janela de configurações (direita).....</i>	124
<i>Figura 122 – Janela das definições dos materiais dos objectos.....</i>	124
<i>Figura 123 – Câmera de jogo e respetiva janela de configurações.....</i>	125
<i>Figura 124 – Câmera principal (Main) do jogo.....</i>	125
<i>Figura 125 – Janela de configurações do componente Luz .....</i>	126
<i>Figura 126 - Janela do ambiente de desenvolvimento do Unity.....</i>	127
<i>Figura 127 – Hierarquia de componentes do cenário.....</i>	127
<i>Figura 128 – Consola Unity.....</i>	128
<i>Figura 129 – Transform Tools .....</i>	132
<i>Figura 130 – Ferramenta Gizmo.....</i>	132
<i>Figura 131 – Botões Play/Pause/Stepa .....</i>	133
<i>Figura 132 – Terreno do jogo e respetiva janela de definições .....</i>	134
<i>Figura 133 – Edição do terreno de jogo .....</i>	134
<i>Figura 134 – Janela de definições da componente Sky Box.....</i>	135
<i>Figura 135 – Exemplo de um script em C# para Unity.....</i>	136
<i>Figura 136 – Janela de definições das variáveis públicas referentes a um Script.....</i>	137
<i>Figura 137 – Definições das texturas de um objeto .....</i>	139

# Índice de códigos

<i>Código 1 – Movimento da estrada (pista) .....</i>	34
<i>Código 2 – Movimento do carro2D.....</i>	35
<i>Código 3 – Colocação aleatória de obstáculos no jogo.....</i>	37
<i>Código 4 – Mostrar cálculos no ecrã.....</i>	38
<i>Código 5 – Valores recebidos por parâmetro da classe NeuroServer .....</i>	39
<i>Código 6 – Classe Send_Attention .....</i>	40
<i>Código 7 – Ligação cliente TCP .....</i>	41
<i>Código 8 – Receber dados por TCP .....</i>	41
<i>Código 9 – Move carro através do valor recebido do MindWave.....</i>	41
<i>Código 10 – Definição de destinos NavMesh.....</i>	51
<i>Código 11 – Mostrar animações, conforme tempos e distâncias .....</i>	54
<i>Código 12 – Deteção de colisão e ativação dos destinos a percorrer.....</i>	54
<i>Código 13 – Funcionamento da tomada de decisão relativa ao destino do táxi .....</i>	55
<i>Código 14 – Definição seta orientadora, distâncias e mudança de estado de ocupação do táxi.....</i>	58
<i>Código 15 – Apanhar e largar um passageiro .....</i>	58
<i>Código 16 – Colocar o jogo em pausa .....</i>	60
<i>Código 17 - Funcionamento da tomada de decisão relativa ao destino do helicóptero.....</i>	69
<i>Código 18 - Escolha da direção através do teclado ou através de touch na versão para Android/iOS</i>	73
<i>Código 19 - Escolha da direção através da escolha visual, via comando TCP/IP enviado pelo MatLab .....</i>	73
<i>Código 20 - Colocar altitude do helicóptero correta em relação a altitude do anel .....</i>	73
<i>Código 21 – Script ApanharMoedas .....</i>	75
<i>Código 22 – Controlo estímulos .....</i>	80
<i>Código 23 – Verificação da plataforma em que o jogo está a ser executado .....</i>	82
<i>Código 24 – Script para controlar as animações.....</i>	86
<i>Código 25 – Criação do Socket, TCP Listener e Thread para escutar clientes .....</i>	88
<i>Código 26 – Função ListenForClients .....</i>	88
<i>Código 27 – Função HandleClientComm .....</i>	89

<i>Código 28 – Função SendMessageGetFreq .....</i>	90
<i>Código 29 – Troca entre estímulos na introdução .....</i>	91

# 1 Introdução

Este projeto foi desenvolvido no âmbito da unidade curricular de Projeto Final do curso de Licenciado em Engenharia Informática pelo Instituto Politécnico de Tomar.

O projeto surgiu no âmbito das atividades do laboratório VITA.upt [1], vida assistida por ambientes inteligentes, na continuação do projeto LiveBioFeedBack [6], para a integração das tecnologias Unity [2] no desenvolvimento de jogos 3D e utilização de estímulos visuais que geram padrões eletroencefalográficos (EEG) designados por SSVEP (Steady state visually evoked potential) [3,4]. A Figura 1 mostra o setup e cenário de um dos jogos implementados.

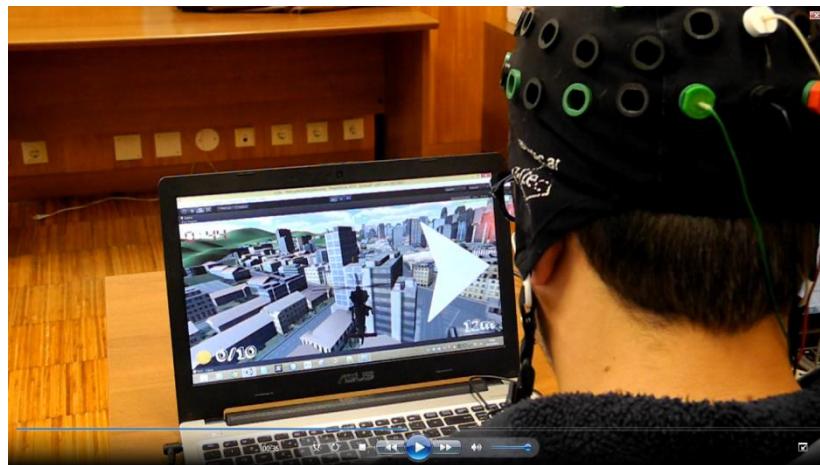


Figura 1 – Setup e cenário de jogo

Os jogos de vídeo têm aumentado exponencialmente a sua popularidade ao longo do tempo, causando uma influência significativa na sociedade atual.

Tradicionalmente jogos comerciais são criados por editoras e desenvolvem os seus próprios motores de jogo, nos quais despendem importantes recursos monetários e por consequência são exclusivos para o desenvolvimento dos seus produtos. Com o aparecimento de motores para o desenvolvimento de jogos gratuitos surgiram oportunidades para produtores independentes desenvolverem os seus próprios jogos. Estes últimos focam-se principalmente na inovação e um exemplo disso são os *Indie games* [5], ou jogos independentes.

Os jogos são normalmente controlados por dispositivos de *input* tradicionais, tais como teclado e o rato, ou por dispositivos próprios como por exemplo volantes ou joysticks. No entanto, têm aparecido novas formas de controlo, com o intuito de testar

novas formas de interação, como por exemplo a Wii[37] que deteta movimentos em três dimensões através do seu controlo remoto ou no caso do Kinect[38] que permite a interação com os jogos sem ser necessário o uso de um controlo remoto ou *joystick*. Estas tecnologias pretendem aumentar o nível de interatividade e imersividade dos jogos na sua vertente mais lúdica, ou para satisfazer necessidades especiais de utilizadores que por possuírem limitações motoras são incapazes de usar os dispositivos de controlo tradicionais de forma a usufruir dos recursos computacionais. Um bom exemplo de tecnologias que pretendem tornar os sistemas computacionais mais acessíveis são as interfaces cérebro-computador, um método de comunicação direto entre o cérebro e um dispositivo externo. Frequentemente, são empregadas com o objetivo de auxiliar, aumentar ou restaurar o sentido cognitivo humano ou funções sensoriais e motoras para controlo de sistemas computacionais.

O desenvolvimento do jogo em Unity com o controlo por EEG é o foco principal deste projeto.

## 1.1 Visão global do projeto

Este projeto pretende dar seguimento ao sistema LiveBioFeedback, desenvolvido no ano letivo 2012/2013 e apresentado como projeto final de curso da licenciatura em Engenharia Informática por alunos do Instituto Politécnico de Tomar.

O sistema anterior consistiu no desenvolvimento de uma plataforma de *software* capaz de permitir a visualização gráfica de informação EEG e o controlo de um jogo chamado DriveByMind. O DriveByMind assenta num cenário em 2D e tem como objetivo o controlo da posição de um carro, fazendo o mesmo desviar-se dos obstáculos existentes. Este, foi desenvolvido utilizando a linguagem de programação *Java* [6].

O presente projeto inicia com a implementação e reprodução do DriveByMind utilizando o motor de desenvolvimento de jogos *Unity*, em 2D.

Posteriormente, e o que nos distancia mais do projeto anterior, é o facto de desenvolvermos um novo jogo m ambiente 3D. Estes ambientes são mais ricos e atrativos visualmente e permitem o controlo do jogo no espaço tridimensional.

Outro fator distintivo passa pelo mecanismo neuronal utilizado para controlar o jogo. Enquanto no DriveByMind, o jogo era controlado usando um sinal supostamente

associado a ritmos de atenção fornecido diretamente pelo MindWave, no nosso caso, o sinal de controlo será a resposta cerebral a estímulos visuais a piscar a uma frequência constante. Essa resposta cerebral designada por SSVEP, precisou que fossem criados estímulos visuais embebidos no jogo.

Com o intuito de alcançar um número elevado e diversificados tipos de utilizadores, este jogo foi desenvolvido de forma a garantir a compatibilidade com os sistemas operativos *Windows* e *Android*.e *Mac OsX* (Figura 2).



*Figura 2 – Plataformas compatíveis com o DriveByMind*

## 1.2 Motivação

Este projeto foi encarado desde o início como um verdadeiro desafio, pelos mais diversos motivos. O facto de, explorar tecnologias nunca estudadas anteriormente por nós, tornou-se o desafio principal. Arriscar sair da nossa zona de conforto, e com isso ter a possibilidade de adquirir conhecimentos em tecnologias emergentes que serão com certeza uma mais-valia no nosso futuro profissional.

Ao pegarmos num projeto cem por cento exequível e bem-sucedido, com o objetivo de o melhorar e fazer com que este seja ainda mais evoluído, foi um grande impulsionador da construção deste nosso projeto.

## 1.3 Tecnologias utilizadas

Para a realização deste projeto recorremos a um conjunto diversificado de tecnologias, as quais serão sucintamente descritas de seguida.

- EEG – Eletroencefalografia é o estudo do registo das correntes elétricas produzidas no encéfalo.
- C# – Linguagem de programação orientada a objetos, desenvolvida pela *Microsoft* pertencendo à plataforma .NET.

- Java – Linguagem de programação orientada a objetos, que é concorrente, e baseada em classes.
- Unity – Unity 3D é um motor para desenvolvimento de jogos, em 2D ou 3D. Com um IDE criado pela *Unity Technologies*, esta plataforma assemelha-se a outras como, *Blender* ou *Torque Game Engine* no que diz respeito à sua interface gráfica. Inicialmente este motor cresceu como suporte para Mac OS X e atualmente expandiu-se para multiplataformas, como por exemplo *Windows*, *Android*, *Windows Phone 8*, *iPhone* ou *iPad*, entre outras. Este motor permite a programação em C# ou JavaScript.
- MindWave – Aparelho de neurofeedback, produzido pela NeuroSky (produtor de interfaces cérebro-computador), planeado para medir e projetar ondas cerebrais através de um sensor. Esta tecnologia pode captar sinais como atenção, meditação ou piscar de olhos.
- g.USBamp – Amplificador de sinais biológicos de alta performance e alta precisão[7].
- CityEngine – *Software* para desenvolvimento de modelos tridimensionais [8].
- SketchUp – *Software* para desenvolvimento de modelos tridimensionais [10].
- Blender – Software de computação gráfica em 3D profissional [12].
- ThinkGear Connector – Processo executado em *Background* responsável por encaminhar os dados do dispositivo EEG para outro dispositivo (Ex: computador) [14].
- SSVEP – *Steady State Visually Evoked Potential*, são sinais cerebrais a uma resposta natural a estímulos visuais.

## 1.4 Contributos do projeto

Este projeto tem como principal objetivo a exploração e criação de jogos em 3D utilizando a plataforma Unity3D.

Complementando-se com a integração de uma tecnologia também em forte crescimento de utilização, as interfaces baseadas em EEG.

A criação de todo o projeto desde a sua raiz, a exploração de toda a plataforma para o desenvolvimento do mesmo, abre portas a futuros projetos que sejam criados em Unity. Pois, é também um dos grandes objetivos deste projeto, adquirir experiência e conhecimentos sobre esta nova e evoluída plataforma para desenvolvimento de jogos, de modo a que possíveis desenvolvedores não sintam as dificuldades por nós encontradas, assim como, adotarem o melhor caminho a seguir.

## 1.5 Organização do relatório

O relatório encontra-se estruturado segundo os seguintes capítulos:

- **Introdução** – Breve descrição introdutória do projeto realizado, expondo uma sumária apresentação do mesmo, assim como, a motivação para a sua concretização, tecnologias utilizadas para a conceção e os contributos obtidos com a sua criação.
- **Enquadramento do projeto** – Reconhecimento ou investigação de soluções, já existentes, idênticas ao projeto em questão e descrição das tecnologias utilizadas na conceção do projeto e fundamentação.
- **Especificação do sistema** – Pormenorização dos diversos itens constituintes do jogo e consequente descrição técnica utilizada para a conceção do mesmo.
- **Testes experimentais** – Resultados dos testes experimentais efetuados e comprovação do correto funcionamento do jogo.
- **Conclusões** – Análise e apreciação final do trabalho realizado e respetivos objetivos alcançados.



## 2 Enquadramento do Projeto

O presente projeto consiste no desenvolvimento de um jogo controlado por ondas cerebrais captadas através de uma tecnologia EEG. Pretende-se que este jogo seja desenvolvido utilizando a plataforma *Unity 3D*.

Para o desenvolvimento deste jogo, foi necessário uma primeira análise, a compreensão e um estudo pormenorizado da plataforma onde irá assentar todo o nosso trabalho. Para tal, recorremos ao projeto LiveBioFeedback. Implementámos, assim, o jogo em 2D na plataforma *Unity*, e efetuamos a respetiva exportação para o sistema *Android*. Este mesmo jogo é controlado através do sistema *MindWave*. Temos então adquiridas as bases necessárias para iniciar todo o nosso projeto na plataforma *Unity 3D*.

### 2.1 Estado da Arte

O projeto integra um conjunto de tecnologias diversificadas o que dificulta um pouco a realização do estado da arte. Assim, procuráramos sistemas que se relacionem com algumas partes do nosso projeto.

#### 2.1.1 Dispositivos de captação de sinais EEG

Têm aparecido nos últimos anos sistemas de aquisição de sinal EEG de baixo custo com o objetivo de chegar mais facilmente ao consumidor comum, e incentivar o consumo de massas. Estes sistemas anunciam muitas vezes funcionalidades e características que são falaciosas. A qualidade dos biossinais, em particular, dos sinais EEG, é quase sempre abaixo do limiar de discriminação. Por enquanto, a qualidade destes sistemas é muito inferior à obtida por sistemas de uso clínico e/ou para investigação. De seguida apresentam-se alguns sistemas de baixo custo tais como o Mindwave, o Emotiv, Mindflex, Bodywave, os quais vêm muito orientados para as aplicações de jogos.

##### 2.1.1.1 MindWave

A parceria entre a NeuroSky com as principais universidades do mundo, como Stanford, Carnegie Mellon, Washington, Wollongon, Trinity College entre outras, deu como fruto o NeuroSky MindWave (Figura 3), sendo este o primeiro produto disponível no mercado consumidor. Utiliza o mesmo bio-sensor que o Mattel

MindFlex, o Star Wars Force Trainer, e a ferramenta de pesquisa da NeuroSky MindSet. Trata-se de dispositivos de baixo custo com um único elétrodo a seco tipicamente localizado na testa.

Existem atualmente mais de 40 jogos e aplicativos educacionais disponíveis para download na AppStore da NeuroSky[16] e o pacote MindWave já traz incluídos alguns, como SpeedMath, Meditation Journal, BlinkZone entre outros [6].

Os sinais cerebrais são transmitidos para o computador via *Wireless*.



*Figura 3 – MindWave da NeuroSky*

#### 2.1.1.2 Emotiv

Semelhante ao MindWave, este dispositivo é também portátil e transmite os dados para o computador do mesmo modo, mas no entanto o Emotiv EPOC [19] (Figura 4) possui 16 sensores [18].

- EPOC

Mais vocacionado para a interação do utilizador com as aplicações, este módulo prende-se essencialmente com as questões de jogos. Assim, o utilizador poderá jogar um jogo sem qualquer contacto físico com ratos, teclados ou joysticks.

O EPOC faz leituras de padrões de sinais e expressões faciais de modo a perceber qua a intenção do utilizador. No campo dos pensamentos cognitivos a empresa anuncia que o dispositivo reconhece 13 tipos de movimentos incutidos pelo utilizador, 6 direções, 6 rotações e o desejo de apagar. Pode

ainda reconhecer emoções tais como excitação ou meditação, expressões faciais como rir ou posição ocular e rotações angulares da cabeça [6].

- EEG

Este módulo é mais vocacionado para leitura e registo de sinais eletroencefalográficos, prende-se mais com questões de sensações e emoções do que interações. À semelhança do MindWave, engloba uma série de aplicações como neuro terapia, biofeedback ou interface cérebro-computador. Possui todas as vantagens do EPOC referidas anteriormente, com o benefício de permitir o acesso aos valores recebidos ou transmitidos pelo dispositivo [6].



*Figura 4 – Emotiv EPOC*

#### 2.1.1.3 BodyWave

Esta tecnologia desenvolvida pelo CEO da Freer Logic, é anunciada como sendo capaz de ler sinais cerebrais através de um monitor de ondas cerebrais inovador amarrados ao braço ou à perna. (Figura 5)

Esta tecnologia é assim anunciada como um sistema que permite substituir o uso das tradicionais toucas EEG [23].

Estes sinais são transmitidos para o computador via *Wireless*.



*Figura 5 – Dispositivo BodyWave*

#### 2.1.1.4 NIA – Neural Impulse Actuator

O NIA Game Controller (Figura 6) é um dispositivo BCI, desenvolvido pela OCZ Techonlogy [24]. Este produto pertence agora a outra empresa, a BCInet [25].

Este dispositivo é atualmente utilizado para controlo de jogos. O NIA traduz gestos, intensões ou movimento dos olhos dos utilizadores em instruções de controlo básicas (combinações de teclas e cliques de rato).



*Figura 6 – Dispositivo NIA Game Controller*

## 2.1.2 Jogos controlados por EEG

### 2.1.2.1 Mindflex

Jogo ou brinquedo (Figura 7), produzido pela Mattel [20]. Transportando assim para os brinquedos a tecnologia da eletroencefalografia dando um novo fôlego a este mercado que tem sido esmagado pela evolução dos jogos de vídeo.

Utilizando o chip da NeuroSky, este brinquedo anuncia o controlo com a mente utilizando as ondas cerebrais como principal meio de interação. Usando os pensamentos para nos aproximarmos da telecinesia, a capacidade de controlar fisicamente um objeto com a força da mente, uma pequena bola sobe para o ar através de uma coluna de ar. Quando o utilizador se concentra com força suficiente para levantar a bola, tenta fazê-la passar entre diversos tipos de obstáculos como, aros, rampas ou portas. Quanto mais intensamente o utilizador pensa na bola, mais alto ela sobe. Quando passa a um estado mais relaxado a bola desce [6].

Este sistema tem no entanto sido ridicularizado pela comunidade científica por apresentar características de forma falaciosa [26].



*Figura 7 – Jogo Mindflex*

### 2.1.2.2 BodyWave Golfinho

Em estado de teste do dispositivo BodyWave (Figura 8), era incluído um jogo para treino da atenção. O jogo consistia no controlo de um golfinho dentro de água, quando

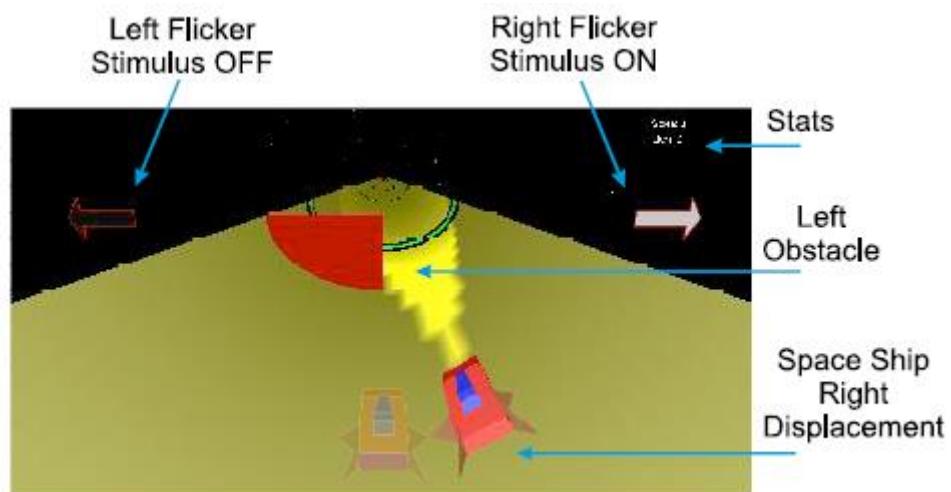
o utilizador foca a sua atenção no animal o dispositivo faz com que ele desça para o fundo do oceano. Se o utilizador se distrair, então o golfinho virá ao cimo da água.



*Figura 8 - Cenário do jogo controlado pelo BodyWave*

#### 2.1.2.3 Jogo spacecraft

O jogo spacecraft (Figura 9), é um jogo de interface cérebro-computador (BCI), onde o objetivo é controlar uma nave espacial num determinado espaço evitando colidir com os obstáculos [3]. Para a obtenção de sinais cerebrais para controlo da nave é utilizado SSVEP. Estes estímulos entre 3 e os 5 Hz são utilizados para mover a nave para a esquerda ou para a direita. O sistema de aquisição de bioassinais EEG é o g.USBamo, um sistema certificado de uso clínico. O jogo foi desenvolvido para possíveis aplicações de neuroterapia direcionado para grupos de população com défice de atenção, Neurofibromatose e autismo.



*Figura 9 - Jogo e respectivos elementos do Spacecraft*

## 2.1.3 Jogos em Unity controlados por EEG

### 2.1.3.1 Simulador de uma cadeira de rodas

Consiste no desenvolvimento de um simulador controlado por BCI, para treinar a utilização deste sistema implementado numa cadeira de rodas [22].

Neste exemplo foi construído em Unity3D um cenário que contém uma cadeira de rodas que se desloca dentro dum campus universitário, onde o intuito será controlá-la através de EEG.

Para tal, este projeto utiliza o MindWave como dispositivo EEG para controlo da cadeira de rodas. Os sinais são detetados através do piscar de olhos, dependendo da intensidade dos mesmos. Para o jogador escolher a direção da cadeira é-lhe mostrado no ecrã setas indicadoras (esquerda, direita, frente). Essas mesmas setas destacam-se alternadamente durante 2 segundos cada. Assim, o utilizador terá que piscar os olhos quando a seta destacada indicar para onde se pretende deslocar (Figura 10).

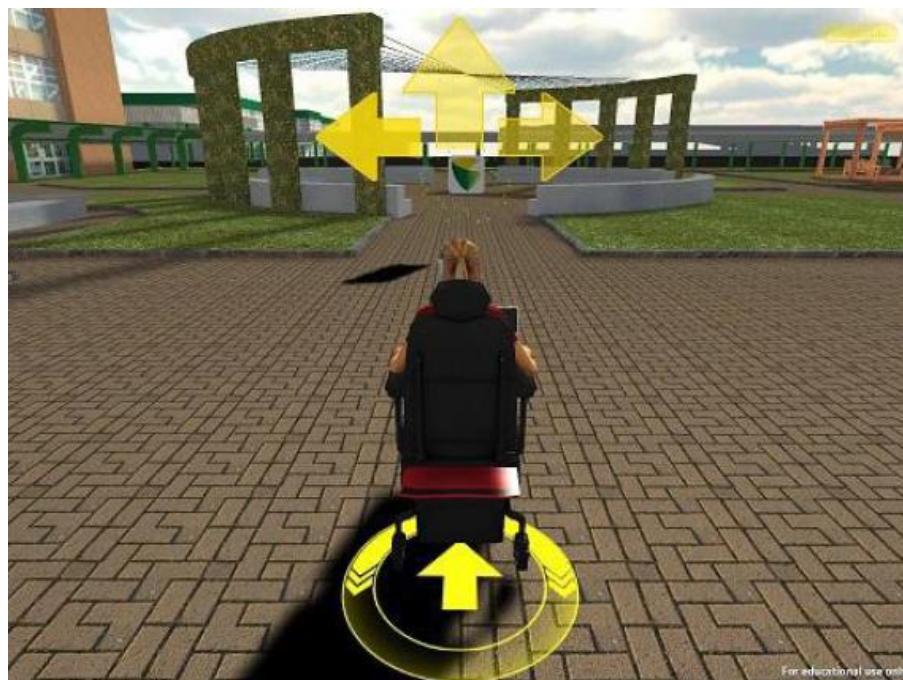


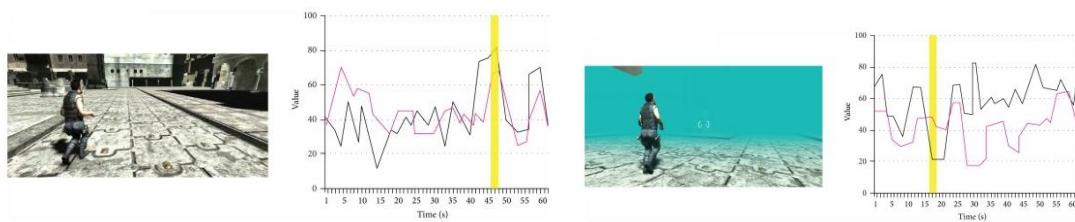
Figura 10 - Simulador cadeira de rodas

### 2.1.3.2 Alteração do ambiente de jogo com base em sensores BCI

Este projeto consiste na alteração em tempo real do ambiente de um jogo FPS (First Personal Shooter), estes ambientes de jogo (Figura 11) são visualizados na perspetiva de primeira pessoa.

Para a alteração do ambiente é utilizado como facto o estado emocional do jogador durante o jogo. O dispositivo utilizado para captação dos sinais cerebrais é o *MindSet* (Figura 12) da Neurosky.

Captando a atenção e a meditação do jogador durante o jogo, o ambiente no cenário de jogo transforma-se consoante os valores captados. As alterações no ambiente afetam a visibilidade do jogador, caso a sua atenção diminua o cenário torna-se escuro impedindo o jogador de ver, caso a meditação diminua o cenário de jogo fica com nevoeiro.



*Figura 11 – Cenário do jogo com respectivos níveis de atenção*



*Figura 12 – Dispositivo MindSet*

## 2.2 Tecnologias utilizadas

Nos dias de hoje, podemos assistir a um avanço tecnológico constante que nos influencia no nosso dia-a-dia. A existência, cada vez maior, de dispositivos de elevada *performance* é um exemplo disso mesmo. Onde podemos constatar o facto de existir um crescimento exponencial de dispositivos móveis como *smartphones* ou *tablets*, com cada vez maiores capacidades de processamento, combatendo até alguns computadores. Assim, não só computadores de elevado desempenho têm a capacidade de execução de determinadas aplicações.

É com base nestes fatores, na contante evolução e tendências dos utilizadores que a escolha das tecnologias indicadas à realização de determinado projeto é importantíssima.

### 2.2.1 Unity

Também conhecido como Unity 3D (Figura 13), trata-se de um *software* para desenvolvimento criado pela Unity Technologies. Este contém as ferramentas e serviços necessários para a construção de conteúdos interativos, como jogos em 2D e 3D.

As capacidades do motor de jogo permitem uma qualidade gráfica elevada e existe uma grande variedade de objetos gráficos (prédios, estradas, casas, carros, etc.), desenvolvidos noutros programas especializados na conceção gráfica de modelos, como por exemplo o CityEngine ou o SketchUp. Para programadores independentes, o Unity, supera os fatores económicos, pelo facto de ser gratuito na sua versão básica, e temporais porque integração um conjunto alargado de tecnologia disponibilizando desta forma uma plataforma simples de desenvolvimento.

Apesar das suas semelhanças com outros produtos como *Blender* ou *Torque Game Engine*, esta por sua vez, é bastante mais intuitiva o que torna mais fácil a sua utilização.

Outra das vantagens do Unity é a facilidade de exportação para diversas plataformas como Windows, MAC OS, Linux, Web, Android, IOS, BlackBerry, PlayStation 4, PlayStation Vita, PlayStation Mobile e Xbox. Tem ainda o grande benefício de ter compatibilidade com diversos programas de modelos em três dimensões, como o *Blender*, o que permite a importação desses mesmos modelos.

Para o desenvolvimento nesta plataforma poderão ser utilizadas três linguagens de programação diferentes, o C#, JavaScript e Boo.

Existem várias versões deste *Software*, neste momento a versão mais recente é a 4.5.5. A par das versões gratuitas a plataforma possui uma versão PRO, que possui mais recursos mas que é paga.



*Figura 13 – Logotipo Unity*

### **2.2.2 EEG**

A Eletroencefalografia [21] reside no estudo do registo das correntes elétricas produzidas no encéfalo. Este é realizado por intermédio de elétrodos aplicados no couro cabeludo. A tecnologia EEG é uma interface cérebro-computador bastante estudada, principalmente pela sua precisão, facilidade de uso, portabilidade e baixo custo de instalação. Frequentemente um apoio clínico para diagnóstico de casos como epilepsia, distúrbios de sono ou mesmo o coma. Existe ainda um vasto leque de empregabilidade desta tecnologia, como por exemplo, o suporte a pessoas com limitações motoras, ou lesões cerebrais. Ao nível do entretenimento, é ainda uma área com bastante terreno por explorar, o que suscita bastante interesse. A Figura 14 apresenta uma pessoa utilizando EEG.

É então com suporte nesta área do entretenimento que surgiu a oportunidade de explorar esta tecnologia, complementando-a com a utilização da plataforma Unity.



*Figura 14 – Pessoa utilizando EEG*

### 2.2.3 SSVEP

*Steady State Visually Evoked Potential*, são sinais a uma resposta natural a estímulos visuais, estes variam entre 3,5 ate 75 Hertz. O cérebro gera energia elétrica à mesma frequência (ou múltiplos desta) do estímulo visual. Estímulos utilizados para captação do sinal dos utilizadores, para estes controlarem o jogo.

### 2.2.4 C#

A linguagem C# (Figura 15), é uma linguagem de programação criada para o desenvolvimento de aplicações assentes sobre a plataforma .NET Framework. É simples, poderosa, com *Strong typing* e orientada a objetos.

As inovações nesta linguagem de programação permitem aos programadores um desenvolvimento rápido de variados tipos de aplicações, conservando a expressividade do estilo de linguagens C. A sintaxe é baseada em C++, mas assemelha-se bastante a outras linguagens como é o caso do Java.

A biblioteca de classes existente na plataforma onde esta assenta providênci a acesso a vários serviços do sistema operativo, contem ainda classes bastante bem estruturadas que aceleram significativamente o processo de desenvolvimento.

Linguagem utilizada na programação do projeto na plataforma *Unity*.



Figura 15 – Microsoft Visual C#

### 2.2.5 Java

Linguagem de programação de alto nível desenvolvida pela *Sun Microsystems* (Figura 16). A linguagem permite o desenvolvimento de aplicações concorrentes de forma nativa, isto significa que poderão ser executadas diversas tarefas ao mesmo tempo de forma não sequencial, não havendo a necessidade de uma tarefa terminar para outra poder ser executada.

É uma linguagem orientada a objetos, que utiliza a herança através da definição de classes de objetos. Esta linguagem foi concebida para ter o menor número de dependências possível, isto é, permite que os programadores compilem o código uma única vez e não seja necessária a sua recompilação para ser executado noutra computador. A este processo dá-se o nome de WORA (write once, run anywhere). Esta linguagem herdou da linguagem C a sintaxe e a semântica e com a incorporação de classes assemelha-se ao C++, mas é mais simples de utilizar do que esta.

Aplicações desenvolvidas nesta linguagem podem ser executadas em qualquer tipo de dispositivo, desde que possua a Java Virtual Machine (JVM). Além disso podem ser executadas num só computador, ou distribuídas entre servidores e clientes numa rede.



Figura 16 – Logotipo Java

### 2.2.6 MindWave

Produzido pela NeuroSky (Figura 3) é um dispositivo EEG mais barato que os seus concorrentes, sendo esta a sua grande vantagem face a outros dispositivos. A sua rápida e fácil utilização é outro ponto a seu favor.

É um dispositivo direcionado tanto para o intuito educacional como de entretenimento.

A sua funcionalidade é a medição e projeção de ondas cerebrais. Para isso a captação do sinal é efetuada através de elétrodos, um colocado na zona frontal da cabeça e outro na orelha. Este dispositivo já possui algoritmos que disponibilizam os dados tratados, assim, é possível obter automaticamente o nível de atenção do utilizador. A monitorização é feita com base nos níveis de atenção dos utilizadores quando estes interagem com diversas aplicações, como por exemplo aplicações matemáticas, memória ou reconhecimento de padrões.

Os dados EEG são transmitidos através de uma interface Wi-Fi para o computador.

### 2.2.7 ThinkGear

É um programa que corre no nosso computador em *Background*. Este é responsável por encaminhar os dados do MindWave através de uma porta serie, até ao nosso computador [15]. Os dados são passados para o computador através de um *Socket*.

Isto permite que jogos ou aplicações especiais reajam aos sinais cerebrais detetados pelo MindWave.

### 2.2.8 g.USBamp

Este sistema (Figura 17), é um amplificador de sinais biológicos de alta performance e de alta precisão. Permite a investigação de atividade em diversas partes do corpo, tais como, cérebro, coração, músculos, movimento dos olhos, respiração entre outros parâmetros fisiológicos e físicos [7]. Tendo em conta as suas especificações técnicas, tornou-se um instrumento padrão para diversas áreas de pesquisa, incluindo neuropsicologia, investigação médica e neurofeedback ou para pesquisa BCI.

Fornece um sinal EEG digital com uma relação sinal-ruído bastante superior à do sistema MindWave. Isto também pelo facto de serem utilizados elétrodos com gel, o que melhora a qualidade dos sinais captados [6].



Figura 17 – Dispositivo g.USBamp da g-Tec

### 2.2.9 CityEngine

*Esri CityEngine* (Figura 18), é uma aplicação para a criação e desenvolvimento de modelos tridimensionais. Concebida pela *Esri R&D Center Zurich*, é um *software* especializado em modelos 3D para áreas urbanas.

Esta aplicação é paga mas existe a possibilidade de efetuar um teste gratuito durante 30 dias.



Figura 18 – Logotipo CityEngine

### 2.2.10 SketchUp

SketchUp (Figura 19), é um programa para desenvolvimento de modelos 3D para diversas aplicações, tais como, arquitetura, design de interiores, engenharia civil e mecânica, cinema ou design de jogos de vídeo [11].

Existe um repositório *online* pertencente a este mesmo programa (3D Warehouse), com os mais diversos tipos de modelos 3D possíveis de adquirir gratuitamente. Existe também a possibilidade de qualquer utilizador poder contribuir, colocando modelos 3D disponíveis para outros utilizadores.

Esta aplicação existe em duas versões, uma mais simples e por isso mesmo gratuita (SketchUp Make), e outra paga com mais funcionalidades (SketchUp Pro).



Figura 19 – Logotipo SketchUp

### 2.2.11 Blender

O Blender (Figura 20), é um *Software* profissional, gratuito e *Open-Source*, para a criação de modelos 3D com fins científicos ou industriais.

Este programa pode ser utilizado para diversos tipos de aplicações 3D, tais como, criação de filmes animados, efeitos visuais, modelos para impressão em 3D, aplicações interativas em 3D ou jogos de vídeo. Para além dos recursos para a criação destes diversos modelos 3D, esta plataforma tem o seu próprio motor de jogo integrado [13].



Figura 20 – Logotipo Blender



### 3 Especificação do sistema

Nos capítulos seguintes vamos apresentar o jogo desenvolvido que apelidamos de DriveByMind, um jogo controlado pela mente para a condução de veículos em cenários em 2D e 3D. Este jogo é constituído por um conjunto de níveis diferentes, à escolha do utilizador, com objetivos e dificuldades distintas em cada um desses mesmos níveis. O jogo tem diversos tipos de controlo, para além dos controlos usuais, teclado rato na versão para PC, é possível controlar através de *touch* na versão para Android, e por EEG em ambas as versões.



Figura 21 – Logotipo do jogo DriveByMind

#### 3.1 Introdução ao jogo

Ao iniciar o jogo é mostrado uma pequena sequência de objetos em movimento, uma pequena apresentação do DriveByMind para que o jogador fique a conhecer minimamente o jogo. Nesta cena são mostrados os objetos essenciais do mesmo, como a cidade, um helicóptero a apanhar uma moeda, o helicóptero a passar um anel, o aparecimento dos estímulos visuais ao som da música, bem como um pequeno táxi a ir ao encontro de um passageiro.



*Figura 22 – Introdução do DriveByMind*

### **3.2 Menu**

O menu inicial (Figura 23) permite aos jogadores do DriveByMind escolher as várias opções diferentes opções que este disponibiliza, como por exemplo, na opção Jogar, qual o cenário (2D ou 3D) ou qual jogo deseja jogar (Helicóptero ou Carros). Os utilizadores podem escolher manualmente qual a frequência desejada para os estímulos (SSVEP) ou podem optar pela opção calibração, esta opção verifica a frequência à qual o utilizador responde melhor aos estímulos, tudo isto através da opção Opções. Esta opção é deveras importante pelo facto de os utilizadores responderem de forma distintas às frequências dos estímulos. Desta forma os estímulos são configurados especialmente para cada jogador de modo a responder aos mesmos da melhor forma possível para um ótimo controlo do jogo.

O menu permite ainda escolher o modo como deseja controlar o jogo (EEG ou teclado).



*Figura 23 – Menu inicial DriveByMind*

### 3.2.1 Jogar

Neste menu (Figura 24) o utilizador decide qual dos três jogos disponíveis que pretende jogar.



*Figura 24 – Menu jogar DriveByMind*

### 3.2.1.1 Carros2D

Este menu (Figura 25) é referente ao jogo Carros2D, onde o utilizador seleciona o modo como irá controlá-lo.

O jogo consiste num cenário 2D. Neste cenário existem obstáculos e um carro, este último controlado pelo utilizador.

Este jogo poderá ser controlado por MindWave, touch ou teclado.



Figura 25 – Menu Carros2D

### 3.2.1.2 Carros3D

Este menu (Figura 26) é referente ao jogo de Carros3D, onde o utilizador seleciona o modo como irá controlá-lo.

O jogo consiste num cenário 3D, onde um táxi se desloca numa cidade e tem como objetivo levar pessoas a diferentes destinos. Cada vez que o jogador deixa uma pessoa no destino correto são-lhe atribuídos pontos de jogo.

Para controlar o táxi o jogador poderá utilizar o sistema g.USBamp da gtec ou o teclado.



*Figura 26 – Menu Carros3D*

### 3.2.1.3 Helicóptero

Este menu (Figura 27) é referente ao jogo do Helicóptero, onde o utilizador seleciona o modo como irá controlá-lo.

O jogo consiste num cenário 3D, onde um helicóptero sobrevoa uma cidade. O jogador terá de controlar o helicóptero através de comandos para esquerda e para a direita.

Este jogo tem como objetivo o controlo de um helicóptero de forma a recolher moedas espalhadas pelo cenário, cada moeda apanhada aumenta a pontuação do jogador.

Para controlar este jogo poderá ser utilizado tanto o teclado como o g.USBamp.



*Figura 27 - Menu Helicóptero*

### 3.2.2 Opções

No menu de opções do jogo (Figura 28) o utilizador poderá escolher a que frequências as setas dos estímulos virtuais piscam, estas frequências são em Hertz.

Neste menu o jogador poderá escolher a opção “calibração”, esta opção mostra a cada 10 segundos uma seta a piscar com uma determinada frequência. O sistema captará a resposta aos estímulos a diferentes frequências e utilizará as frequências às quais o utilizador respondeu melhor.

Para cada uma das setas direcionais existirá uma frequência diferente, sendo assim possível o jogador controlar o jogo através do g.USBamp.



*Figura 28 – Menu opções DriveByMind*

### 3.2.2.1 Frequências

Neste menu (Figura 29) o utilizador poderá escolher manualmente qual a frequência que deseja para cada uma das setas (esquerda, direita, frente), aumentando-as ou diminuindo-as nos botões com sinais mais e menos. Depois de definidas as frequências o utilizador deverá clicar no botão definir para estas mesmas frequências serem utilizadas no jogo.



Figura 29 – Menu definição manual frequências

### 3.2.2.2 Escolher Estímulos

Neste menu (Figura 30) o utilizador poderá escolher duas cores diferentes para os estímulos visuais. As cores deverão fazer um contraste elevado, para o sinal ser o mais percepível possível.

Depois de escolhidas as cores o utilizador deverá clicar em definir para estas mesmas cores serem utilizadas no controlo do jogo.

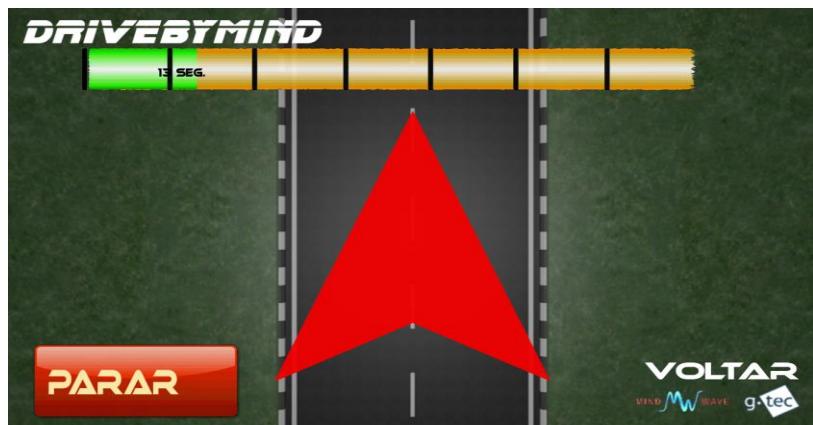


Figura 30 – Menu escolha de estímulos

### 3.2.2.3 Calibrar

O menu de calibração (Figura 31) tem como finalidade a definição automática por parte do sistema dos estímulos visuais para controlo do jogo.

Durante 70 segundos o sistema apresenta no ecrã estímulos a diferentes frequências e irá verificar a que frequências o utilizador responde melhor, aplicando-as de seguida no jogo.



*Figura 31 – Menu calibração dos estímulos*

### 3.2.2.4 Comunicação do Unity com Matlab/Simulink

A aquisição de sinal e implementação dos algoritmos de deteção dos padrões EEG são realizados num módulo independente do Unity, recorrendo ao Highspeed Simulink (integrado no Matlab), que permite aquisição e processamento de sinal em tempo real. O Unity comunica com o Matlab/Simulink através de um canal comunicação TCP/IP. Foram definidos códigos associados aos vários estímulos, modos de jogo, calibração etc.

O menu dos valores relativos ao Matlab (Figura 32), contém os comandos que o jogo irá enviar ou receber via TCP, para que ambas as aplicações possam trocar algumas informações sobre o estado do jogo.

Os valores enviados pelo Matlab podem ser de direção e calibração. Em relação às opções direcionais, o Matlab envia os valores para a Esquerda (comando “65”), Direita (comando “66”), Frente (comando “67”) e Trás (comando “68”). Para a calibração são enviados sete comandos, com os valores de “70” a “76”, ou seja, desde a frequência 1 à frequência 7.

Como referido anteriormente, o DriveByMind também enviará alguns comandos para o Matlab, para este saber algumas informações sobre o que está a acontecer no jogo. Estas informações serão enviadas sob forma de comandos via TCP, servindo estes para informar o Matlab sobre o modo de jogo e indicação de início de estímulos. Sobre o modo de jogo, o DriveByMind pode enviar o comando “80”, que é relativo ao modo de jogo 1 (condução automática, apenas controlo de direção nas bifurcações), e enviar o comando “81” (controlo quase contínuo, com envio de comandos de um em um segundo). Para além dos valores de modo de jogo, o DriveByMind poderá enviar os comandos “82” e “83”, que são relativos ao pedido de início de estímulo e início de calibração respetivamente.



Figura 32 – Menu de comandos para comunicação Unity / MatLab

### 3.2.3 Sair

Opção do menu de jogo para o jogador sair do jogo. Após o jogador escolher esta opção a aplicação será fechada e o jogo não será guardado.

## 3.3 Níveis

Nas secções seguintes vamos apresentar de forma detalhada o jogo DriveByMind. Para uma melhor compreensão da sua implementação faremos uma definição de alguns elementos que fazem parte do Unity.

- Cenário – Contém os objetos do jogo.
- BoxCollider – Colisão básica em forma de cubo para deteções de interações entre objetos.

- Sprite – Objetos gráficos 2D utilizados para personagens, adereços, projeteis e outros elementos da jogabilidade 2D.
- Rigidbody – Objeto de jogo sujeito às leis da física.
- Scripts – Código executável que controla os objetos do jogo.
- Transform – Transformações geométricas (posição rotação e escala) de um objeto.
- Câmera – Dispositivo através do qual o jogador vê o cenário de jogo.
- Animation – Componente utilizado para reproduzir animações.

Para uma explicação mais detalhada de cada um dos componentes é fornecido um tutorial de Unity que se encontra no anexo 3.

### 3.3.1 Carros 2D

Este nível do jogo foi inspirado no LiveBiofeedback, tendo o mesmo objetivo de jogo. Onde o utilizador controla um carro num cenário em duas dimensões, desviando-se dos obstáculos que vão surgindo pela pista onde o carro se encontra.

Ao colidir com os obstáculos serão retirados pontos ao jogador.

Para controlar o carro o utilizador pode utilizar o teclado o sistema MindWave da NeuroSky ou através de estímulos visuais em conjunto com o sistema g.USBamp.

Com o sistema MindWave consoante os níveis de concentração do jogador o carro move-se para cima ou para baixo. Para controlo do carro com o sistema g.USBamp são mostradas no ecrã de jogo duas setas direcionais (cima e baixo) a piscar a diferentes frequências, focando o olhar na seta que indica a direção que deseja o carro move-se.

#### 3.3.1.1 Arquitectura do jogo

Sendo este jogo proveniente do projeto LiveBiofeedback, alguns dos objetos utilizados na sua construção são eles também provenientes do mesmo projeto. Objetos como a pista, o carro, ou os obstáculos são os mesmos utilizados no LiveBiofeedback, adaptados agora para a plataforma Unity.

Toda a arquitetura do jogo é composta pelos seguintes elementos:

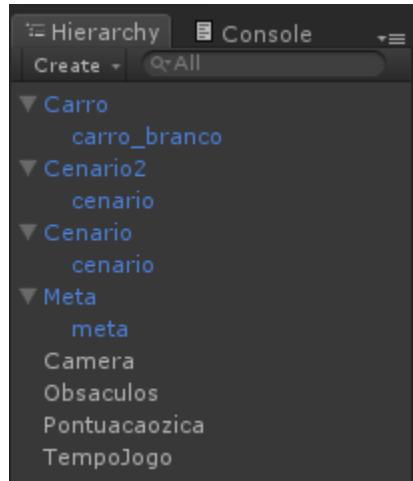


Figura 33 – Hierarquia do jogo Carros2D

- Pista (Cenario, Cenario2, Meta)

O jogo é constituído por dois objetos que representam a pista de jogo. Estes objetos têm entre outros, um componente designado por *Sprite Renderer*.

O *Sprite Renderer* é utilizado para ilustrar imagens do tipo *Sprite*, que consiste num conjunto de imagens aplicadas numa imagem apenas, em cenários 2D ou 3D [29]. A este componente associamos a imagem da pista de jogo que será o nosso cenário.

Para limitarmos o cenário de jogo, ou seja, a nossa pista onde o carro se desloca adicionamos um componente ao cenário, uma *Box Collider* em cada um dos limites da pista (superior e inferior), assim podemos detetar quando o objeto carro embateu num dos limites da pista.

Na Figura 34 pode verificar-se os respetivos componentes adicionados ao objeto “Cenario”, neste caso será a nossa pista do jogo.

O modo de funcionamento de jogo consiste na alteração da posição das imagens da pista no eixo dos *xx* a uma determinada velocidade, neste caso negativa, fazendo a pista mover-se para trás. Criando assim a impressão do movimento do carro na direção contrária. Para tal, utilizamos a função *Transform* (Código 1), esta permite a alteração da posição, rotação ou escala de um objeto do cenário de jogo [30].

A função *Vector2* (Código 1), faz a representação de vetores e pontos 2D, neste caso específico representa a posição que a imagem da estrada irá tomar.

```

public class MovEstrada : MonoBehaviour {

    public float velocidade = -2f;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        transform.Translate (velocidade*Time.deltaTime, 0, 0);
        if (transform.position.x < -13) {
            transform.position = new Vector2(38, transform.position.y);
        }
    }
}

```

Código 1 – Movimento da estrada (pista)

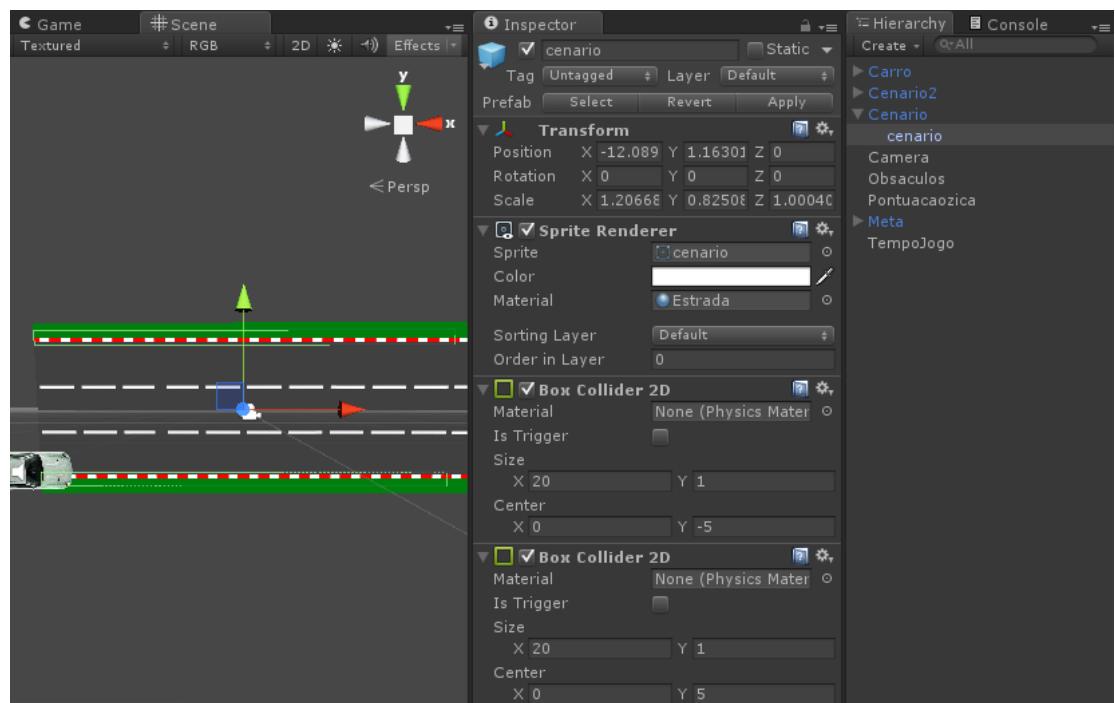


Figura 34 – Componentes da pista de jogo

- Carro

No jogo dos carros2D o objeto carro é o foco principal, sendo este o objeto controlado pelo utilizador.

Este objeto é representado no ecrã através do mesmo método da pista de jogo, utilizando o *Sprite Renderer*. A este componente adicionamos uma imagens de um carro representando assim o nosso objeto no ecrã de jogo.

Ao objeto carro adicionamos um componente da física do jogo, o *Rigidbody 2D*. Este componente é adicionado a um *Sprite Renderer* com o intuito de o

colocar sobre controlo das físicas do motor de jogo, assim, este mesmo objecto será afetado pela gravidade e poderá ser controlado através de *Scripts* [31].

Para sabermos os limites do carro e detetarmos colisões do mesmo com outros objetos adicionamos-lhe o componente *Box Collider 2D* [32]. Este componente é representado por um retângulo em volta do objeto em questão.

O movimento do objeto carro é feito através da função *Transform* (Código 2).

Visto que este objeto apenas se desloca no eixo dos *Ys*, através da função *Translate*, que faz mover o objeto consoante a tecla que o utilizador carrega (cima ou baixo), a posição do carro é alterada.

```
public class Movimento : MonoBehaviour {

    // Use this for initialization
    public float velocidade;
    void Start () {
        Time.timeScale = 1.0f;
    }

    // Update is called once per frame
    void Update () {
        // mexer conforme a tecla
        transform.Translate (0,Input.GetAxis ("Vertical")*Time.deltaTime*velocidade, -5);
        //Mover apenas eixo do y
        transform.position = new Vector3 (-10, transform.position.y, -2);
    }
}
```

#### *Código 2 – Movimento do carro2D*

A este objeto ainda são adicionados outros componentes, como áudio e um respetivo *Script*, assim ao embater num objeto é emitido um som. Na Figura 35 podemos verificar todos os componentes associados ao objeto carro.

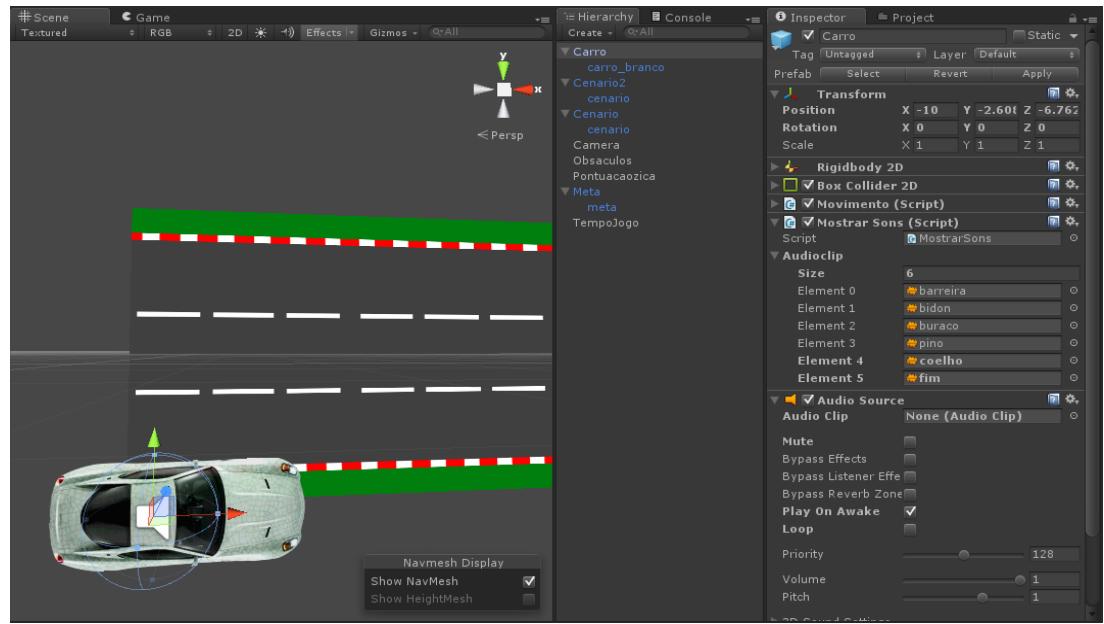


Figura 35 – Componentes do objecto carro 2D

- Câmera

O *GameObject Camera* (Figura 36) mostra-nos durante a execução do jogo o cenário. No caso deste cenário 2D a câmara é posicionada de modo a ser visível toda a pista e o carro. Neste jogo a câmara é um objeto estático, sendo os restantes objetos a moverem-se. A este objeto adicionamos também um componente de áudio, que será o som do jogo, enquanto este está em execução.

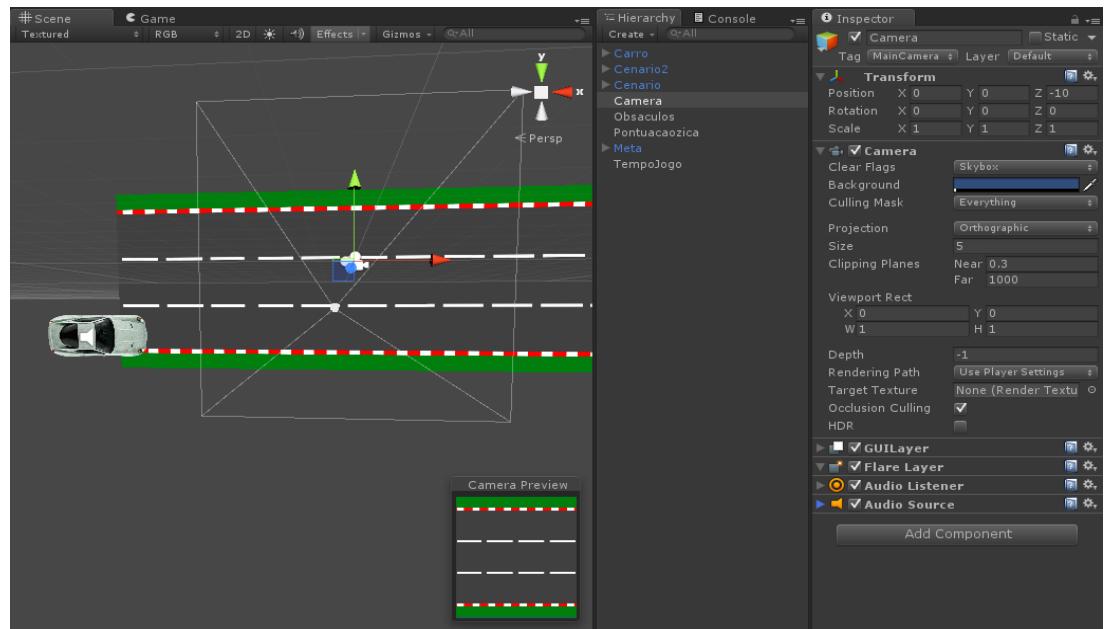


Figura 36 – Componentes do objecto Camera

- Obstáculos

O *GameObject* Obstáculos representa os diferentes obstáculos existentes no jogo que vão aparecendo aleatoriamente na pista.

Para tal, criámos uma lista de *GameObjects* obstáculos, onde constam os diferentes objetos que representam os mesmos. Estes objetos são barreiras, buracos, sinal de STOP, etc. A esses objetos é individualmente associado um componente *Sprite Renderer*, para ser possível a cada objeto obstáculo ser definida uma determinada imagem representativa do mesmo. A estes objetos definimos, através de um *Script* a eles associados, uma altura máxima e mínima para estes serem aleatoriamente dispostos apenas dentro da pista. Existe um objeto temporário que irá conter o *GameObject* Obstáculo quando for necessário coloca-lo no jogo (Código 3).

Na função *Random.Range* podemos definir um intervalo mínimo e máximo para o sistema aleatoriamente obter um valor.

```
public List<GameObject> ListaObstaculos;
public void PosicaoObstaculo(){
    // posicao aleatoria dos obstaculos
    int posicaoaleatoria = Random.Range (alturaMinima,alturaMaxima);
    // escolha aleatoria do obstaculo a mostrar
    int obstaculoMostrar = Random.Range (0,6);
    // criação de um Objecto temporario e alocar o Obstaculo
    GameObject tempObj = ListaObstaculos [obstaculoMostrar];
    //Criação do Obstaculo no cenário
    tempObj.transform.position = new Vector3 (transform.position.x, posicaoaleatoria,-1 );
    // Activar o Obstaculo
    tempObj.SetActive (true);
}
```

Código 3 – Colocação aleatória de obstáculos no jogo

- Indução de concentração através de cálculos matemáticos

Uma das questões que nos fez abandonar o sistema MindWave e adotar outro método para controlar o nosso jogo nos outros níveis diferentes foi a sua fraca fiabilidade no que diz respeito aos sinais EEG captados.

Para tal teríamos que testar se de facto, o sistema da NeuroSky consegue captar níveis de atenção de um utilizador. Foram introduzidas cálculos matemáticos que são mostrados na pista durante o decorrer do jogo. Assim, conseguimos ter uma noção se ao ter que se concentrar na conta para a realizar o movimento do carro se altera.

Os cálculos introduzidos foram cálculos de somar, subtrair, multiplicar ou dividir simples, com apenas dois números. Estes números podem variar de 0 a 20.

Estes cálculos são mostrados aleatoriamente no ecrã através do método *Random* (Código 4).

```
void MostrarContas(){
    // Escolha numero entre 0 e 20
    numero1 = Random.Range (0, 20);
    // Escolha numero entre 0 e 20
    numero2 = Random.Range (0, 20);
    // Escolha numero entre 0 e 4 (+, -, *, /)
    operador = Random.Range (0, 4);
}
```

Código 4 – Mostrar cálculos no ecrã

- Conexão MindWave

Para o estabelecimento da conexão entre o dispositivo da NeuroSky MindWave e o jogo Carros2D são utilizados *Sockets TCP/IP*. Do lado do Unity é utilizado um cliente TCP, o servidor foi implementado em Java.

O servidor utilizado por nós para a conexão é, no geral, o mesmo do projeto LiveBiofeedback, por isso as classes por nós criadas utilizam outras classes anteriormente criadas pelos nossos colegas como podemos ver no diagrama de classes (Figura 38). Visto que o sistema funciona de forma semelhante, alteramos no servidor TCP apenas o essencial para satisfazer as nossas necessidades. Para tal, construímos uma classe que herda propriedades de uma *JFrame*, isto para o utilizador ter um ambiente gráfico, simples e de fácil utilização para conectar o dispositivo com o jogo. Nesta classe referenciamos a classe NeuroServer criada pelos nossos colegas do projeto LiveBiofeedback, que através do método NeuroServer nos é passado por parâmetro os valores recebidos pelo MindWave (Código 5).

```

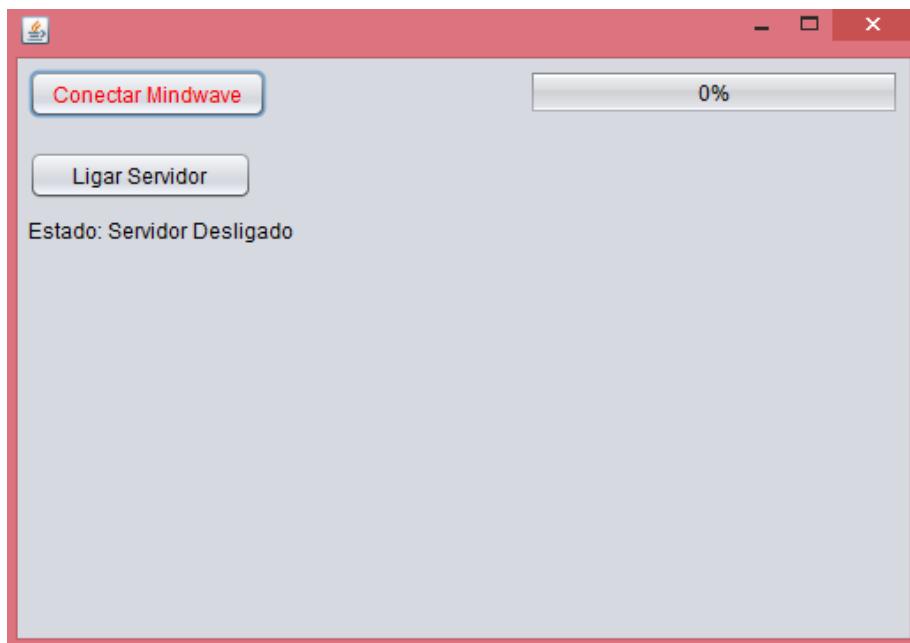
public NeuroServer server;

public Mindwave_Server() {
    initComponents();
    this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    server = new NeuroServer(FFTValues, concentra, timevalues, potencia);
    server.init();
}

```

*Código 5 – Valores recebidos por parâmetro da classe NeuroServer*

A nossa classe Mindwave\_Server que herda as propriedades de uma classe JFrame, é a classe utilizada para conectar o dispositivo MindWave ao jogo como antes referido. Para isso, existe uma janela onde o utilizador pode ver a atividade em tempo real do dispositivo através de uma *Progressbar* que se altera consoante o nível de atenção do utilizador, existe também um botão para efetuar a conexão com o dispositivo e outro para abrir o canal de comunicação com o *Socket TCP/IP* (Figura 37).



*Figura 37 – Janela conexão MindWave*

Para o envio dos valores do nível de atenção, recebidos do MindWave, para o jogo implementámos a classe Send\_Attention. Esta classe é uma *Thread*, e é aqui que é criado o *Socket* para a comunicação entre o Java e o Unity. Quando o utilizador pressiona o botão “Ligar Servidor” a *Thread* é iniciada e são enviados através do *Socket* os valores dos níveis de atenção.

```

public class Send_Attention extends Thread {

    boolean aux = true;
    Mindwave_Server MindwaveServer;

    /**
     *Trabalho a ser executado pela Thread
     * Verifica se existem servidores disponíveis e adiciona-os a lista
     */
    public void run() {
        try {
            //socket servidor - escuta a porta 10000
            ServerSocket server = new ServerSocket(10000);
            display(server);
            MindwaveServer.Lbl_DadosCliente.setText("Estado : Servidor Ligado [A Esperar Clientes...]");
            int numeros;
            //esperar pela ligação
            Socket socket = server.accept();
            String str = socket.getInetAddress().toString();
            MindwaveServer.Lbl_DadosCliente.setText("Estado : Servidor Ligado ao cliente " + str);
            while (true){
                PrintStream out = new PrintStream(socket.getOutputStream());
                numeros = MindwaveServer.PrgBar_Atencao.getValue();
                System.out.println("ATENCAO THREAD: " + numeros);
                out.println(numeros);
                try {
                    Thread.sleep(10);
                } catch (InterruptedException ex) {
                    Logger.getLogger(Mindwave_Server.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        } catch (IOException ex) {
            Logger.getLogger(Send_Attention.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}

```

#### Código 6 – Classe Send\_Attention

Depois de enviados os dados teremos de recebermos no Unity e implementá-los no nosso jogo, para tal implementámos um cliente TCP/IP.

A nossa classe ClientTCP contém um objeto *TcpClient* (Código 7) para receber por *Socket* os valores enviados pelo servidor Java. Na função Update (Código 8), que é executada a cada frame recebemos os dados pelo *Socket* e deciframo-lo para o podermos utilizar como sendo um número através do método SetTransformY onde passamos o valor da atenção por parâmetro para movermos o carro (Código 9).

```

public class tcpClient : MonoBehaviour {

    // coordenada para mover o carro eixo do y
    float cordY = 0;
    //Valor da atencao
    int value;
    TcpClient client;
    NetworkStream stm;
    byte[] receivebytes = new byte[1024];
    int atencaoAGORA = 0;
    long WaitingTime = 1000;
    long milliseconds;

    void Start () {
        IPAddress ipad = IPAddress.Parse("127.0.0.1");
        client = new TcpClient();
        client.Connect(ipad, 10000);
        stm = client.GetStream();
        milliseconds = System.DateTime.Now.Ticks / System.TimeSpan.TicksPerMillisecond;
        WaitingTime = milliseconds + WaitingTime;
    }
}

```

*Código 7 – Ligação cliente TCP*

```

void Update () {
    //receber dados
    int bytesreceived = stm.Read (receivebytes, 0, 1024);
    //Colocar dados como String
    string str = Encoding.ASCII.GetString (receivebytes, 0, bytesreceived);
    //Converter String num numero
    int.TryParse (str,out value);
    if (value != 0) {
        Debug.Log (value);
        atencaoAGORA = value;
        // cordenada actual do Carro
        cordY = transform.position.y;
        // cordenada para onde ira o Carro
        cordY = (float)((value*5.557126)/100) -2.778563;
        SetTransformY(cordY);
    }
}

```

*Código 8 – Receber dados por TCP*

```

void SetTransformY(float Cordenada){
    //Alterar posicao carro
    this.transform.position = new Vector2 (-7, Cordenada);
}

```

*Código 9 – Move carro através do valor recebido do MindWave*

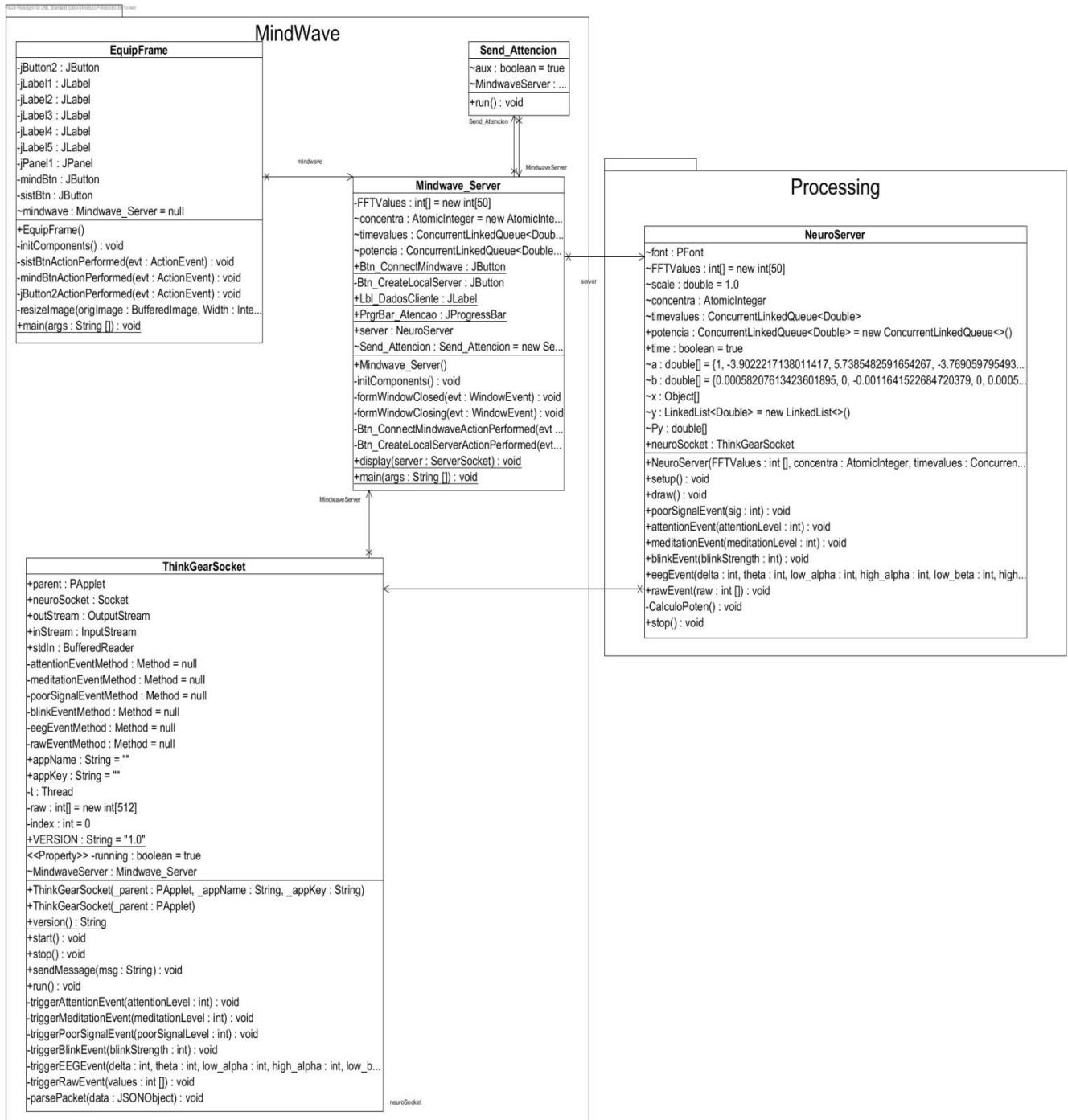
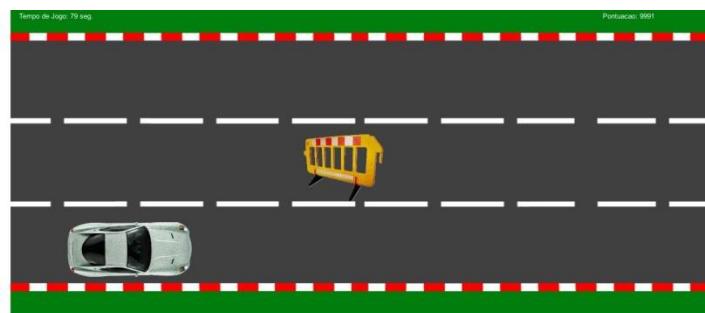


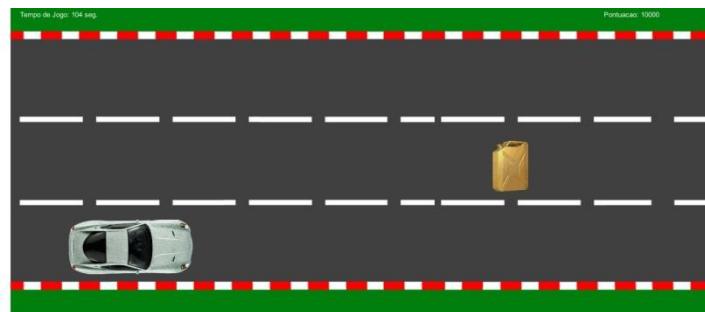
Figura 38 – Diagrama classes da conexão do MindWave

### 3.3.1.2 Implementação

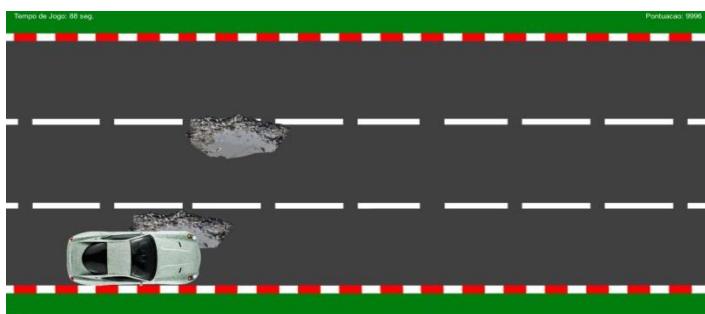
Depois de construído obtemos então o jogo já implementado e a correr no Unity como demonstrado nas Figuras seguintes. Podemos ver os vários objetos (obstáculos) que são dispostos no cenário (pista) aleatoriamente. No ecrã de jogo podemos ver também na parte superior o tempo de jogo e a pontuação atual do jogador que decresce quando este embate num obstáculo. Para a versão *Touch* (Android), existem setas (cima e baixo) para controlo do carro. Na versão do jogo para controlo através do MindWave podemos ver os cálculos matemáticos que aparecem no centro do ecrã sensivelmente.



*Figura 39 – Simulação de jogo (Obstáculo Barreira)*



*Figura 40 – Simulação de jogo (Obstáculo Bidon)*



*Figura 41 – Simulação de jogo (Obstáculo Buraco)*

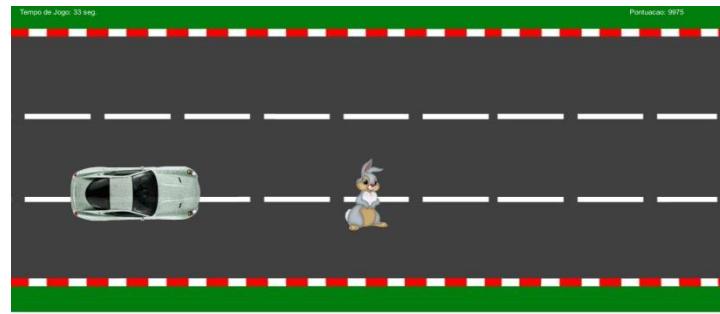


Figura 42 – Simulação de jogo (Obstáculo Coelho)

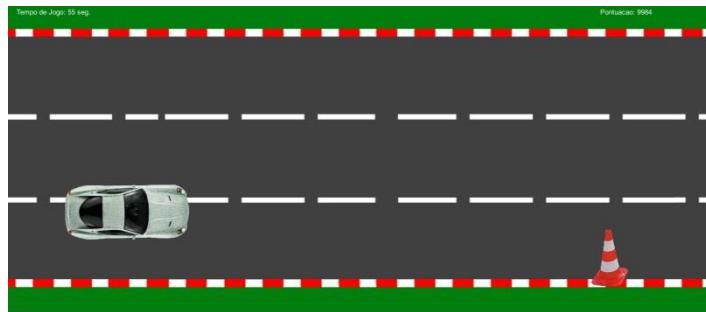


Figura 43 – Simulação de jogo (Obstáculo Pino)

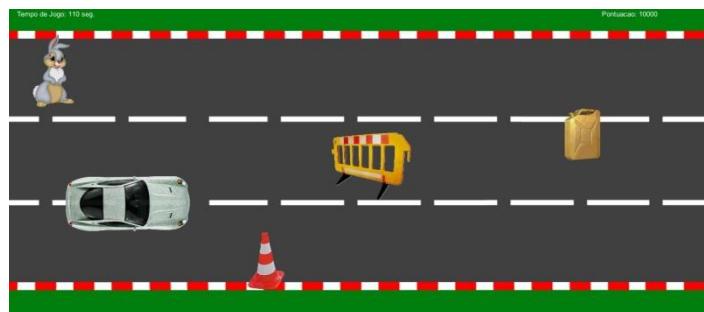


Figura 44 – Simulação de jogo (Vários obstáculos)

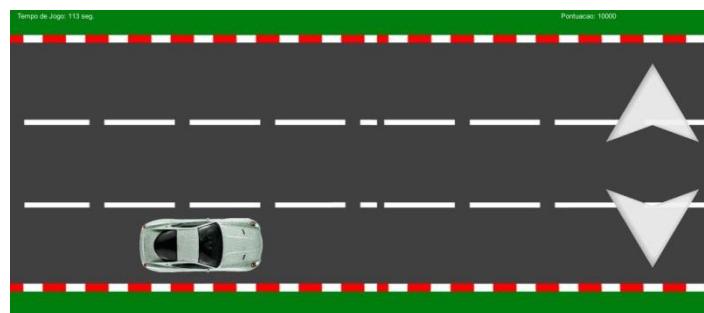
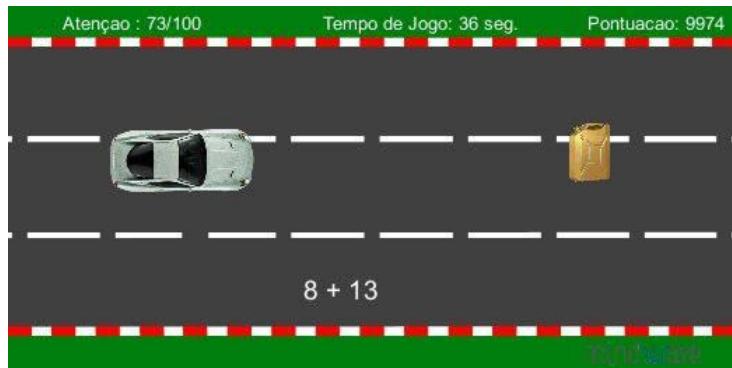


Figura 45 – Simulação de jogo (versão Android)



*Figura 46 – Simulação do jogo versão MindWave*

### 3.3.2 Carros 3D

Neste nível do jogo, o jogador controla um táxi num cenário em 3D, percorrendo as ruas de uma cidade. Nessa cidade existem algumas pessoas que desejam ser transportadas para determinados locais, caso o jogador consiga transportar uma pessoa desde a origem até ao seu destino, este ganha pontos.

Neste cenário de jogo existem setas indicativas da direção que o jogador pretende tomar. Para tal, as setas direcionais, case jogue a versão teclado, ou os estímulos visuais, caso escolha a versão g.USBamp, serão mostrados momentos antes da tomada de decisão. Caso o jogador não escolha nenhuma das opções, então o táxi ficará parado no local da tomada de decisão, neste caso o cruzamento, até que o jogador escolha para onde deseja virar.

Por cima do táxi existe uma seta orientadora para auxiliar o jogador sobre a sua navegação no jogo (Figura 47). Este objeto irá indicar onde se encontram os passageiros que pretendem ser transportados, bem como o seu futuro destino.

Este jogo poderá ser jogado através de PC ou para a plataforma móvel Android. Ainda que não tenho sido testado o jogo encontra-se preparado para ser jogado em dispositivos com plataforma iOS.



*Figura 47 – Seta orientadora*

Este nível de jogo poderá ser controlado através do g.USBamp ou pelo teclado, sendo que o teclado poderá ser utilizado mesmo quando o jogador escolhe controlar o jogo através de EEG.

### 3.3.2.1 Arquitetura do jogo

Para a concretização deste jogo utilizámos vários tipos de objetos, alguns deles criados por nós, outros recorrendo à plataforma de modelos 3D desenvolvida pela Google, 3D Warehouse [28] onde é possível fazer *download* gratuito desses mesmos modelos utilizando o software SketchUp.

A possibilidade de edição dos modelos disponíveis neste repositório e a sua elevada qualidade gráfica foram fatores importantes para a sua utilização neste projeto. A construção de uma cidade para um cenário de jogo em Unity3D é bastante complexa, isto porque há vários fatores a ter em conta, como por exemplo os objetos utilizados, as texturas que esses objetos contêm, os materiais que utilizam e o número de polígonos que contêm. Todos estes fatores são importantíssimos para obtermos um jogo com uma boa *performance*.

Em termos de *design* gráfico, este foi um grande desafio neste projeto, pois a utilização de modelos 3D muito complexos torna o processo de *rendering* do Unity 3D bastante lento, baixando assim o número de FPS o que torna o jogo bastante lento e difícil de jogar.

Toda a arquitetura do jogo é constituída pelos seguintes elementos que iremos descrever:

- Terreno

O objeto terreno no nosso projeto (Figura 48), tem o intuito de ser a base de construção de toda a cidade, ou seja, o chão da cidade. É sobre este objeto que incide toda a arquitetura que faz parte da mesma, como as estradas, prédios, árvores, monumentos, entre outros.

Através das opções de terreno disponibilizadas, foram implementadas algumas texturas ao terreno, como relva, terra, areia, bem como as montanhas que se pode visualizar durante o jogo.



Figura 48 – Terreno de construção do cenário de jogo

- Arquitetura

Os objetos que constituem a arquitetura de jogo são as estradas, passeios, prédios, casas, estádio, aeroporto, entre outros. Estes são compostos por diferentes tipos de materiais e texturas que formam o modelo 3D. Quando importados para o Unity existem algumas definições básicas que podemos alterar no objeto como a sua escala se pretendemos comprimir o objeto ou otimiza-lo de modo a aumentar a *performance* do jogo (Figura 49).

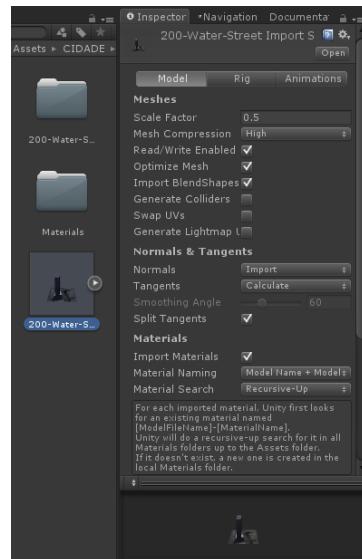


Figura 49 – Propriedades de um modelo 3D (prefab)

Estes modelos que enriquecem a cidade foram descarregados através da plataforma de modelos 3D desenvolvida pela Google, 3D Warehouse [28], e transformados em modelos 3D compatíveis com o Unity através do software SketchUp.

Os modelos descarregados necessitam de ser transformados em modelos .fbx (significa Filmbox, e foi desenvolvido pela Autodesk em 2006) para poderem ser inseridos em Unity. Este processo é feito no software SketchUp, através da opção *Export 3D Model*.

A importação dos objetos para o Unity é feita através de *Drag-and-Drop* do objeto para o software Unity, ou através da opção *Import new Asset*. Depois de importado um modelo para o cenário de jogo (Figura 50), é possível alterar o seu tamanho posição e rodá-lo em torno de qualquer um dos eixos, isto para o dispor no cenário do modo mais adequado. Nas propriedades destes objetos importados podemos ver os diferentes materiais que os compõem e alterá-los ou eliminá-los.

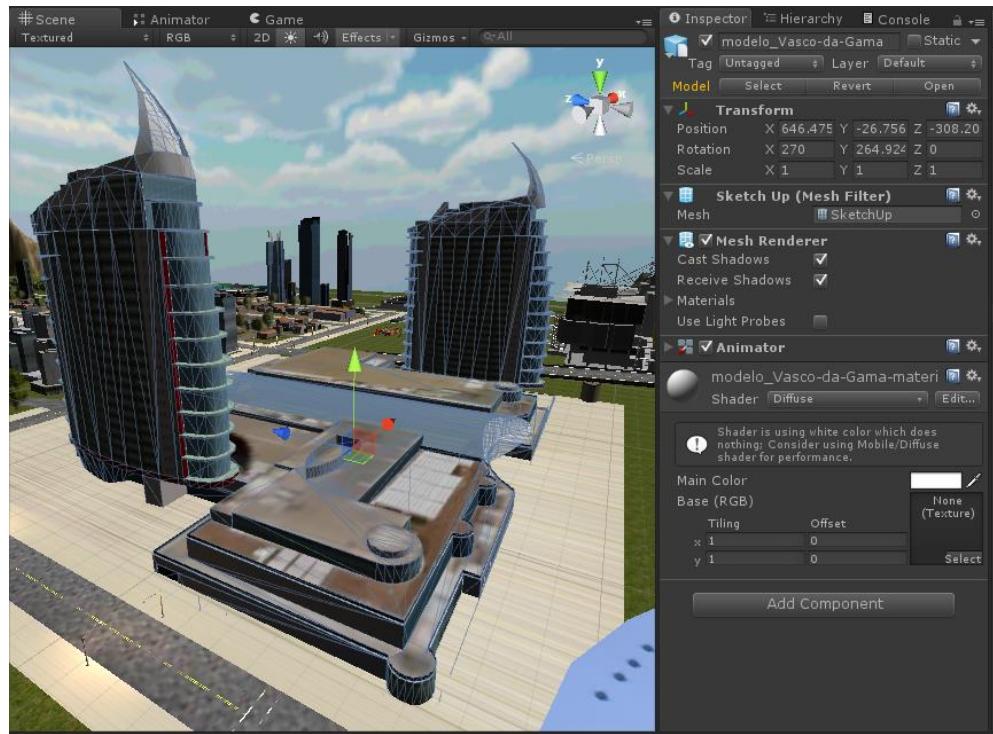


Figura 50 – Modelo 3D no cenário de jogo e suas propriedades

- **Taxi**

Neste modelo de jogo em 3D, o táxi é o objeto que o jogador irá controlar dentro do cenário de jogo.

Este modelo é composto por texturas e materiais que fazem parte do modelo .fbx importado para Unity, denominado *prefab*. Este *prefab* foi moldado conforme as características pretendidas, ou seja, alguns objetos foram separados do modelo original, como a porta, para esta poder ser movida sem interferir com o resto do veículo.

Além destas componentes originais que este modelo inclui, foram adicionados outras, como o *Box Collider*, o *RigidBody*, o *Mesh Renderer*, uma componente de áudio denominada *Audio Source* e a componente de navegação automática intitulada *NavMeshAgent* (Figura 51).

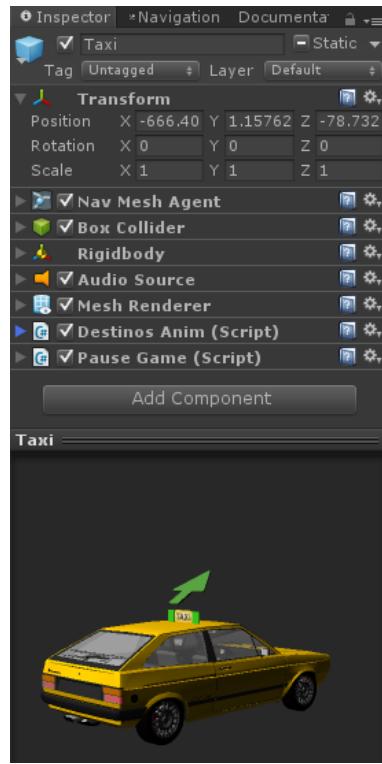


Figura 51 – Componentes do modelo 3D do táxi

O *NavMeshAgent* é uma componente que pode ser implementada em qualquer objeto. Esta componente faz parte da ferramenta *NavMesh*, e serve para definir os locais por onde um objeto se poderá mover entre uma origem e um determinado destino, através de cálculos é determinado o caminho mais próximo. Estes locais serão definidos na janela *Navigation*, através da opção *Bake*, sendo apenas os objetos estáticos a fazer parte dos locais de navegação, no caso do jogo do táxi, apenas a estrada será estática para assim o carro poder mover-se pelas ruas. Os objetos que farão parte da navegação serão marcados a azul (Figura 52).

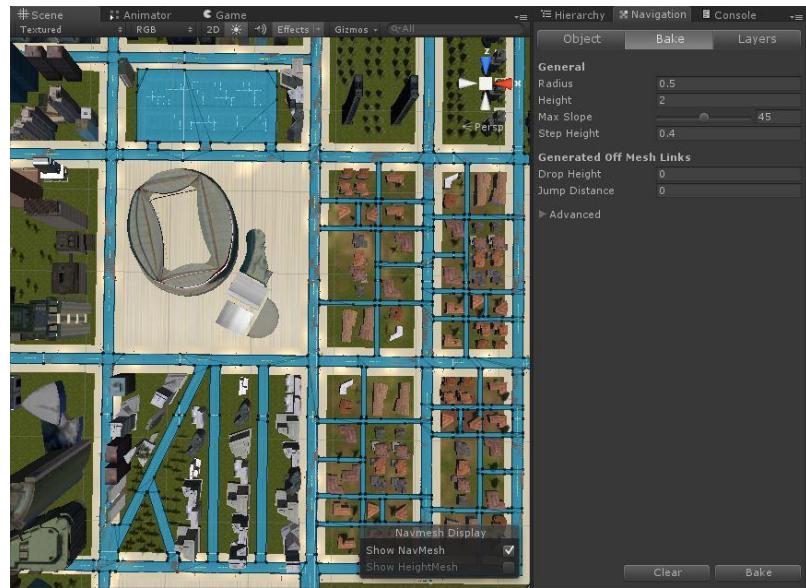


Figura 52 – *NavMesh* jogo *Taxi*

Como referido anteriormente, para o *NavMesh* funcionar corretamente, o objeto em questão, neste caso o táxi, terá de conter o componente *NavMeshAgent* e um pequeno script associado para determinar o destino a atingir (Código 10), onde o objeto de destino terá de ser associado ao *GameObject* público “destino”.

```
using UnityEngine;
using System.Collections;

public class NavMeshDestinos : MonoBehaviour {

    public Transform destino;
    private NavMeshAgent agente;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        agente = gameObject.GetComponent<NavMeshAgent>();
        agente.SetDestination (destino.position);
    }
}
```

Código 10 – Definição de destinos *NavMesh*

O modelo táxi, tendo em sua constituição a componente *NavMeshAgent*, terá também de ter um script de destino associado, denominado *DestinosAnim*, no caso do jogo para teclado, e *DestinosTCPAnim* no caso de jogar através do g.USBamp, sendo que este é muito mais complexo do que o demonstrado em “Código 10” devido a sua grande quantidade de cruzamentos (Figura 53),

sendo que cada cruzamento é um destino possível, para assim o jogador poder escolher livremente o seu caminho.

Para além deste script, foi adicionado um outro, denominado *PauseGame* que serve para pausar o jogo a cada momento do jogo, sempre que o utilizador pressione a tecla *Escape*.



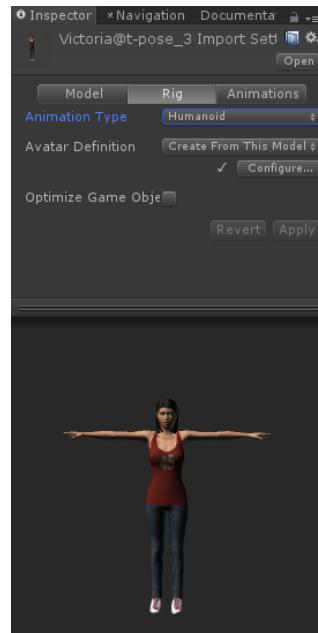
Figura 53 – Destinos jogo Táxi

- Passageiros

Os passageiros são modelos de pessoas em 3D, do tipo *Character Controller*, descarregados através da plataforma Mixamo[34].

Os passageiros, fazem parte do objetivo do jogo, isto porque o jogador tem de controlar o táxi até eles e deixá-los nos seus destinos. Sendo assim, estes são um objeto fundamental para o jogo.

Estes modelos 3D deverão ser do tipo *Character Controller* para lhes poderem ser adicionadas animações, e do tipo *Humanoid*, definido nas opções do modelo (Figura 54).



*Figura 54 – Componentes do modelo 3D Character Controller*

As diferentes componentes do passageiro são um *Animator*, que serve para adicionar uma animação ao modelo, uma *Capsule Collider*, um *Rigidbody*, um *NavMeshAgent* (que servirá para o passageiro ir ao encontro do táxi através de vários pontos de destino a volta do veículo, sendo calculado o destino mais próximo e percorrendo os restantes destinos até a porta do táxi) (Figura 55), e um script que servirá para escolher que animações serão escolhidas nos diferentes momentos, através de cálculos de distâncias e tempo (Código 11), e deteção de colisões para o passageiro percorrer o táxi até a sua respetiva porta e entrada (Código 12).



*Figura 55 – Destinos NavMesh a volta do táxi para o passageiro*

```

// calculo da distancia entre o taxi e o destino
distancia = (int)Vector3.Distance (Taxi.transform.position, destino.transform.position);

if (distancia > 30){
    // Rodar passageiro, para ficar de frente para o taxi
    this.transform.LookAt (referencia.transform);
}

// Caso a distancia seja <100m, o passageiro levanta a mao para chamar o taxi
if (distancia > 2 && distancia < 100) {
    anim.SetBool ("ChamarTaxi", true);
} else {
    anim.SetBool ("ChamarTaxi", false);
}

// AbrirPorta = true e definido quando o passageiro percorre todos os destinos
// a volta do carro e chega ao destino 0
if (AbrirPorta == true) {
    time += Time.deltaTime; //contador de tempo
    if (time > 2.1 && time < 2.9) {
        RotacaoPorta = -40 * Time.deltaTime; // Abrir a Porta
        porta.transform.Rotate (new Vector3 (0, RotacaoPorta, 0));
    } else if (time > 3.1 && time < 3.29) {
        RotacaoPorta = 170 * Time.deltaTime; // Fechar a Porta
        porta.transform.Rotate (new Vector3 (0, RotacaoPorta, 0));
        entrouNoCarro = true;
    } else if (AbrirPorta == true && entrouNoCarro == true) {
        // Colocar a porta na posicao correta, e mandar seguir o carro.
        porta.transform.localEulerAngles = new Vector3 (0, 0, 0);
        SeguirCarro = true;
        AbrirPorta = false;
        AtualizarDestino = true;
    }
}

```

*Código 11 – Mostrar animações, conforme tempos e distâncias*

```

void OnTriggerEnter (Collider colisao)
{
    // Activar a animação abrir o carro caso o passageiro chegue ao taxi
    if (colisao.gameObject.name == "TAXI") {
        anim.SetBool ("Andar", false);
        anim.SetBool ("AbrirCarro", true);
        AbrirPorta = true;
    }
    // Percorrer os destinos a volta do taxi ate ao Destino0 (Porta)
    if (colisao.gameObject.name == "DestinoNavi") {
        numeroDestino = 0;
        this.transform.LookAt (referencia.transform);
    } else if (colisao.gameObject.name == "DestinoNav2") {
        numeroDestino = 1;
        this.transform.LookAt (NavDestino [1].position);
    } else if (colisao.gameObject.name == "DestinoNav3") {
        numeroDestino = 2;
        this.transform.LookAt (NavDestino [2].position);
    } else if (colisao.gameObject.name == "DestinoNav4") {
        numeroDestino = 3;
        this.transform.LookAt (NavDestino [3].position);
    } else if (colisao.gameObject.name == "DestinoNav5") {
        numeroDestino = 4;
        this.transform.LookAt (NavDestino [4].position);
    }
}

```

*Código 12 – Detecção de colisão e ativação dos destinos a percorrer*

- Scripts

⇒ *DestinosAnim / DestinosTCPAnim*

Os scripts de destino *DestinosAnim* (Teclado) e *DestinosTCPAnim* (g.USBamp), associados ao objeto táxi, são os scripts principais, para as diferentes versões do jogo. Nestes scripts é onde se pode encontrar as tomadas de decisão relativas ao destino que o táxi tomará nos diferentes pontos de decisão, ou seja, nos cruzamentos existentes na cidade. Em cada cruzamento foi colocado estrategicamente um *GameObject* apenas com o componente *Collider*, devidamente numerado (Figura 53), para o táxi detetar a colisão e assim a direção escolhida pelo jogador. Sempre que o jogador embater num ponto de destino, o nome deste é guardado, isto serve para quando embater no próximo ponto de decisão, o táxi conhecer qual foi o destino anterior e conseguir tomar a rota definida pelo jogador (Código 13).

```
void VirarEsquerda ()
{
    if (DestinoActual == "Destino1" && DestinoAnterior == "Destino2") {
        NumeroDestino = 63;
    } else if (DestinoActual == "Destino2" && DestinoAnterior == "Destino3") {
        NumeroDestino = 1;
    } else if (DestinoActual == "Destino3" && DestinoAnterior == "Destino2") {
        NumeroDestino = 4;
    } else if (DestinoActual == "Destino3" && DestinoAnterior == "Destino4") {
        NumeroDestino = 12;
    } else if (DestinoActual == "Destino3" && DestinoAnterior == "Destino10") {
        NumeroDestino = 2;
    } else if (DestinoActual == "Destino3" && DestinoAnterior == "Destino12") {
        NumeroDestino = 10;
    } else if (DestinoActual == "Destino4" && DestinoAnterior == "Destino5") {
        NumeroDestino = 13;
    } else if (DestinoActual == "Destino4" && DestinoAnterior == "Destino13") {
        NumeroDestino = 3;
```

Código 13 – Funcionamento da tomada de decisão relativa ao destino do táxi

A direção que o táxi seguirá, pode ser escolhida pelo jogador através do teclado (versão para teclado, através do script *DestinosAnim*), ou através de estímulos visuais (versão para g.USBamp, através do script *DestinosTCPAnim*). Esta rota poderá ser definida quando o táxi estiver a cerca de cinquenta metros do ponto de decisão, mostrando no ecrã as setas direcionais com as respetivas teclas para a direção (Figura 56 e Figura 57),

ou apresentando os estímulos visuais piscando a diferentes frequências (Figura 58 e Figura 59). No caso do script *DestinosTCPAnim*, aquando do aparecimento dos estímulos visuais, também é enviado para o MatLab um comando via TCP/IP para informar que os estímulos estão ativos para o jogador, e assim poder começar a deteção de frequência através do g.USBamp (Figura 59).



Figura 56 – Escolha de destinos através do teclado

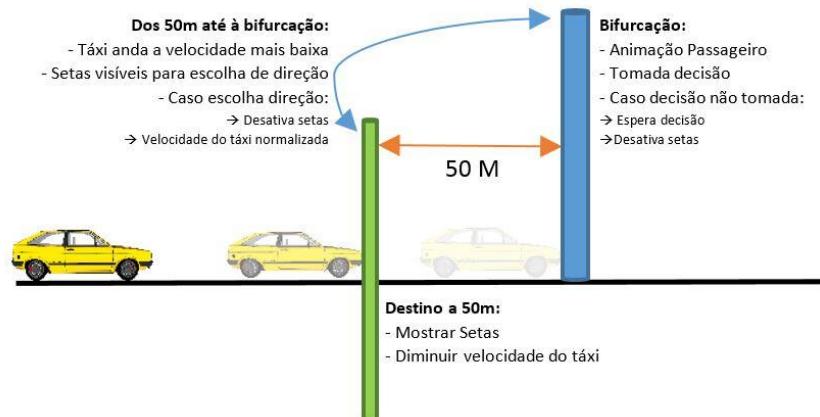


Figura 57 - Procedimento da escolha de direção para a versão teclado

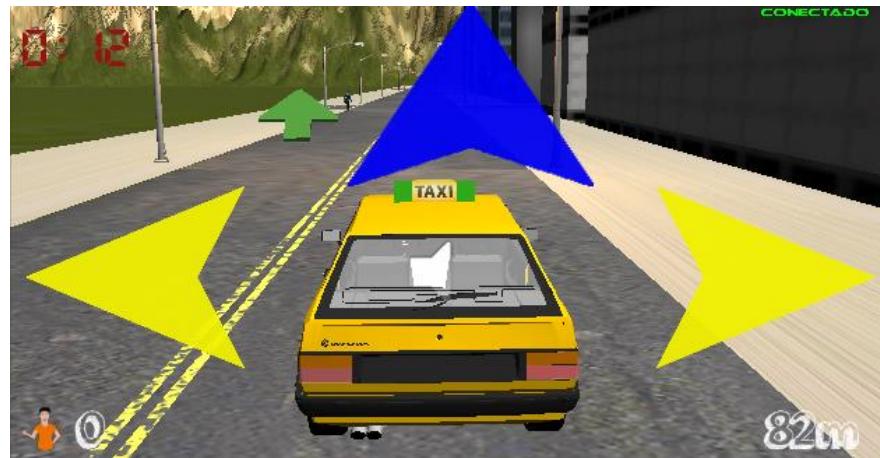


Figura 58 – Escolha de destino através de estímulos visuais

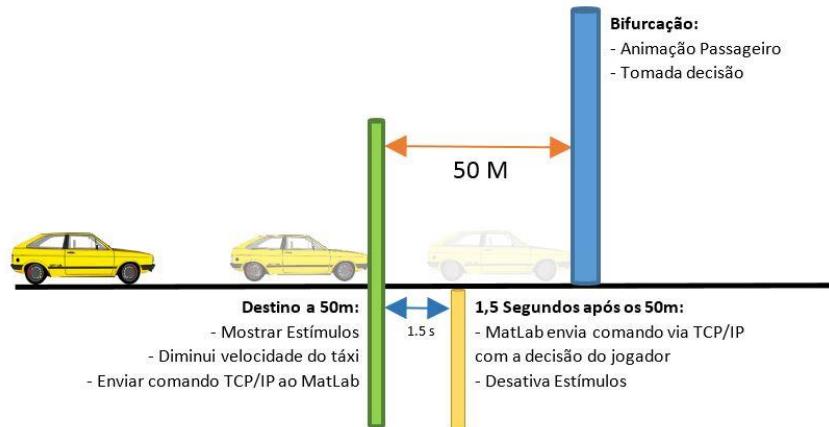


Figura 59 – Procedimento da escolha de direção para a versão com os estímulos

Estes scripts não indicam apenas os destinos que o táxi deve percorrer, também servem para contar o tempo de jogo, sistema de pontuação, definição da seta orientadora (Código 14), distância até ao passageiro ou destino (Código 14), bem como apanhar e largar os passageiros.

```

if (viajanteABordo == false) {
    // indica a distancia entre o taxi e o passageiro
    distanciaViaj = (int)Vector3.Distance (this.transform.position, Viajantes [NumeroViajante].transform.position);
    // actualizacao da seta orientadora, neste caso, para o passageiro a ir apanhar
    Setas.transform.LookAt (Viajantes [NumeroViajante].transform);
    // Mudar a cor da placa em cima do taxi, para avisar se ocupado ou livre
    GameObject.Find ("EstadoDir").renderer.material = TaxiLivre;
    GameObject.Find ("EstadoEsq").renderer.material = TaxiLivre;
} else {
    // indica a distancia entre o taxi e o destino do passageiro
    distanciaViaj = (int)Vector3.Distance (this.transform.position, ParticulasDestino [NumeroViajante].transform.position);
    // actualizacao da seta orientadora, neste caso, para o destino do passageiro
    Setas.transform.LookAt (ParticulasDestino [NumeroViajante].transform);
    // Mudar a cor da placa em cima do taxi, para avisar se ocupado ou livre
    GameObject.Find ("EstadoDir").renderer.material = TaxiOcupado;
    GameObject.Find ("EstadoEsq").renderer.material = TaxiOcupado;
}

```

*Código 14 – Definição seta orientadora, distâncias e mudança de estado de ocupação do táxi*

Para apanhar e largar os passageiros, os viajantes e os destinos foram definidos estrategicamente nos pontos de bifurcação (cruzamentos da cidade), ou seja, cada um destes intervenientes está inserido dentro de um destino *NavMesh*, que servirá para quando o táxi colidir com esse destino e o passageiro ainda não tenha sido transportado, o táxi aguarda pela entrada do viajante (Código 15).

```

// ----- VIAJANTE 1 -----
// Caso o destino em que o taxi bateu o local do passageiro em questao e este esteja ativo
if (collision.gameObject.name == "Destino40" & & Viajantes [1].activeSelf == true) {
    // preparar animação de andamento do passageiro
    AndarViajante = true;
    // Desabilita a componente NavMesh, para o carro ficar imóvel até o passageiro entrar
    this.gameObject.GetComponent<NavMeshAgent> ().enabled = false;
}

// Caso o taxi colida com o local de destino, e o numero de passageiro for o destinado a este local
if (collision.gameObject.name == "Destino46" & & NumeroViajante == 1) {
    LargarViajante (NumeroViajante);
}

```

*Código 15 – Apanhar e largar um passageiro*

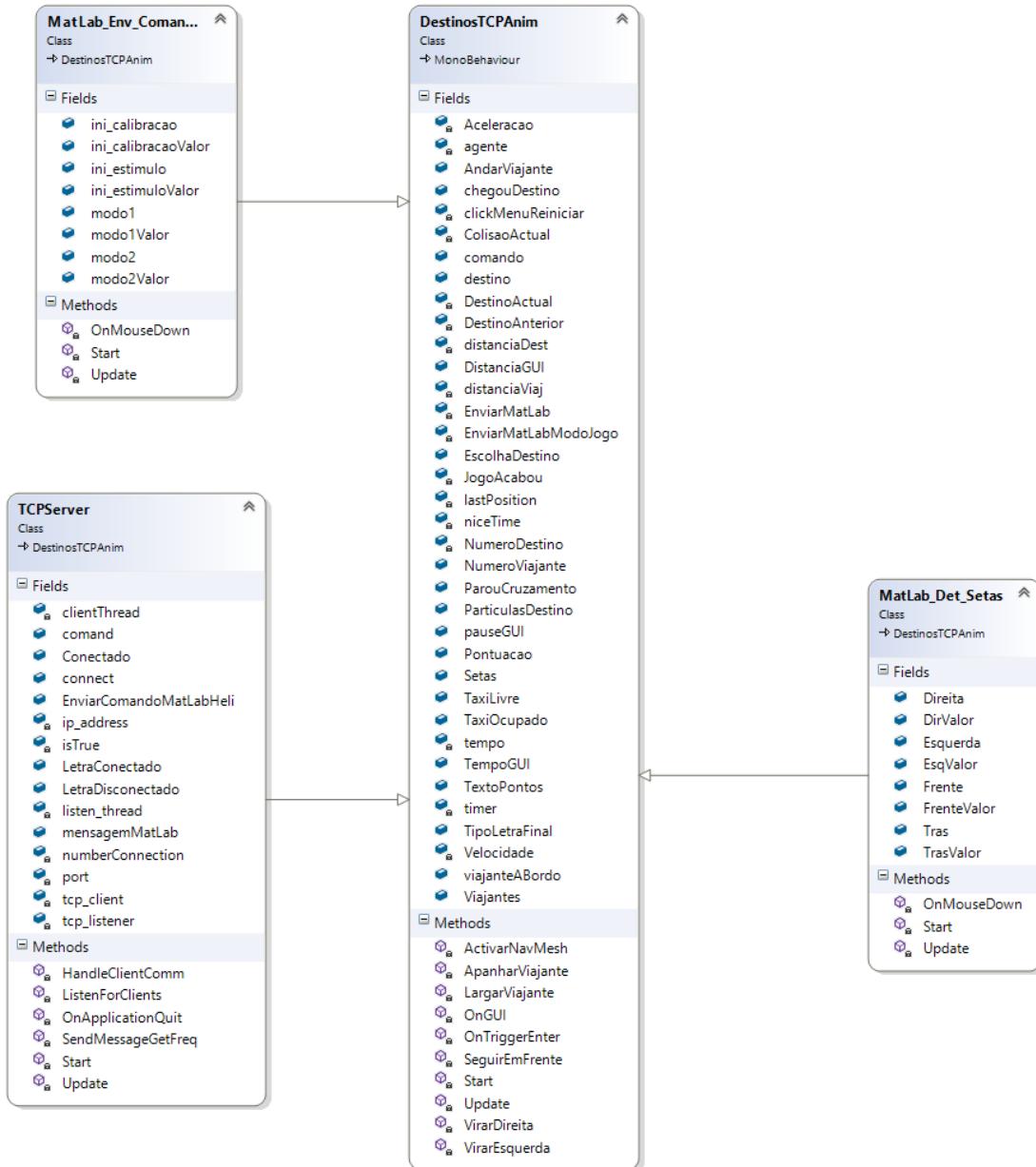


Figura 60 – Diagrama de classes que controlam movimento do táxi

⇒ *PauseGame*

Este script *PauseGame* tem o intuito de interromper o jogo sempre que o jogador quiser, carregando na tecla *Escape* do teclado.

Sempre que o jogador pressiona a tecla em cima referenciada, o jogo ficará imóvel (Código 16), e aparecerá um pequeno menu (Figura 61) onde o jogador poderá escolher se pretende reiniciar ou sair do jogo, sendo possível voltar ao mesmo pressionando novamente a tecla *Escape*.

```

        if (Input.GetKeyUp (KeyCode.Escape)) {
            if (pause == true) {
                // Interromper os sons do jogo
                if (Taxi3DTecladoScene || Taxi3DGusBampScene) {
                    GameObject.Find ("TAXI").audio.Play ();
                }
                Time.timeScale = 1.0f;

                pause = false;

            } else if (pause == false) {
                // Colocar novamente os sons no jogo
                if (Taxi3DTecladoScene || Taxi3DGusBampScene) {
                    GameObject.Find ("TAXI").audio.Pause ();
                }
                Time.timeScale = 0.0f;
                pause = true;
            }
        }
    }
}

```

*Código 16 – Colocar o jogo em pausa*



*Figura 61 – Menu de Pausa*

### 3.3.2.2 Implementação

Depois de construída toda a nossa cidade e desenvolvidos todos os *Scripts* e objetos necessários para o seu correto funcionamento obtemos o seguinte resultado final deste nível dos Carros3D. Através das seguintes figuras podemos verificar as diferentes situações de jogo, como por exemplo o momento em que o táxi apanha um viajante (Figura 63) ou o local de destino de um viajante (Figura 64).



Figura 62 – Início jogo carros3D



Figura 63 – Táxi apanha primeiro viajante



Figura 64 – Destino primeiro viajante

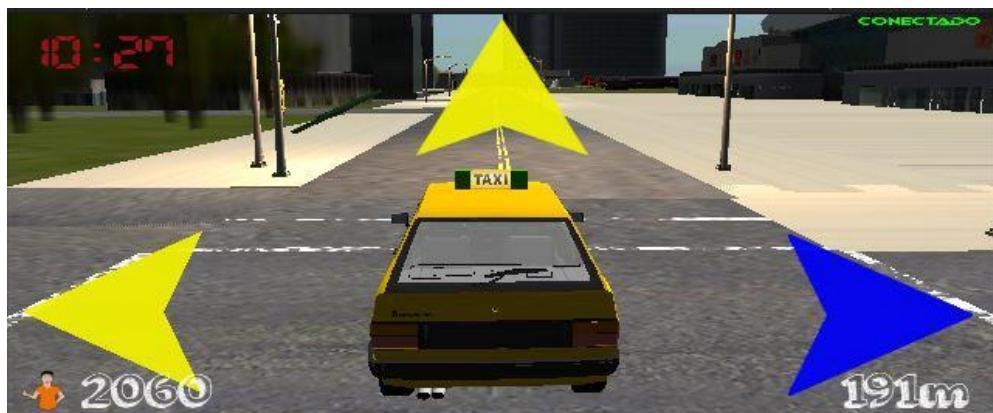
Quando o táxi se encontra livre, podemos ver as suas luzes no tejadilho verdes (Figura 65), significa que a próxima missão do utilizador é encontrar o próximo viajante. Quando este entra dentro do táxi as luzes passam a vermelho, significando que o objetivo passa a ser deixar o viajante no seu destino (Figura 64).



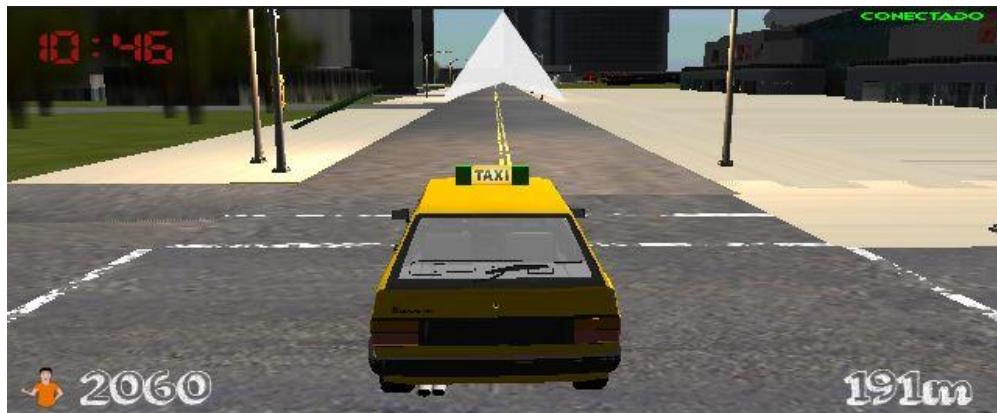
*Figura 65 – Apanha terceiro viajante*

Nas informações que nos são dadas podemos observar que ao largar um viajante no local desejado a nossa pontuação aumenta (canto inferior esquerdo do ecrã). Existem algumas ajudas para os utilizadores se deslocarem pela cidade e direcionarem para os destinos pedidos. No canto inferior direito podemos ver a distância a que nos encontramos do nosso destino, assim como a seta por cima do táxi aponta para a direção que devemos tomar, no caso da Figura 62 ou Figura 65 a seta aponta para o viajante pois estando o táxi livre o objetivo é apanhá-lo. Existe ainda no canto superior esquerdo o tempo de jogo decorrido.

Na Figura 66 podemos observar os estímulos a aparecer no ecrã para o utilizador decidir a direcção que deseja tomar. Já na Figura 67 observa-mos através da seta branca a direcção escolhido pelo utilizador.



*Figura 66 – Estímulos para decisão da direcção*



*Figura 67 – Indicação do estímulo detectado*

### 3.3.3 Helicóptero 3D

Neste nível de jogo o conceito é um pouco diferente, assim como o seu objetivo, tendo o jogador uma perspetiva diferente do cenário, onde poderá explorar uma enorme cidade com grandes edifícios.

O jogo do helicóptero consiste num cenário 3D, onde o utilizador controla um helicóptero a sobrevoar uma cidade. Ao longe poderão ser vistas moedas, sendo o objetivo do jogador apanhar as 10 moedas existentes, sendo que ao apanhar todas as moedas, o jogo é concluído com sucesso.

Para controlar o helicóptero é possível utilizar o teclado ou o sistema g.USBamp. Para o controlo com o sistema da g.tec, são mostradas setas indicativas com estímulos visuais a diferentes frequências onde o utilizador decide se deseja ir para a esquerda ou para a direita. Neste primeiro nível do jogo do helicóptero existem algumas rotas pré-definidas e a altitude a que o helicóptero se desloca é constante, isto para facilitar o jogador a ambientar-se com este modo de controlo através de estímulos visuais.

Num nível mais avançado do jogo, o jogador controla o helicóptero por si próprio sem que este siga qualquer rota já pré-definida. O objetivo é que o jogador consiga fazer com que o helicóptero passe dentro dos 10 anéis que estão suspensos no ar, ao consegui-lo é indicado quantos já consegui ultrapassar dos existentes, caso contrário terá de voltar atrás para tentar novamente. Quando ultrapassados todos os anéis, este nível é concluído com sucesso. Os controlos neste nível são efetuados da mesma forma, apenas acresce um estímulo visual, opção do jogador mover o helicóptero para cima.

Este jogo poderá ser jogado através de PC ou para a plataforma móvel Android. Ainda que não tenho sido testado o jogo encontra-se preparado para ser jogado em dispositivos com plataforma iOS.

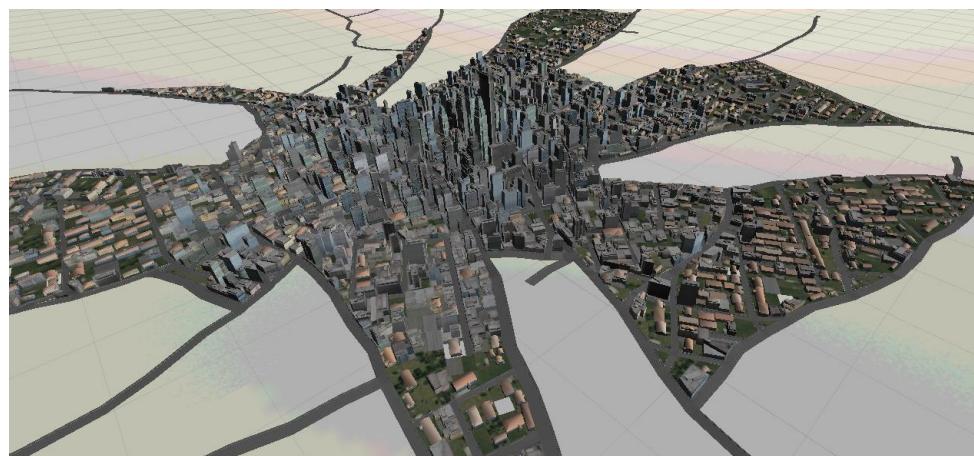
### 3.3.3.1 Arquitetura do jogo

No jogo 3D do helicóptero, foram utilizados vários modelos que estão disponíveis para descarregar de modo gratuito na plataforma de modelos 3D TF3DM [35], bem como a cidade que foi descarregada também de modo gratuito através da Asset Store do Unity [36].

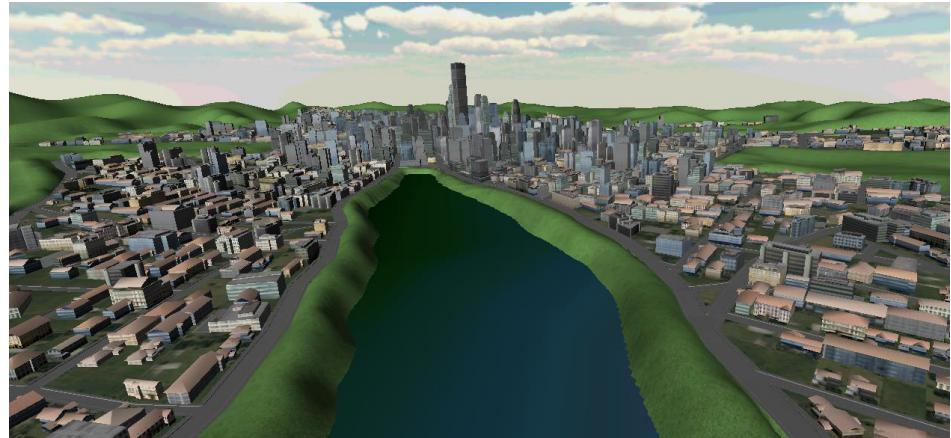
A arquitetura deste jogo é constituída pelos seguintes elementos que iremos descrever:

- Cidade

A cidade é um objeto que foi adquirido através da Asset Store do Unity. Ao fazer o download deste Asset, apenas precisamos de o importar para o nosso projeto. Ao importar o modelo da cidade, ficámos com a arquitetura da mesma disponível (Figura 68), sendo posteriormente necessário algumas alterações para a completar, como o alinhamento do terreno com as margens da cidade, a textura do terreno, bem como a adição de água, ficando assim com um modelo de cidade visivelmente mais elegante (Figura 69).



*Figura 68 – Arquitetura original da cidade*



*Figura 69 – Arquitetura final da cidade*

- Helicóptero

O helicóptero (Figura 70) é o objeto que o jogador controlará para atingir os objetivos do jogo, e foi descarregado através da plataforma TF3DM[35].



*Figura 70 – Componentes do helicóptero*

A este objeto foi adicionado um script denominado *RodarHelices*, com o intuito de rodar a hélice traseira e a hélice de cima. Além deste script foi adicionado um *Rigidbody*, um *Audio Source*, que serve para reproduzir o som do helicóptero, um *NavMeshAgent* (Figura 71) (no caso do jogo em modo 1, ou seja, destinos definidos e definição nas bifurcações), para o helicóptero se deslocar para os destinos previamente definidos na cidade (Figura 72). Também foram adicionados dois scripts denominados *DestinosHelicoptero* e *PauseGame*.



Figura 71 – NavMesh do jogo helicóptero

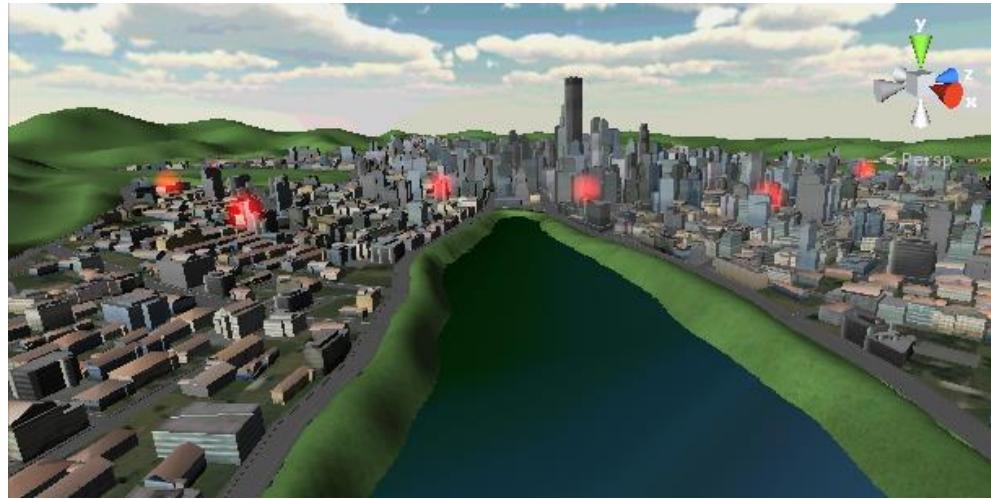


Figura 72 – Destinos Navmesh do jogo helicóptero

- Moedas

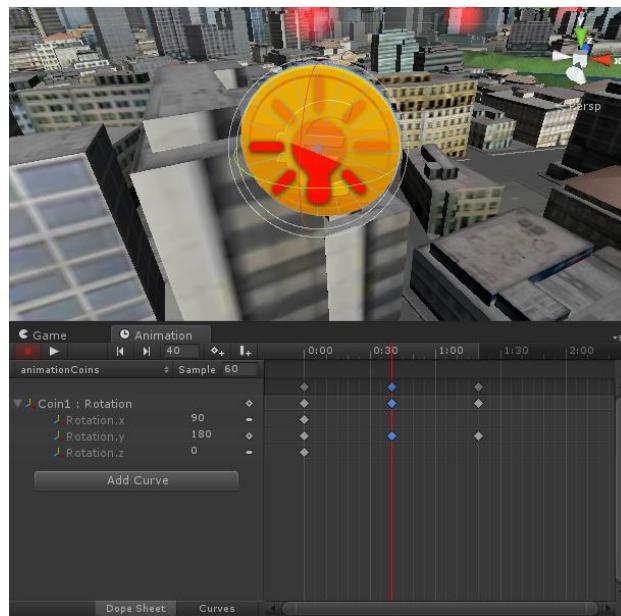
No nível um do jogo do helicóptero, o objetivo do jogo será encontrar e apanhar as moedas espalhadas no cenário do jogo. Este modelo descarregado através da plataforma TF3DM [35], e foi moldado de acordo com as exigências pretendidas. Para além das componentes que este objeto possuía, foram adicionados os componentes *Capsule Collider*, que serve para detetar uma colisão, a componente *Animator*, que irá fazer com que a moeda esteja sempre a rodar, um script denominado *ApanharMoedas*, que tem o intuito de

dar pontuação e apresentar um som sempre que haja uma colisão entre o helicóptero e a moeda, e uma luz vermelha do tipo *Point*, que servirá para o jogador conseguir identificar a moeda a uma longa distância (Figura 73).



*Figura 73 – Visualização das moedas do jogo*

A animação foi criada através da janela Animation, onde foi criado um novo clipe de animação. Para isto tipo de animação, foi selecionado a opção de rotação, para através do eixo dos Ys rodar a moeda. A animação é definida com a rotação a zero graus quando o tempo é zero, cento e oitenta graus quando o tempo é quarenta, e trezentos e sessenta graus quando o tempo é oitenta, fazendo sempre um ciclo, como mostra a Figura 74.



*Figura 74 – Animação da moeda*

Todas as moedas estão armazenadas dentro de um GameObject denominado Moedas, que terá a si associado um script chamado *EscolherMoedas* que servirá para escolher aleatoriamente quais as moedas a ser mostradas no jogo.

- Anéis

Os anéis foram inseridos do segundo nível do jogo do helicóptero, onde o objetivo será a passagem do helicóptero pelo interior destes anéis.

Estes modelos foram descarregados através da plataforma TF3DM[35], e foram adicionados alguns efeitos de luz, uma *Sphere Collider* e um script denominado *PassarAneis*. Os efeitos de luz adicionados servem para o jogador conseguir identificar mais facilmente para onde terá de se deslocar durante o jogo (Figura 75). A *Sphere Collider* é um componente de física do Unity, que neste caso serve para detetar quando o helicóptero colide com o interior de um anel.

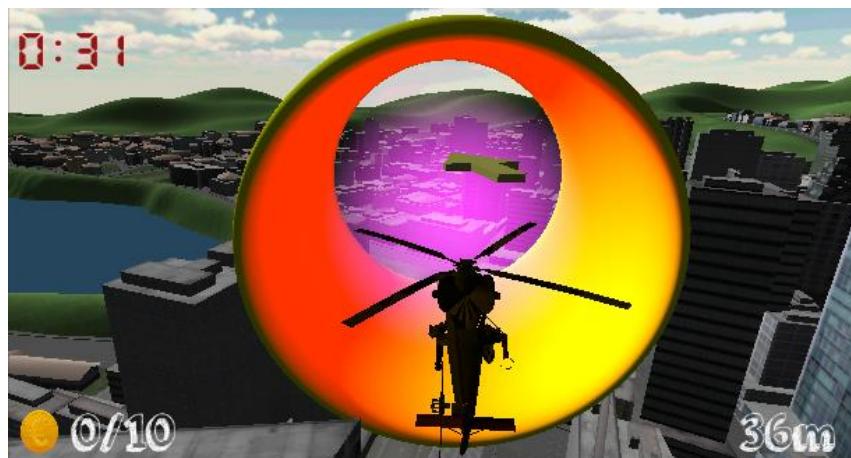


Figura 75 – Anéis do nível II do jogo 3D do helicóptero

Existem dezasseis anéis disponíveis espalhados na cidade, onde apenas oito são escolhidos aleatoriamente para o utilizador ultrapassar. Este processo é feito através da escolha aleatória de oito números em dezasseis, sendo depois esta lista associada ao número do Anel, sob a variável publica do tipo GameObject Aneis. Este processo é feito através do script *EscolherAneis*, que está associado ao GameObject Aneis.

- Scripts

⇒ *DestinosHelicóptero / DestinoEstimulos*

Estes dois scripts são os principais do nível 1 do jogo do helicóptero, sendo o *DestinosHelicóptero* para a versão de teclado, e *DestinoEstimulos* para a versão g.USBamp. Tal como nos scripts de destino do jogo do táxi, estes servem para tomar as decisões nos diferentes pontos de destino espalhados pelo cenário de jogo, também devidamente numerados (Figura 72). Sempre que o jogador embater num ponto de destino, o nome do deste é guardado, isto serve para quando embater no próximo ponto de decisão, o táxi conhecer qual foi o destino anterior e conseguir tomar a rota definida pelo jogador (Código 17).

```
// ----- FUNCOES RELATIVAS A DIRECAO ESCOLHIDA -----
void VirarDireita ()
{
    if (DestinoActual == "Destino1" && DestinoAnterior == "Destino20") {
        NumeroDestino = 3;
    }
    if (DestinoActual == "Destino1" && DestinoAnterior == "Destino2") {
        NumeroDestino = 20;
    }
    if (DestinoActual == "Destino1" && DestinoAnterior == "Destino3") {
        NumeroDestino = 2;
    }
    if (DestinoActual == "Destino1" && DestinoAnterior == "Destino7") {
        NumeroDestino = 2;
    }

    if (DestinoActual == "Destino2" && DestinoAnterior == "Destino1") {
        NumeroDestino = 5;
    }
    if (DestinoActual == "Destino2" && DestinoAnterior == "Destino3") {
        NumeroDestino = 4;
    }
}
```

Código 17 - Funcionamento da tomada de decisão relativa ao destino do helicóptero

A direção que o helicóptero tomará, poderá ser escolhida através do teclado (versão para teclado, através do script *DestinosHelicóptero*), ou através de estímulos visuais a piscar a diferentes frequências (versão para g.USBamp, através do script *DestinosEstimulos*). Esta rota poderá ser escolhida quando o helicóptero estiver a cerca de cem metros do ponto de decisão, mostrando no ecrã as setas direcionais (Figura 76 e Figura 77) ou os estímulos (Figura 78 e Figura 79). No caso do script *DestinoEstimulos*, simultaneamente ao aparecimento dos estímulos visuais, também é enviado para o MatLab um comando via TCP/IP para informar que os

estímulos estão ativos para o jogador, e assim poder começar a deteção de frequência através do g.USBamp



Figura 76 – Escolha dos destinos através de teclado

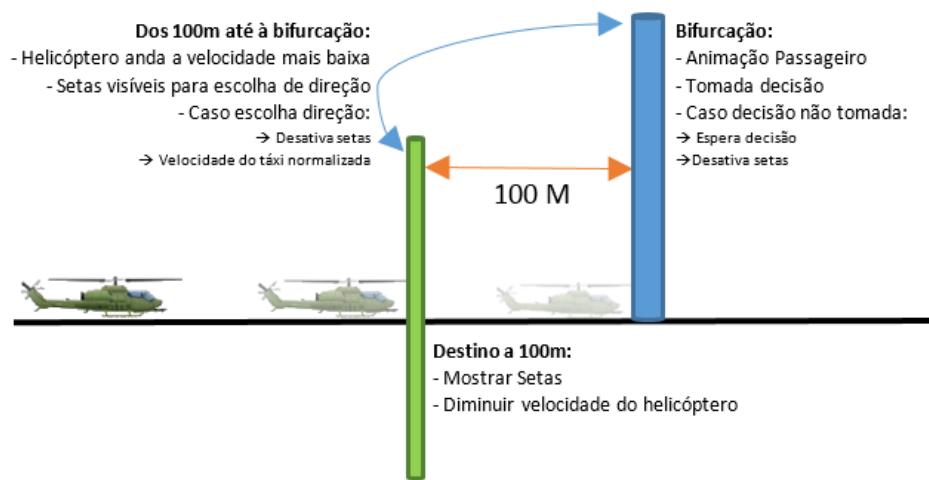


Figura 77 - Procedimento da escolha de direção para a versão teclado



Figura 78 – Escolha dos destinos através do g.USBamp

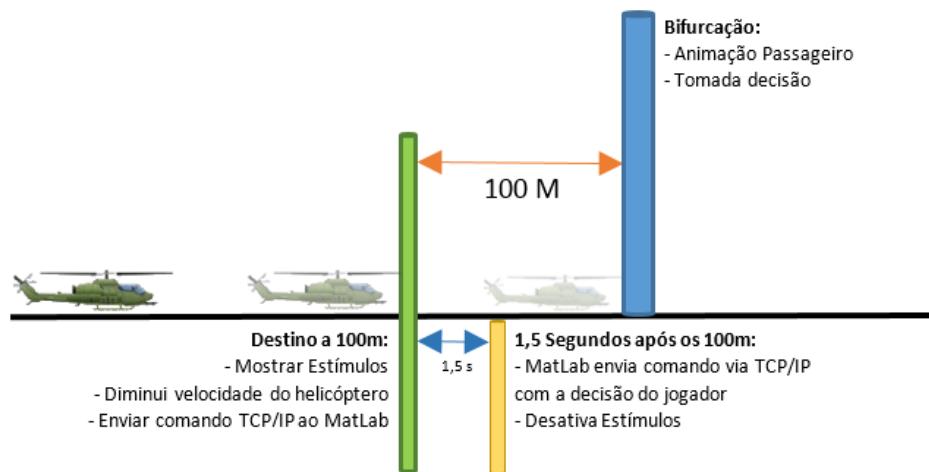
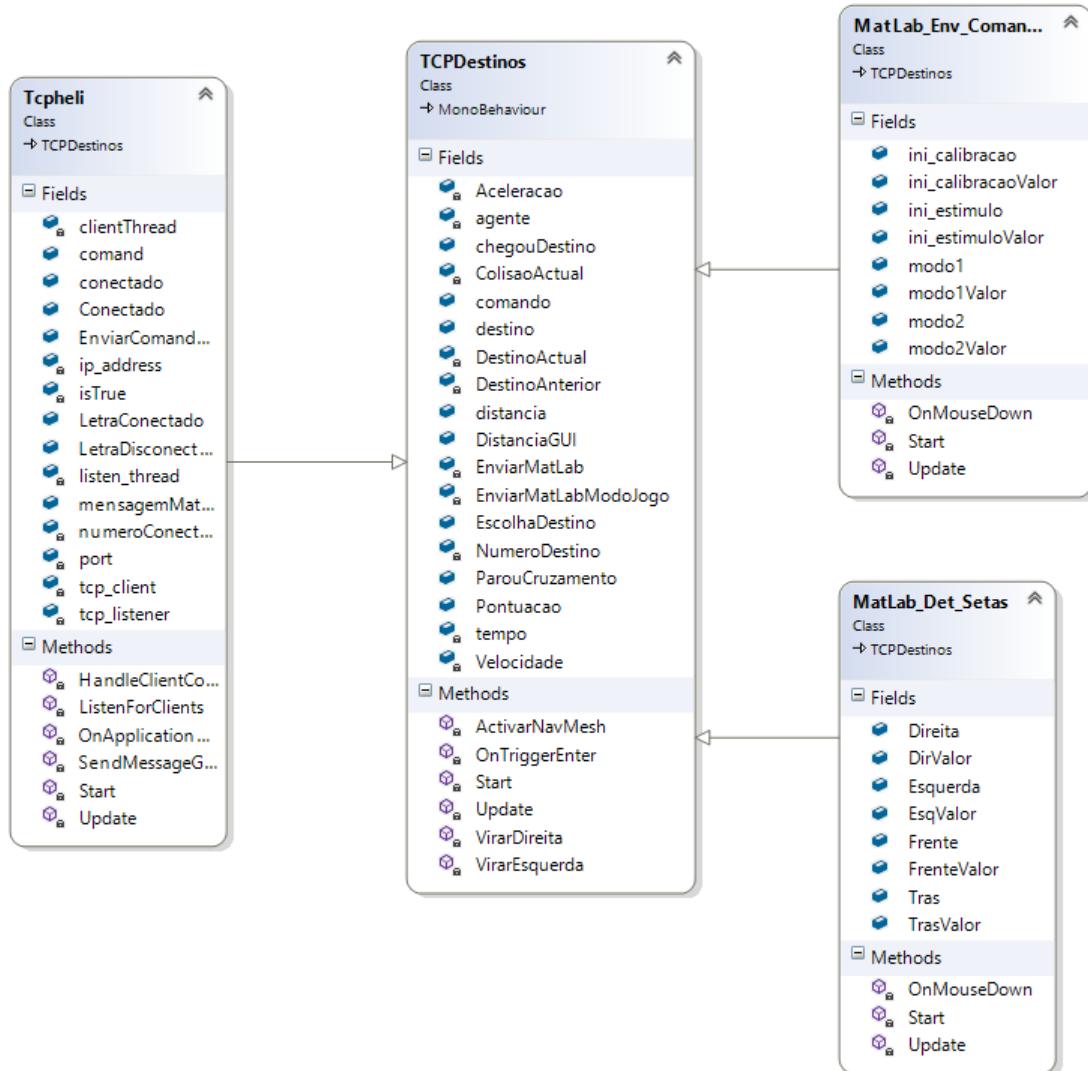


Figura 79 - Procedimento da escolha de direção para a versão g.USBamp



*Figura 80 – Diagrama de classes que controlam o movimento do helicóptero*

⇒ *TCPHeliLivre / HeliLivre*

Estes scripts relativos ao jogo do helicóptero, referentes ao nível 2 do jogo, têm o intuito de fazer com que o helicóptero esteja sempre a andar para a frente, só mudando de direção quando o utilizador entender, isto é, o helicóptero andará sozinho e o utilizador escolherá o caminho livremente com o objetivo de ultrapassar os anéis (Código 18 e Código 19). Este script também fará com que helicóptero mantenha sempre a posição aérea (eixo dos Ys), na mesma posição do anel que o jogador terá de ultrapassar (Código 20).

```

// Comando para seguir em frente
this.transform.Translate (new Vector3 (0, 0, 20 * Time.deltaTime));
// Virar a direita
if (Input.GetKey (KeyCode.RightArrow) || DireitaAndroid.DirAndroid == true) {
    this.transform.Rotate (new Vector3 (0, 10 * Time.deltaTime, 0));
}
// Virar a esquerda
if (Input.GetKey (KeyCode.LeftArrow) || EsquerdaAndroid.EsqAndroid == true) {
    this.transform.Rotate (new Vector3 (0, -10 * Time.deltaTime, 0));
}
// Subir o helicoptero (eixo dos yy)
if (Input.GetKey (KeyCode.UpArrow) || FrenteAndroid.FrnAndroid == true) {
    this.transform.Translate (new Vector3 (0, 2 * Time.deltaTime, 0));
}

```

*Código 18 - Escolha da direção através do teclado ou através de touch na versão para Android/iOS*

```

//Andar para a frente continuamente
this.transform.Translate (new Vector3 (0, 0, 5 * Time.deltaTime));
//Comando TCP para virar a direita
if (TcpPheli.comand.Equals (MatLab_Det_Setas.DirValor.ToString ())) {
    this.transform.Rotate (new Vector3 (0, 10 * Time.deltaTime, 0));
}
//Comando TCP para virar a esquerda
if (TcpPheli.comand.Equals (MatLab_Det_Setas.EsqValor.ToString ())) {
    this.transform.Rotate (new Vector3 (0, -10 * Time.deltaTime, 0));
}
//Comando TCP para o helicoptero apenas seguir em frente, sem virar alguma direção
if (TcpPheli.comand.Equals (MatLab_Det_Setas.FrenteValor.ToString ())) {
    this.transform.Translate (new Vector3 (0, 0, 0));
}

```

*Código 19 - Escolha da direção através da escolha visual, via comando TCP/IP enviado pelo MatLab*

```

// Colocar posição Y do helicoptero, igual a posição Y do anel, caso esteja baixo demais
if (this.gameObject.transform.position.y < EscolherAneisTeclado.PosicaoYAneis) {
    this.transform.Translate (new Vector3 (0, 2 * Time.deltaTime, 0));
}
// Colocar posição Y do helicoptero, igual a posição Y do anel, caso esteja alto demais
if (this.gameObject.transform.position.y > EscolherAneis.PosicaoYAneis) {
    this.transform.Translate (new Vector3 (0, -2 * Time.deltaTime, 0));
}

```

*Código 20 - Colocar altitude do helicóptero correta em relação a altitude do anel*

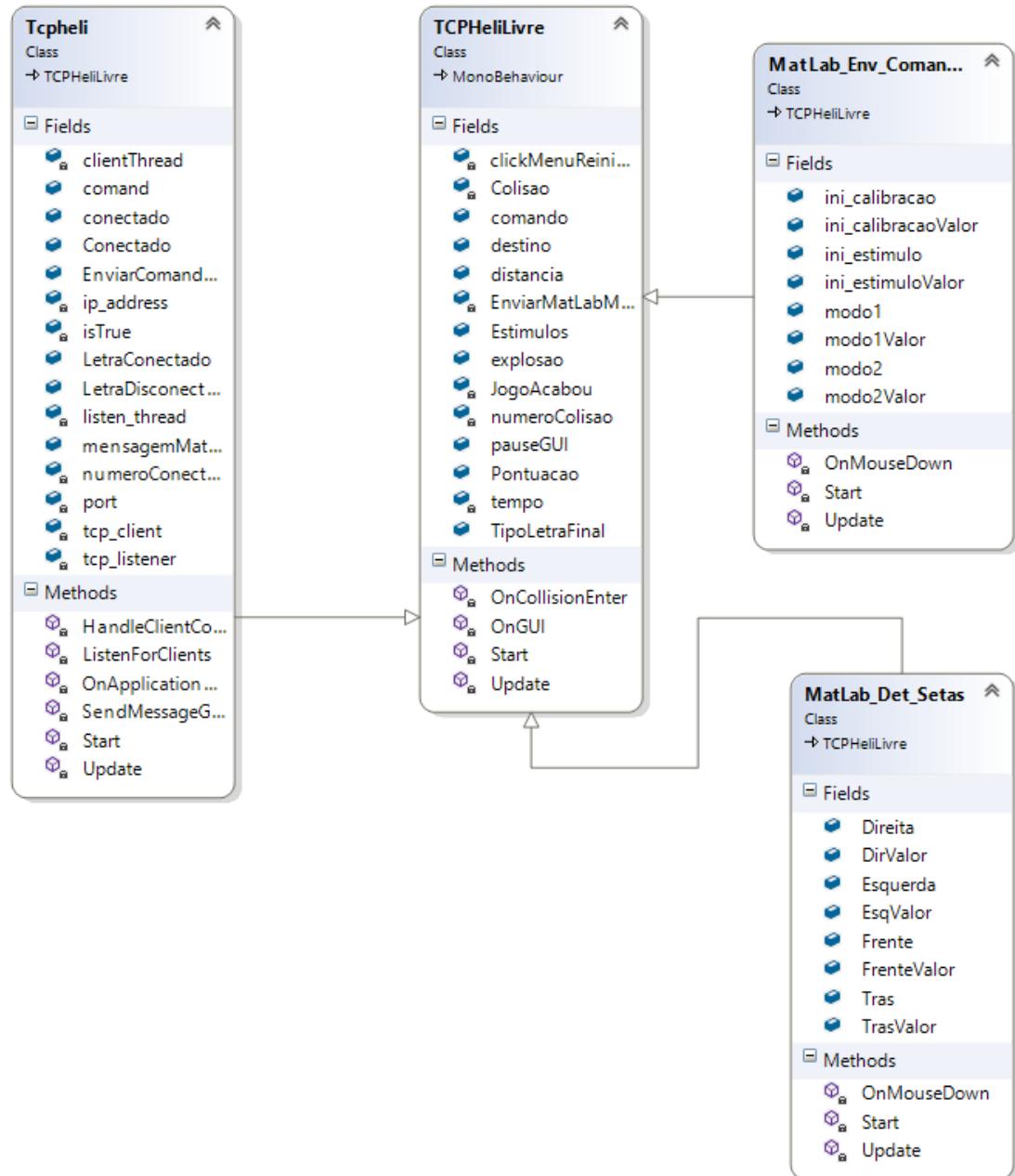


Figura 81 – Diagrama de classes que controlam o movimento do helicóptero no modo livre

⇒ ApanharMoedas

Este script é bastante simples, servindo apenas para detetar quando o jogador embate com a moeda, aumentando assim a pontuação e reproduzindo um som de bónus (Código 21).

```

public class ApanharMoedas : MonoBehaviour
{
    // variavel static para poder ser acessada nos script HeliLivre e TCPHeliLivre
    public static int Pontuacao= 0;

    void Start()
    {
        Pontuacao= 0;
    }
    // OnTriggerEnter() é onde é detetada a colisão
    void OnTriggerEnter (Collider collision)
    {
        // Reproduzir o som - esta na camera, devido a camera conter o AudioListener
        GameObject.Find ("Main Camera").audio.Play ();
        // Desativa a moeda
        this.gameObject.SetActive (false);
        // Aumenta a pontuação
        Pontuacao += 1;
    }
}

```

*Código 21 – Script ApanharMoedas*

⇒ *PassarAneis*

Tal como o script *ApanharMoedas*, este script fará exatamente o mesmo, mas este servirá para os anéis, que estarão disponíveis no nível 2 do jogo do helicóptero.

⇒ *EscolherMoedas / EscolherAneis*

Estes scripts de escolha servem para efetuar a escolha aleatória das moedas/anéis a aparecer no jogo. No caso das moedas, existem vinte moedas espalhadas pelo cenário de jogo, onde apenas dez serão escolhidas, já no caso dos anéis, são dezasseis anéis e apenas oito serão escolhidos. Estes scripts também servirão para mostrar a pontuação no ecrã, o tempo de jogo, atualização da seta orientadora, no caso do nível 2 do jogo do helicóptero, bem como verificação se o jogador já terminou o jogo, apanhando todas as moedas/anéis disponíveis (Figura 82).



*Figura 82 – Fim do jogo do helicóptero*

### 3.3.3.2 Implementação

Depois de construída toda a nossa cidade e desenvolvidos todos os *Scripts* e objetos necessários para o seu correto funcionamento obtemos o seguinte resultado final destes níveis do jogo Helicóptero. Através das seguintes figuras podemos verificar as diferentes situações de jogo.

Na Figura 83 temos o início do jogo, o helicóptero encontra-se num heliporto e quando a conexão com o Matlab é efetuada este começa a voar.



*Figura 83 – Início jogo nível I Helicóptero*

No ecrã de jogo podemos observar as várias informações necessárias para o utilizador, como o número de moedas apanhadas, o tempo de jogo ou a distância, em metros, até ao próximo ponto de decisão, assim como os estímulos visuais, isto para o jogo nível I do helicóptero (Figura 84).

Na situação de jogo ilustrada na Figura 85, onde o helicóptero se prepara para apanhar uma moeda, podemos observar que a decisão de tomada de direção já foi interpretada pelo sistema pois a seta passou a estar fixa e a branco, indicando assim a direção escolhida pelo utilizador.



*Figura 84 – Jogo helicóptero nível I*



*Figura 85 – Helicóptero apanha moeda*

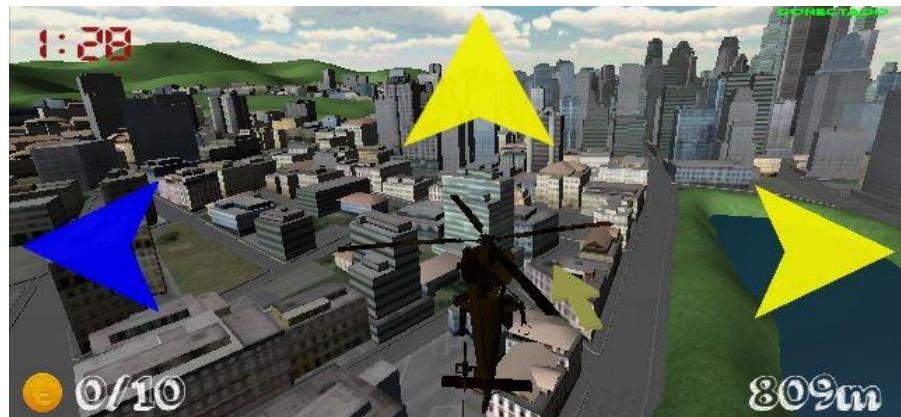


*Figura 86 – Helicóptero apanha segunda moeda*

No segundo nível do jogo do helicóptero Figura 87, continuamos a ter o mesmo cenário mas existem pequenas diferenças na jogabilidade.

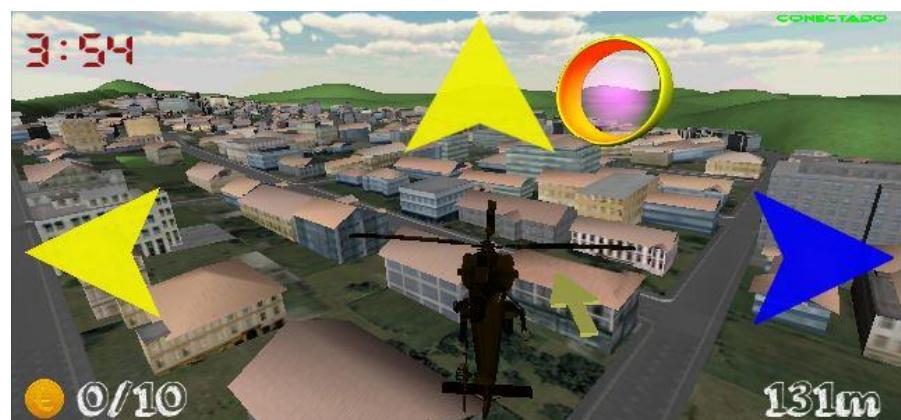
Neste nível os estímulos estão constantemente presentes no ecrã de jogo, pois as mudanças de direção podem ser a cada instante. Assim, este nível contém também

uma ajuda adicional tal como no jogo do táxi, uma seta indicadora para o utilizador saber a direção que tem de seguir.



*Figura 87 – Jogo helicóptero nível II*

Também o objetivo do jogo neste nível foi alterado, passando então o utilizador a ter que fazer o helicóptero passar por dentro dos diversos anéis suspensos no ar (Figura 88 e Figura 89).



*Figura 88 – Preparação para entrada do helicóptero no anel*



*Figura 89 – Helicóptero entra no anel*

### 3.3.4 Estímulos

Os estímulos visuais são uma das peças fundamentais para o correto funcionamento do nosso jogo, sem esta peça era impossível o controlo do jogo através do sistema g.USBamp. Isto porque, as frequências a que as setas piscam têm de ser exatamente precisas, e foi graças ao nosso algoritmo (Código 22), que conseguimos implementar os estímulos visuais no nosso projeto.

O Código 22 permite que o controlo da frequência com que os estímulos são apresentados ao jogador seja independente da taxa de refreshamento do jogo e da carga computacional do processador. Estas duas características permitem que os estímulos tenham o comportamento adequado para serem captados pelos dispositivos EEG e processados de forma a torná-los no input do jogo.

Estes estímulos são objetos independentes de todo o jogo, como tal existem *Scripts* a eles associados de modo a controlar as suas frequências. Para cada uma das setas direcionais existe um *Script* a ela associado, estes são:

- ⇒ DireitaEstimulo, EsquerdaEstimulo e FrenteEstímulo

O modo de funcionamento dos estímulos baseia-se em duas imagens sobrepostas com diferentes cores, fazendo assim um contraste que provoca o estímulo visual. Cada uma das setas fica ativa no ecrã durante um tempo estipulado, dependendo da frequência a que desejamos que estas pisquem.

Estas três classes que controlam as frequências de cada uma das setas (esquerda, direita e frente) (Figura 91), contêm uma *Thread* que controla a frequência com que uma imagem e outra ficam ativas. Esse tempo de *sleep* da *Thread* é o ponto chave do nosso algoritmo, a variável resto sincroniza o tempo que cada seta tem que ficar ativa (delay), com o *refresh rate* do sistema.

```

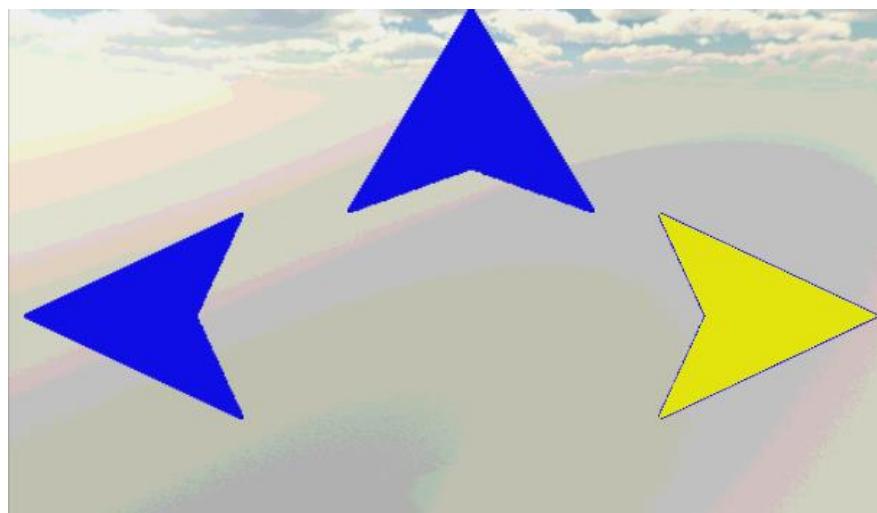
void Start ()
{
    /*
     * Verifica se a frequencia foi definida manualmente
     * ou por calibração
     */
    if(FreqEscolhe.DefFreqManual==true)
    {
        frequencia = FreqEscolhe.FreqFreteVal;
    } else if(Calibracao.DefFreqCalibrar==true){
        frequencia= Calibracao.FreqRecMeioFrete;
    }
    // Tempo activo de cada seta
    delay = (int) (1000 / frequencia)/2;
    //iniciar Thread
    ThreadBlink = new Thread (Blink);
    ThreadBlink.Start ();
}

void Blink ()
{
    while (true) {
        // Resto divisão entre o tempo actual e o delay
        int resto = DateTime.Now.Millisecond % delay;
        //Alternar estado da imagem (Activado ou desactivado)
        state = !state;
        // Desactivar thread durante o tempo de delay menos o resto
        Thread.Sleep (delay-resto);
    }
}

void Update ()
{
    if (state) {
        Setal1.enabled=true;
        Setal2.enabled=false;
    } else {
        Setal1.enabled=false;
        Setal2.enabled=true;
    }
}

```

*Código 22 – Controlo estímulos*



*Figura 90 – Estímulos*

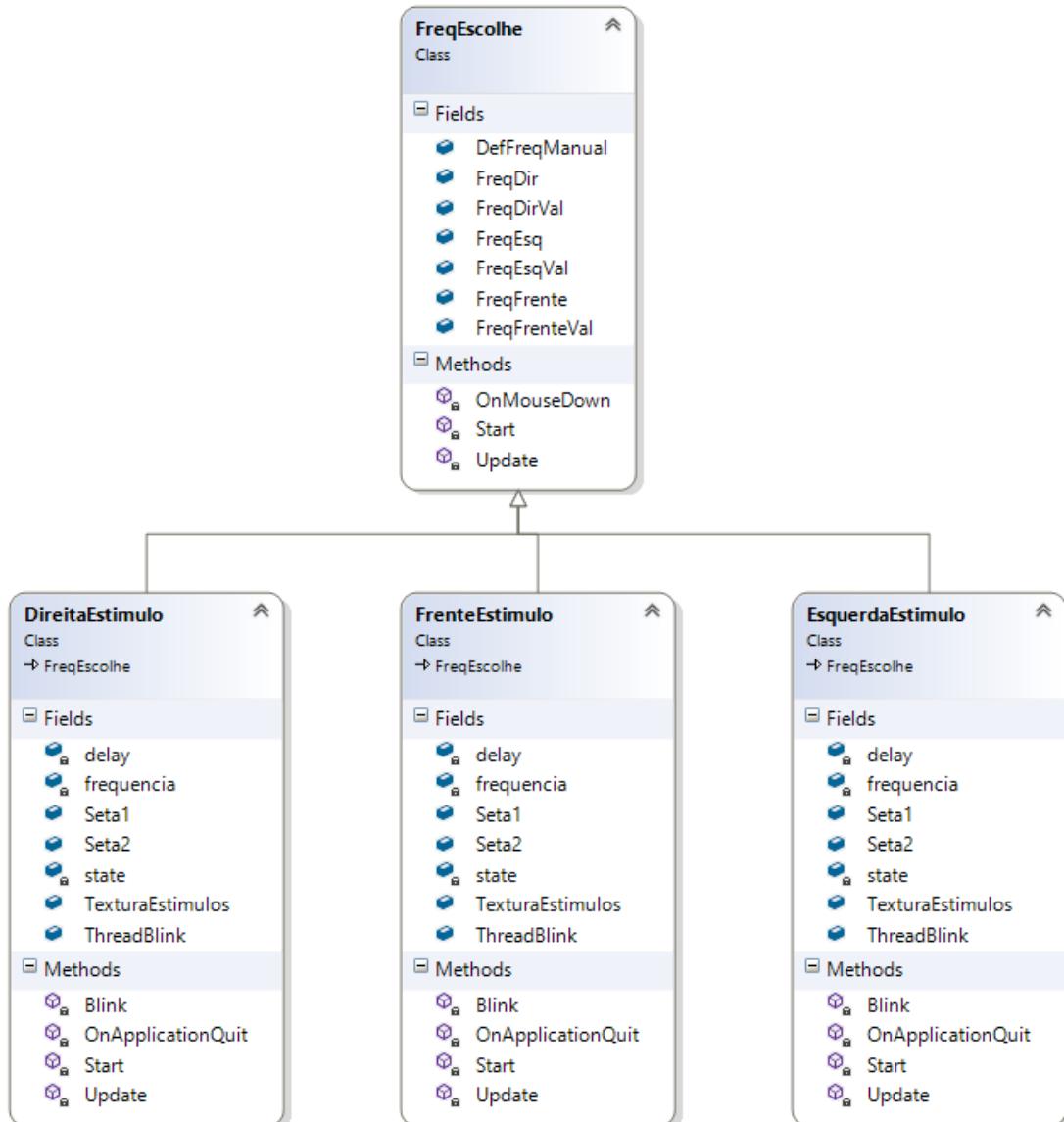


Figura 91 – Diagrama de classes que controlam os estímulos visuais

### 3.3.5 Setas direcionais

As setas direcionais são texturas que são apresentadas no ecrã sempre que o jogador tiver de escolher a direção a tomar. Existem três setas direcionais, a seta direita e esquerda, que servem para tomar a direção direita ou esquerda respetivamente, e a seta para cima, que no caso do jogo do táxi servirá para o jogador seguir em frente, e no caso do helicóptero servirá para fazer subir o helicóptero.

Cada uma destas texturas tem a si associado um script, para o jogador poder escolher a direção que deseja seguir através do toque na seta direcional pretendida, assim,

quando da compilação do jogo para plataformas móveis, este pode ser jogado através do toque nas texturas.

Nestes scripts também podemos encontrar a verificação da plataforma em que o jogo está a correr, no caso de PC, Android ou iOS (Código 23), que serve para apresentar as opções de direção no ecrã de diferentes modos. No caso do PC é apresentado as setas com a tecla a pressionar (Figura 92), nas plataformas móveis apenas teremos as setas (Figura 93). No caso do nível 2 do jogo do helicóptero para a versão móvel, as setas aparecerão estrategicamente colocadas para oferecer um melhor controlo sobre o helicóptero, já que na versão para PC as setas não existem, controlando o jogador o helicóptero através das setas direcionais do teclado (Figura 94 e Figura 95).

```
void Start () {
    if (Application.platform == RuntimePlatform.Android) {
        this.guiTexture.texture = Dir;
    } else if (Application.platform == RuntimePlatform.IPhonePlayer) {
        this.guiTexture.texture = Dir;
    }
}
```

*Código 23 – Verificação da plataforma em que o jogo está a ser executado*



*Figura 92 – Setas direcionais para a versão PC*

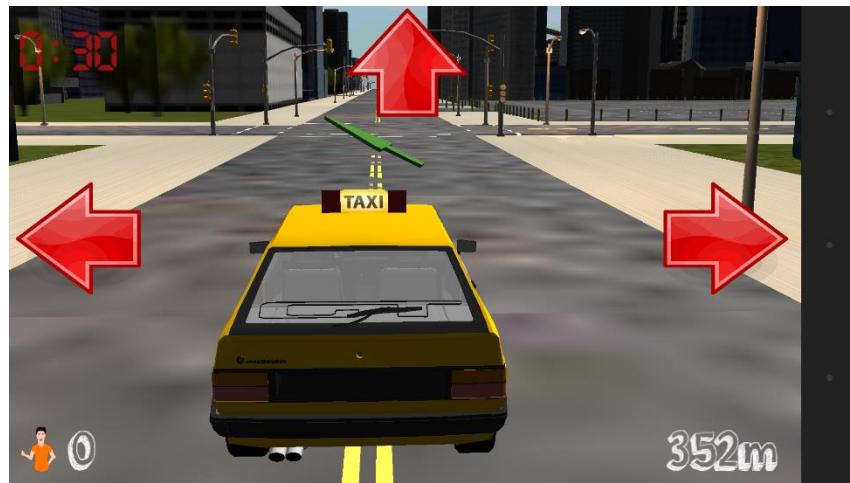


Figura 93 – Setas direcionais para a versão móvel



Figura 94 – Jogo helicóptero nível 2, sem setas direcionais



Figura 95 – Disposição das setas direcionais para a versão móvel

### 3.3.6 Animações

As animações são um dos componentes disponibilizados pelo Unity. Este componente tem o intuito de fazer com que personagens humanas, ou mesmo simples objetos, ganhem vida no ambiente de jogo, efetuando movimentos naturais e fluidos.

Neste projeto do DriveByMind também foram utilizadas algumas animações, descarregadas gratuitamente através da plataforma Mixamo[34], nomeadamente no jogo do táxi. A animação apresentada neste jogo é aplicada ao passageiro que espera o táxi. São três as animações colocadas nesta personagem, sendo elas quando o passageiro espera o táxi, levantando o braço para ter a sensação de que estar a chamar o táxi, a ida ao encontro do veículo a andar, e de seguida a abertura da porta do lado do passageiro.

Estas animações para funcionarem corretamente terão de estar associadas a um componente *AnimatorController*. Será neste *AnimatorController* que será feita a gestão da animação, funcionando como uma máquina de estados (Figura 96).

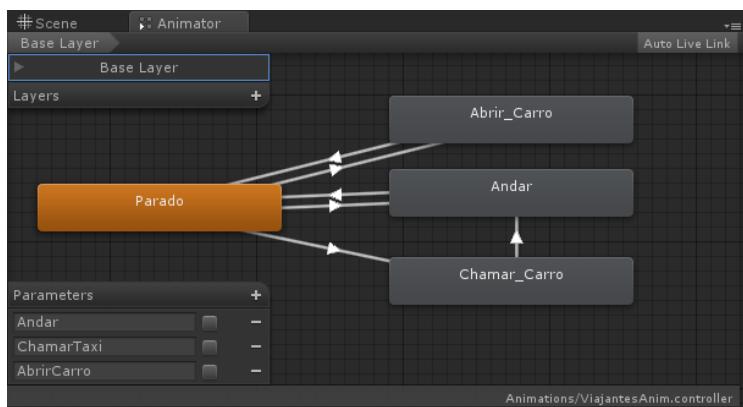


Figura 96 - Animação passageiro no jogo Táxi3D

Como podemos verificar na figura acima representada, existem quatro estados, o estado “Parado”, que está a cor diferente devido a ser o estado principal, e os estados “Chamar\_Carro”, “Andar”, e “Abrir\_Carro”. As ligações entre estados contêm os estados necessários para haver a transição entre animações, seguindo a rota para onde a seta da ligação aponta. Estes estados das ligações são definidos através dos parâmetros (canto inferior esquerdo da Figura 96), que neste caso são variáveis do tipo boolean (verdadeiro e falso) para controlar o estado da transição da animação. Na Figura 97 está representada a condição para mudança do estado entre a animação

“Parado” e “Chamar\_Carro” As variáveis poderão ser do tipo Boolean, Integer, Float e Trigger (característica de uma BoxCollider).

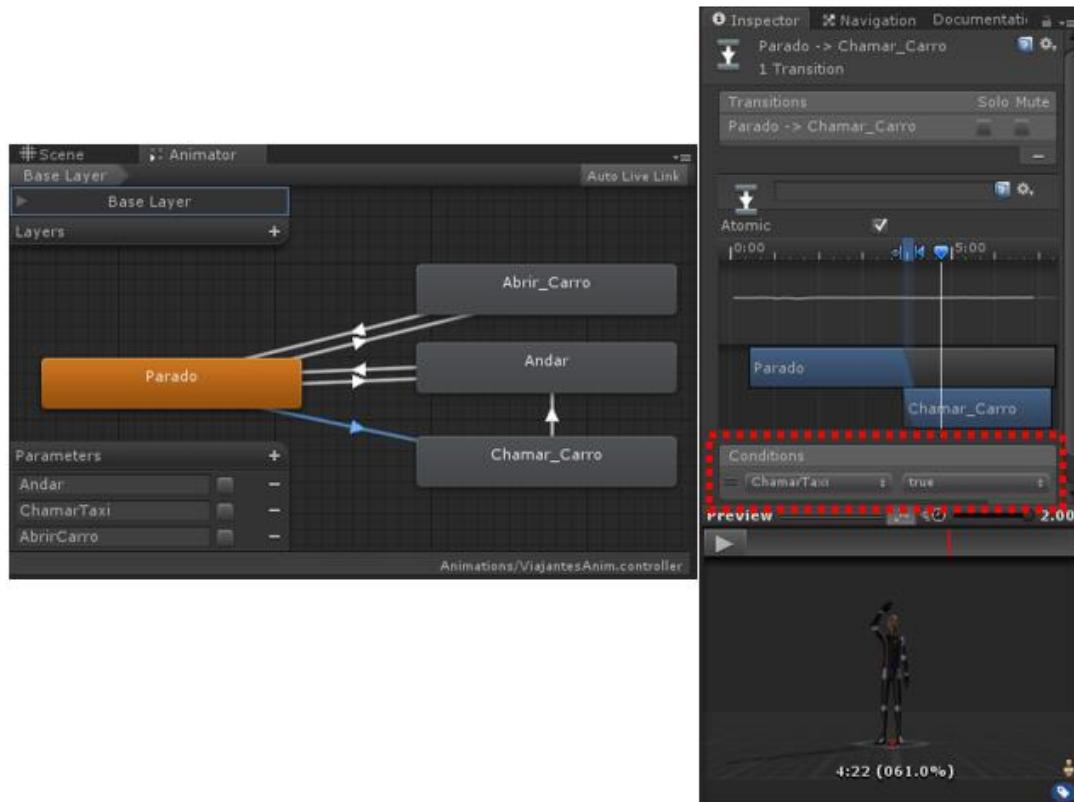


Figura 97 - Transição relativa à animação “Parado” e “Chamar\_Carro”

Depois de escolhidos os parâmetros para a transição entre animações, foi desenvolvido um script (Código 24), para as variáveis introduzidas nos parâmetros puderem ser alteradas e assim haver uma transição entre animações. No caso do jogo do táxi, este script irá mostrar o passageiro parado até o veículo estar a cerca de cem metros, ao que nesse momento será alterada a variável referente a essa transição “ChamarTaxi”, para que o passageiro possa começar a nova animação. A animação “Andar” será chamada quando o táxi colidir com a BoxCollider do destino referente ao passageiro, sendo que também foi adicionado um componente NavMesh para o passageiro percorrer o táxi até a porta de entrada. A animação “Abrir\_Carro” ficará ativa quando o passageiro colidir com o táxi, pois ao percorrer todos os destinos a volta do táxi, este irá embater no veículo.

```

// Verificar se taxi embateu no destino
if (DestinosAnim.AndarViajante == true || DestinosTCPAnim.AndarViajante == true) {
    SeguirCarro = false; // Parar o carro
    anim.SetBool ("Andar", true); // Activar a AnimaçãoCarros "Andar"
    // Verificar qual é o destino a volta do taxi mais perto do passageiro, para não percorrer todos
    for (int i=0; i<6; i++) {
        distancia = (int)Vector3.Distance (this.transform.position, NavDestino [i].transform.position);
        if (distancia < distanciaPeq) {
            distanciaPeq = distancia;
            numeroDestino = i;
        }
    }
    // Desativar a opção de fazer o passageiro Andar
    DestinosAnim.AndarViajante = false;
    DestinosTCPAnim.AndarViajante = false;
}

// Percorrer os destinos relativos ao NavMesh
if (numeroDestino >= 0 && numeroDestino != 10) {
    agente = gameObject.GetComponent<NavMeshAgent> ();
    agente.SetDestination (NavDestino [numeroDestino].position);
}

// cálculo da distância entre o taxi e o destino
distancia = (int)Vector3.Distance (Taxi.transform.position, destino.transform.position);
// Rodar passageiro, para ficar de frente para o taxi
if (distancia > 30) { this.transform.LookAt (referencia.transform); }
// Caso a distância seja <100m, o passageiro levanta a mão para chamar o taxi
if (distancia > 2 && distancia < 100) { anim.SetBool ("ChamarTaxi", true); }
else { anim.SetBool ("ChamarTaxi", false); }
// AbrirPorta = true é definido quando o passageiro percorre todos os destinos
// a volta do carro e chega ao destino 0
if (AbrirPorta == true) {
    time += Time.deltaTime; // contador de tempo
    if (time > 2.1 && time < 2.9) {
        RotacaoPorta = -40 * Time.deltaTime; // Abrir a Porta
        porta.transform.Rotate (new Vector3 (0, RotacaoPorta, 0));
    } else if (time > 3.1 && time < 3.29) {
        RotacaoPorta = 170 * Time.deltaTime; // Fechar a Porta
        porta.transform.Rotate (new Vector3 (0, RotacaoPorta, 0));
        entrouNoCarro = true;
    } else if (AbrirPorta == true && entrouNoCarro == true) {
        // Colocar a porta na posição correta, e mandar seguir o carro.
        porta.transform.localEulerAngles = new Vector3 (0, 0, 0);
        SeguirCarro = true;
        AbrirPorta = false;
        AtualizarDestino = true;
    }
}
void OnTriggerEnter (Collider colisao)
{
    // Activar a animação abrir o carro caso o passageiro chegue ao taxi
    if (colisao.gameObject.name == "TAXI") {
        anim.SetBool ("Andar", false);
        anim.SetBool ("AbrirCarro", true);
        AbrirPorta = true;
    }
    // Percorrer os destinos a volta do taxi até ao Destino0 (Porta)
    if (colisao.gameObject.name == "DestinoNav1") {
        numeroDestino = 0;
        this.transform.LookAt (referencia.transform);
    } else if (colisao.gameObject.name == "DestinoNav2") {
        numeroDestino = 1;
        this.transform.LookAt (NavDestino [1].position);
    } else if (colisao.gameObject.name == "DestinoNav3") {
        numeroDestino = 2;
        this.transform.LookAt (NavDestino [2].position);
    } else if (colisao.gameObject.name == "DestinoNav4") {
        numeroDestino = 3;
        this.transform.LookAt (NavDestino [3].position);
    } else if (colisao.gameObject.name == "DestinoNav5") {
        numeroDestino = 4;
        this.transform.LookAt (NavDestino [4].position);
    }
}

```

*Código 24 – Script para controlar as animações*

### 3.3.7 Comunicação TCP/IP entre DriveByMind e MatLab

A comunicação entre o jogo DriveByMind e o MatLab é de extrema importância no nosso projeto, pois é através da informação trocada entre ambos que é possível controlar o jogo por EEG. Esta comunicação é efetuada na versão dos jogos a serem controlados através do dispositivo g.USBamp, assim como no menu de calibração de frequências. A comunicação apenas será efetuada se ambas as aplicações estiverem dentro da mesma rede, ou em localhost.

Para haver comunicação entre estas aplicações, foi desenvolvido, pelo professor e orientador do projeto Gabriel Pires, um algoritmo em MatLab que trata alguns dos dados necessários para o controlo do DriveByMind. Esta comunicação é orientada à conexão TCP/IP Cliente-Servidor, sendo o MatLab o cliente, e o jogo o servidor.

Para a criação do servidor, foi criado um script que está associado a um objeto ativo do jogo, para assim o servidor poder estar em constante funcionamento. Este script tem o intuito de fazer com que o servidor criado esteja sempre a espera de um cliente, para assim que este se ligar haja uma comunicação entre ambos. Esta comunicação é feita através do Porto “10100”.

No desenvolvimento deste script, é criado um Socket com o endereço IP público do dispositivo em que o jogo está a correr, um *TCP Listener*, com o IP do dispositivo e o respetivo porto de comunicação definido, que servirá para o servidor poder aceitar os pedidos de ligação de clientes, bem como a criação de uma Thread, com o intuito de estar sempre a esperar da conexão de um cliente (Código 25).

```

void Start ()
{
    // IP do dispositivo em que o jogo esta a correr
    IPHostEntry IPHost = Dns.GetHostByName(Dns.GetHostName());
    // string com o endereço IP do dispositivo
    ip_address = "" + IPHost.AddressList[0].ToString();
    // Criacao do Socket
    IPAddress ip_addy = IPAddress.Parse(ip_address);
    // Criacao de um listener, para poder aceitar pedidos de ligacao
    tcp_listener = new TcpListener(ip_addy, port);
    // Thread que estara a escuta por novos clientes
    listen_thread = new Thread(new ThreadStart(ListenForClients));
    listen_thread.Start();
    listen_thread.IsBackground = true;
    Debug.Log("Espera Clientes ....");
    // variavel para controlar outros scripts a existencia da comunicacao
    connect = false;
}

```

*Código 25 – Criação do Socket, TCP Listener e Thread para escutar clientes*

A *Thread* *listen\_thread*, tem como objetivo chamar a função *ListenForClients*, como mostra o código acima. Esta função tem o propósito de ativar o *TCP Listener* criado na função *Start* (Código 25), assim como esperar pela conexão de clientes. Sempre que um cliente efetua uma ligação ao servidor, este ativa uma nova *Thread* denominada *clientThread* (Código 26).

```

private void ListenForClients ()
{
    // Ativar o TCP Listener para ligacao de clientes
    this.tcp_listener.Start();
    while (isTrue == true) {
        //Espera ate um cliente se conectar com o servidor
        TcpClient client = this.tcp_listener.AcceptTcpClient();
        //Cria uma Thread para estar a escuta de uma mensagem do cliente
        clientThread = new Thread(new ParameterizedThreadStart(HandleClientComm));
        clientThread.Start(client);
        // Variavel que controla as ligacoes, devido ao MatLab efetuar 2 ligacoes
        // umma no menu principal, e outro numa janela posterior
        numberConnection++;
        if (numberConnection == 2) {
            connect = true;
            numberConnection = 0;
            Debug.Log("Conexao 2/2.. Pode Jogar");
        } else {
            Debug.Log("Conexao 1/2.. ");
        }
    }
}

```

*Código 26 – Função ListenForClients*

Na função *ListenForClients*, foi criada uma *Thread*, chamada *clientThread*, que servirá para a leitura das mensagens do cliente. Esta leitura é feita através da função *HandleClientComm* (Código 27), que contém um parâmetro de entrada chamado *client*, sendo este objeto o cliente que efetuou a ligação ao servidor. A leitura da mensagem é feita através da criação de uma *Stream*, que contará com um array de bytes para verificar os bytes recebidos. Esta *Stream*, com o nome *cliente\_stream*, estará sempre a espera que o cliente envie uma mensagem através da sua componente *Read*. Quando uma mensagem é recebida, esta é descodificada devido a ser enviada através de bytes, e de seguida colocada numa string para haver uma leitura do comando enviado pelo cliente. Caso haja alguma falha na comunicação, a ligação entre servidor e cliente é terminada, bem como as *Threads* a si associadas.

```
private void HandleClientComm (object client)
{
    tcp_client = (TcpClient)client; // variavel "tcp_client" recebe o objeto passado por parametro "client"
    NetworkStream client_stream = tcp_client.GetStream (); // Criacao de uma Stream, para ler a mensagem
    byte[] message = new byte[4096]; // criacao de um array de bytes para ler a mensagem a receber
    int bytes_read; //Controlar o numero de bytes lidos na mensagem recebida
    while (isTrue == true) {
        bytes_read = 0;
        bytes_read = client_stream.Read (message, 0, 4096); //Bloqueia ate que cliente envie uma mensagem
        //Caso ocorra um erro no Socket, termina a ligacao com o cliente
        if (bytes_read == 0) {
            connect = false;
            tcp_client.Close ();
            break;
        }
        ASCIIEncoding encoder = new ASCIIEncoding (); // Descodificacao ASCII da mensagem recebida
        Debug.Log (encoder.GetString (message, 0, bytes_read));
        comand = encoder.GetString (message, 0, bytes_read); // string que guardara com mensagem recebida
        RecebeuComando = true;
    }
    // Terminar ligacao caso esta seja abortada
    if (isTrue == false) {
        tcp_client.Close ();
        connect = false;
        clientThread.Abort ();
        listen_thread.Abort ();
        tcp_listener.Stop ();
    }
}
```

Código 27 – Função *HandleClientComm*

Para o servidor enviar uma mensagem, foi criada também a função *SendMessageGetFreq*, com dois parâmetros de entrada, sendo eles um TCP Client, que será o cliente que está ligado ao servidor, e a mensagem a enviar, que é definida nos scripts de Destino do Táxi ou Helicóptero. Para o envio da mensagem é criada uma *Stream* com os dados do cliente (stream), e uma *Stream* para escrita da

mensagem (writer). O envio da mensagem é feito através do componente *WriteLine* da *Stream writer*.

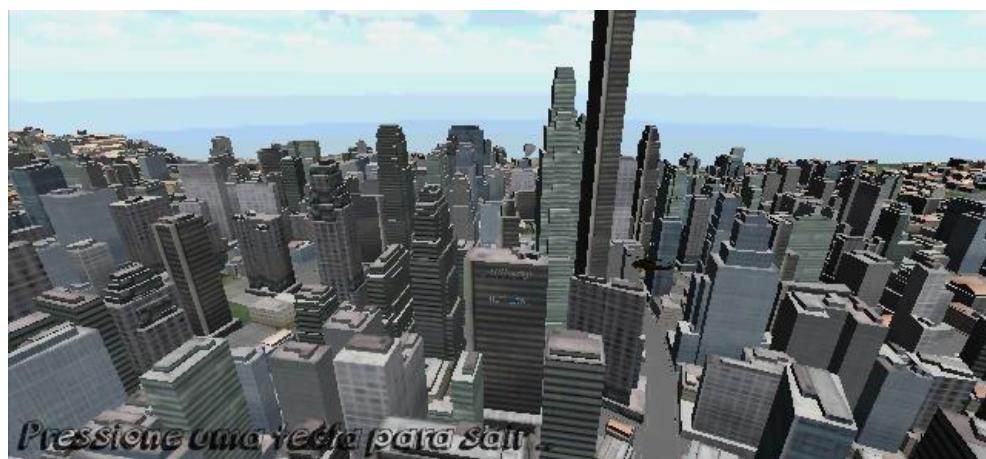
```
void SendMessageGetFreq(TcpClient client, string message)
{
    NetworkStream stream = client.GetStream(); // criacao de uma stream para envio de mensagem
    StreamWriter writer = new StreamWriter(stream); // Criacao de uma stream de escrita
    writer.WriteLine(message); // Enviar mensagem
    writer.Flush(); // Libertar o buffer
}
```

*Código 28 – Função SendMessageGetFreq*

### 3.3.8 Introdução ao Jogo

A cena de introdução ao jogo trata-se de uma breve apresentação do que o jogo se trata e é feita através da movimentação de objetos numa direção, alternando entre câmaras de jogo. Nesta cena, foram colocadas quatro câmaras, com o intuito de criar uma pequena demonstração dos objetivos dos dois jogos 3D disponíveis na aplicação DriveByMind.

A primeira câmara começa de uma posição definida estrategicamente (Figura 98), e faz um movimento de queda ligeira até ir ao encontro de um helicóptero que estará em movimento. Este helicóptero movimenta-se através do componente Navmesh a si associado, tendo como destino uma moeda, tal como no nível 1 do jogo do helicóptero.



*Figura 98 – Visão da primeira câmara da introdução do jogo*

A primeira câmara ao colidir com o helicóptero será desativada, passando a segunda câmara a estar ativa. Esta segunda câmara segue o helicóptero até este apanhar a moeda, mostrando durante esta viagem os estímulos visuais que o jogador poderá encontrar ao jogar o DriveByMind para a versão g.USBamp (Figura 99). Estes estímulos aparecem sincronizados com uma música audível durante a introdução, através de um contador de tempo que está a contar desde o início da cena, e trocando entre estímulos sempre que necessário. (Código 29).

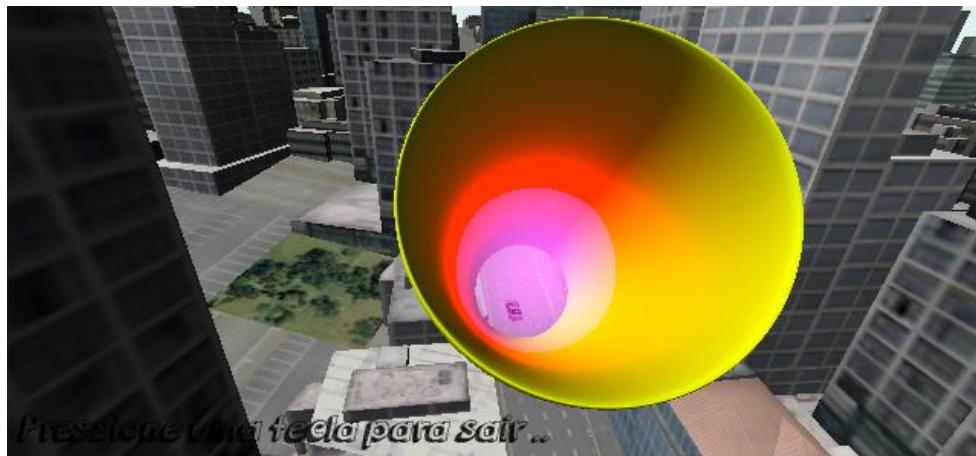


*Figura 99 – Segunda câmara da introdução, com a apresentação de estímulos*

```
void EstimulosSom (float time)
{
    if (time > 10) { SEsq1.SetActive (true); }
    if (time > 10.2) { SEsq1.SetActive (false); SEsq2.SetActive (true); }
    if (time > 10.4) { SEsq2.SetActive (false); }
    if (time > 12.3) { SDir1.SetActive (true); }
    if (time > 12.5) { SDir1.SetActive (false); SDir2.SetActive (true); }
    if (time > 12.7) { SDir2.SetActive (false); }
    if (time > 14.6) { SFrn1.SetActive (true); }
    if (time > 14.8) { SFrn1.SetActive (false); SFrn2.SetActive (true); }
    if (time > 15) { SFrn2.SetActive (false); }
    if (time > 16.8) { SDir1.SetActive (true); SEsq1.SetActive (true); SFrn1.SetActive (true); }
    if (time > 17) {
        SDir1.SetActive (false);
        SEsq1.SetActive (false);
        SFrn1.SetActive (false);
        SDir2.SetActive (true);
        SEsq2.SetActive (true);
        SFrn2.SetActive (true);
    }
    if (time > 17.2) {
        SDir2.SetActive (false);
        SEsq2.SetActive (false);
        SFrn2.SetActive (false);
    }
}
```

*Código 29 – Troca entre estímulos na introdução*

A terceira câmara da introdução será visível quando o helicóptero colidir com a moeda, desativando assim a câmara anterior. Esta terceira câmara servirá para mostrar ao jogador a passagem por um anel de jogo, que fará parte do nível 2 do jogo do helicóptero (Figura 100).



*Figura 100 - Terceira câmara da introdução*

A quarta e última câmara (Figura 101) fica visível assim que a câmara anterior chegar até ao táxi, que circulará pela estrada através de uma componente NavMesh. Esta última câmara segue o táxi até ao passageiro, estando em rotação para o céu para mostrar o nome do jogo (Figura 22).



*Figura 101 – Quarta câmara da introdução*

## 4 Testes e resultados experimentais

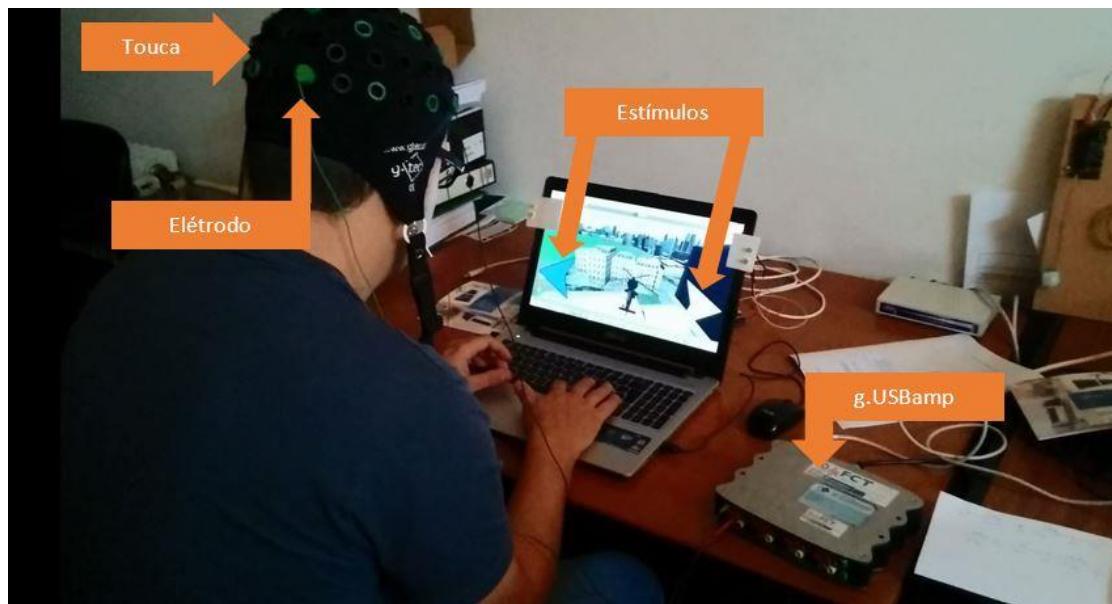
Foram realizados várias experiências para validar os estímulos visuais e o funcionamento do jogo online. As secções seguintes descrevem as experiências e os resultados obtidos.

### 4.1 Setup experimental e análise offline dos SSVEP

#### 4.1.1 Setup experimental e procedimento de montagem

A Figura 102 apresenta uma foto do setup experimental bem como do cenário de jogo.

Os testes foram realizados com uma montagem de elétrodos localizado no córtex visual (região occipital) referido ao lóbulo da orelha e Ground em AFz, utilizando o sistema de aquisição g.USBamp. Os canais EEG utilizados foram o O1, O2 e Oz, localizados de acordo com o sistema internacional 10-20 estendido (Figura 103).



*Figura 102 – Setup experimental*

O procedimento para a colocação dos elétrodos é iniciado pela colocação da touca. Depois, é colocado o gel nos elétrodos para melhorar a captação do sinal. Na posição onde o elétrodo é colocado, o coro cabeludo do utilizador é limpo com gel de limpeza.

Depois de colocados os elétrodos estes são então ligados ao g.USBamp.

Quando terminado o teste é importante a limpeza dos respetivos elétrodos e do coro cabeludo do utilizador para remover o gel.

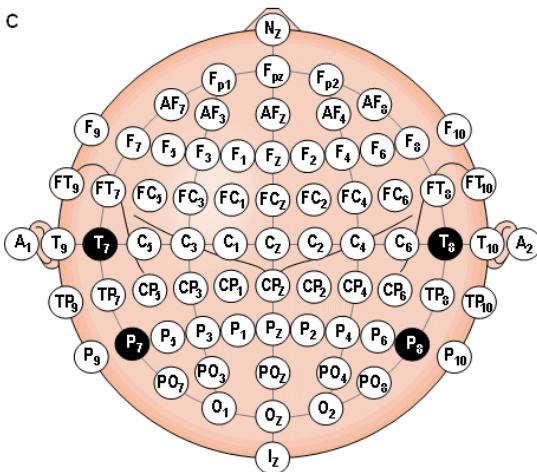


Figura 103 – Sistema internacional 10-20 estendido (posicionamento dos eletródos)

## 4.2 Experiência 1 – análise SSVEP

Na experiência 1, pediu-se ao utilizador que olhasse de forma contínua para um estímulo a uma dada frequência e recolheu-se um dataset para análise dos SSVEP.

O gráfico da Figura 104 apresenta a sobreposição da densidade espectral de potência da resposta SSVEP obtida de dois estímulos com frequências 6 Hz e 9Hz. Para cada um dos estímulos o sinal EEG foi gravado durante aproximadamente 50 segundos com estimulação contínua a 6 Hz ou 9Hz, sem interferência um do outro (um a descoberto e o outro tapado e vice-versa). O sinal foi recolhido na posição O2 do sistema internacional 10-20 (Figura 103). A Figura 104 ilustra claramente os picos de frequência às frequências de estimulação bem como aos múltiplos dessas frequências (12Hz e 18Hz respetivamente, as quais correspondem aos harmónicos).

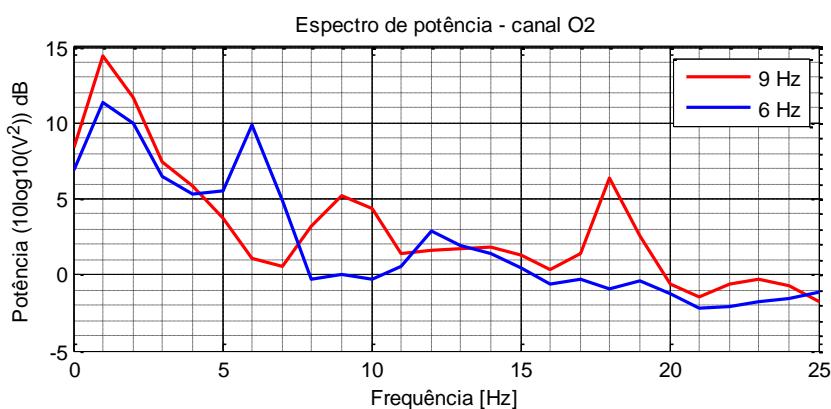
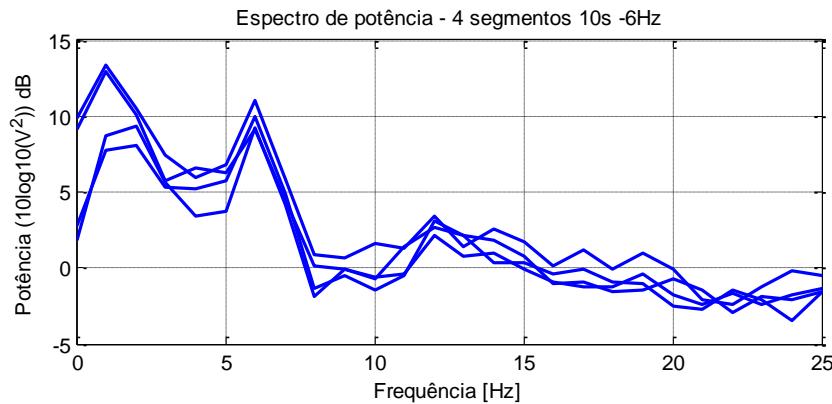
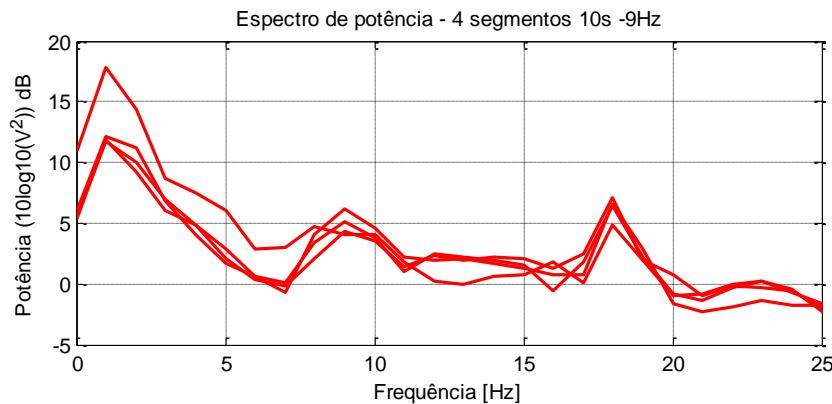


Figura 104 – Espectro de frequência para aproximadamente 50 segundos de estimulação a 6 Hz (azul) e a 9 Hz (vermelho)

Para inferir a robustez da discriminação para segmentos de menor período de tempo, os dados foram segmentados em 4 segmentos de 10s cada. A Figura 105 e a Figura 106 mostram a sobreposição dos espectros de frequência dos 4 segmentos. Os resultados indicam que as respostas aos dois estímulos permanecem discrimináveis para segmentos de 10 s.

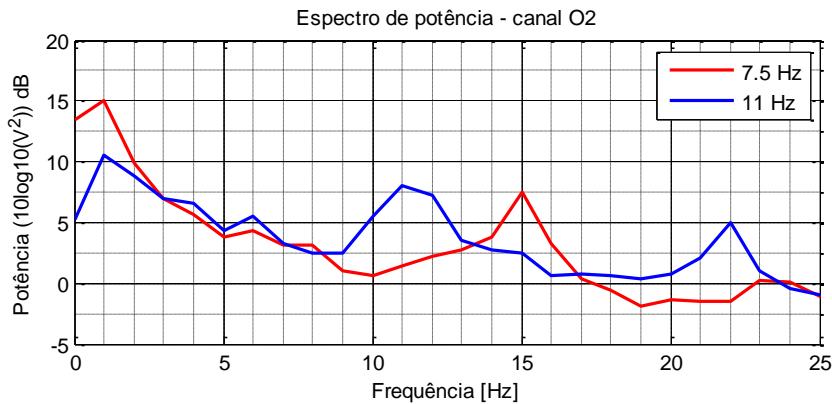


*Figura 105 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 6 Hz*

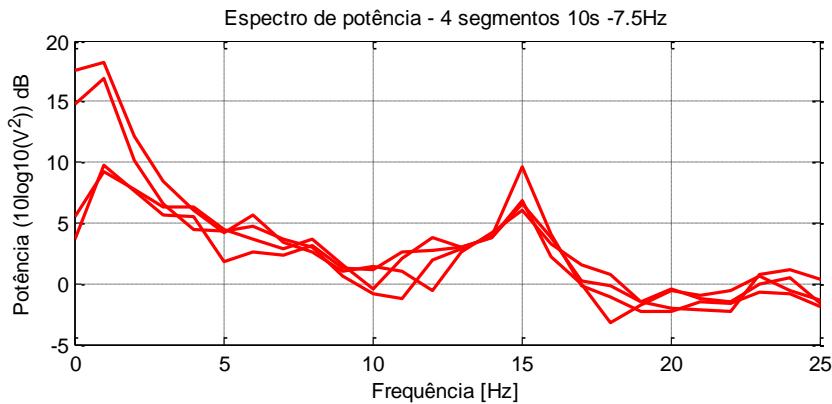


*Figura 106 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 9 Hz.*

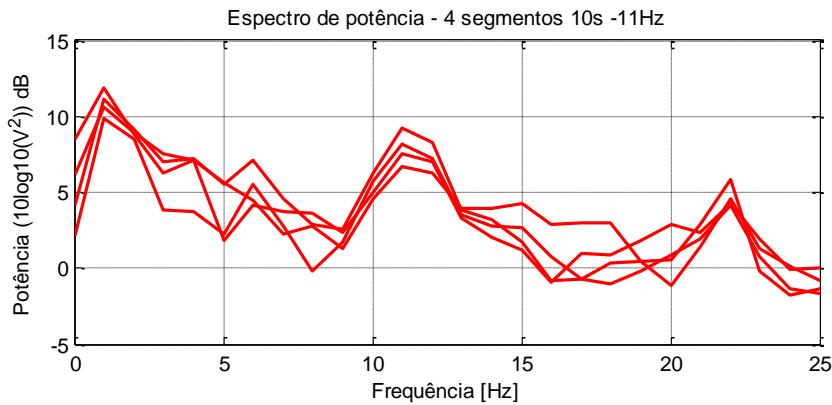
Os testes foram repetidos para frequências de estímulo 7.5 Hz e 11 Hz conforme observado na Figura 107, Figura 108 e Figura 109. Na resposta ao estímulo de 11 Hz é visível o pico de 11Hz e respetivo harmônico a 22 Hz. Na resposta ao estímulo de 7.5 Hz, o pico aos 7.5 é pouco perceptível, no entanto, o seu harmônico aos 15Hz é claramente visível.



*Figura 107 – Espectro de frequência obtido para aproximadamente 50 segundos de estimulação a 7.5 Hz (vermelho) e 11 Hz (azul)*



*Figura 108 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 7.5 Hz*



*Figura 109 – Sobreposição do espectro de frequência de 4 segmentos de 10 segundos com estimulação a 11 Hz.*

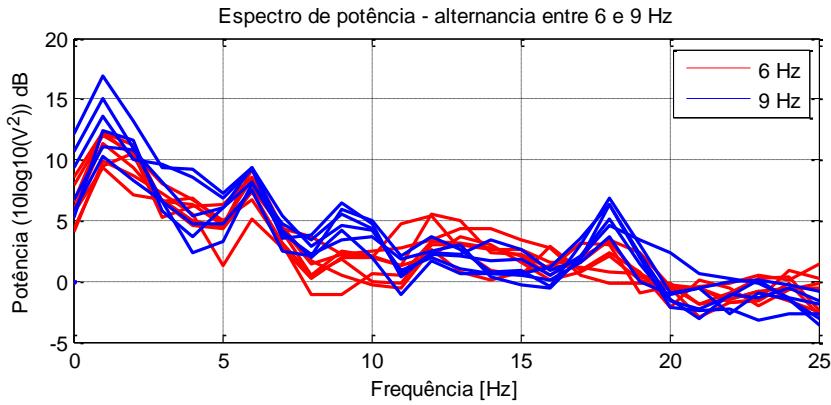
### 4.3 Experiência 2 – análise offline

Na experiência 2, pediu-se ao utilizador que alternasse o olhar de 10 em 10 segundos entre dois estímulos (Esquerda e Direita) a piscar a frequências diferentes (os dois estímulos encontravam-se sempre a descoberto).

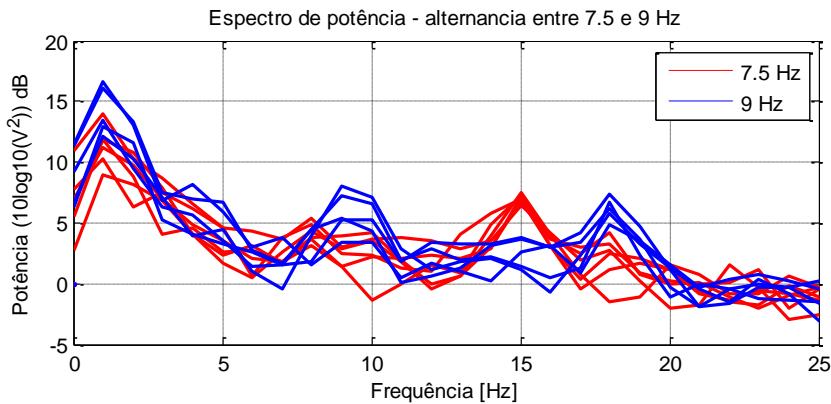


Figura 110 – Teste do jogo DriveByMind utilizando o g.USBamp na experiência 2

Este cenário corresponde a uma situação de controlo do jogo. Foi realizada a experiência com estímulos a 6 Hz (Esquerda) vs. 9 Hz (direita) e com os estímulos 7.5 Hz (Esquerda) vs. 9 Hz (Direita). Os resultados obtidos encontram-se nas Figura 111 e Figura 112. Verifica-se ser possível discriminar entre os dois padrões SSVEP tanto num caso como outro, tendo no entanto sido obtidos melhores resultados com os estímulos 7.5 Hz vs. 9 Hz. A alternância entre os dois estímulos é imediata, pelo que sempre que ocorre uma transição existe “contaminação” entre segmentos, pois a sincronização das ondas SSVEP não é instantânea.



*Figura 111 – Espectros de frequência da resposta alternada entre estímulo esquerdo (6 Hz) e estímulo direito (9 Hz)*



*Figura 112 - Espectros de frequência da resposta alternada entre estímulo esquerdo (7.5 Hz) e estímulo direito (9 Hz)*

#### 4.4 Experiencia 3 – Online

Tendo os estímulos sido validados pelas experiências anteriores, passou-se à realização de experiência online. Como um dos modos de funcionamento do jogo exigia a utilização de 3 estímulos diferentes em simultâneo, fez-se uma análise prévia dos SSVEP. Os dados foram obtidos num cenário em que o utilizador olhava alternadamente para o estímulo de 7Hz, 9Hz e 11Hz, durante 10 segundos. A Figura seguinte corresponde ao espectro de frequência do sinal obtido. Existe alguma “contaminação” de estímulos, no entanto verifica-se que aparecem picos às frequências de 9Hz e 11Hz e respectivos harmónicos (18Hz e 22Hz). O pico à frequência de 7Hz está pouco discriminado, no entanto seu harmônico aos 14 Hz é claro.

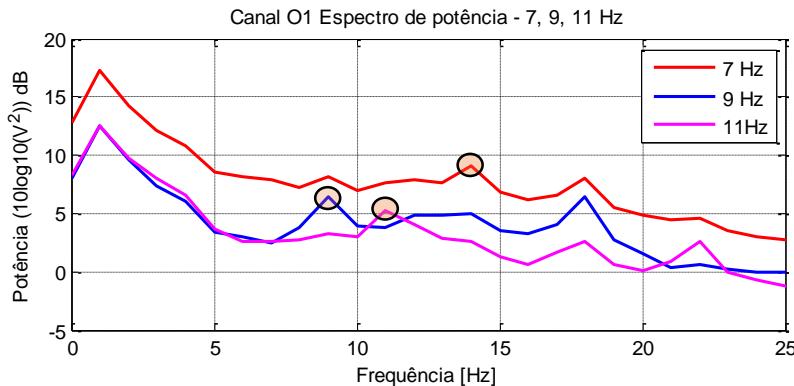


Figura 113 - Espectros de frequência da resposta alternada entre estímulo esquerdo (7.5 Hz), estímulo direito (9 Hz) e estímulo frente (11Hz)

#### 4.4.1 Comando online do jogo

Tal como foi descrito anteriormente, existem dois modos de funcionamento. No primeiro modo de funcionamento, designado por modo 1, o utilizador escolhe entre dois ou três possíveis comandos, esquerda, direita, frente. O helicóptero ou o carro segue por caminhos pré-definidos e nos pontos de bifurcação/decisão, toma a direção detetada (esquerda ou direita). Neste modo, os estímulos aparecem alguns segundos (que pode ser configurado 5 a 10 seg) antes do aparecimento do próximo ponto de decisão. O algoritmo<sup>1</sup> de deteção da intenção do utilizador (descodificação do padrão SSVEP) tem acesso a informação pré-estimulo e pós-estimulo, o que pode ser usado para robustecer a deteção. O algoritmo de deteção é executado em tempo real no real-time Simulink). A deteção utiliza o sinal EEG dos 4 canais já referidos (O1, O2, Oz, POz). Este é o modo de comando utilizado no jogo dos carros 3D e nível I do helicóptero (ver Figura 115 e Figura 116). No jogo de carros 3D, é dada indicação ao utilizador de qual a direção a seguir. No caso do helicóptero, o utilizado tem selecionar a direção que o leva a uma moeda.

<sup>1</sup> Os algoritmos de deteção foram realizados pelo Prof. Gabriel Pires.



Figura 114 – Teste dos estímulos utilizando o g.USBamp na experiência 1

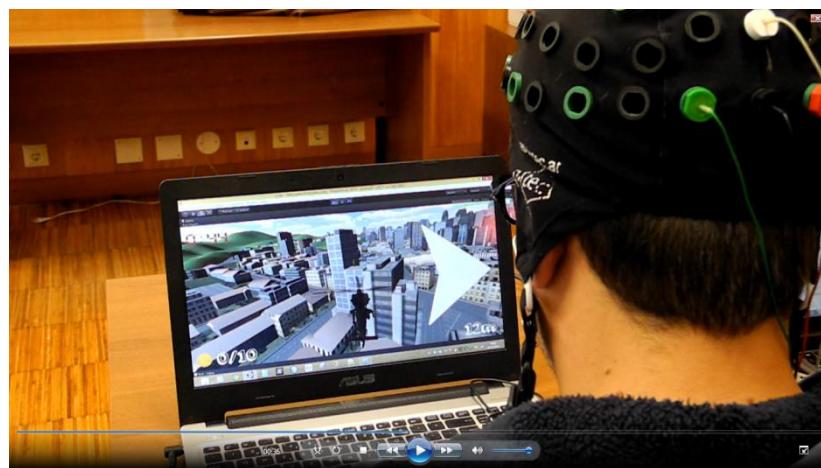


Figura 115 – Controlo jogo do helicóptero modo 1



Figura 116 – Controlo jogo táxi

No funcionamento do jogo em modo 2 (implementado apenas no jogo do Helicóptero), os estímulos estão sempre visíveis e os comandos detetados são enviados consecutivamente, com um determinado intervalo de tempo (ver Figura 117). Para um intervalo pequeno, este modo fornece um comando quase contínuo, semelhante ao obtido com controlo por teclas. Sempre que um comando é detetado, o helicóptero segue a direção detetada até ao próximo comando detetado. A altitude do helicóptero é controlada automaticamente. O objetivo do jogo é seguir o caminho indicado pela seta, tendo de passar dentro de túneis (anéis).



*Figura 117 – Controlo jogo helicóptero modo 2*

A Tabela 1, apresenta os resultados, em percentagem de deteção bem-sucedida, para os vários jogos. Os resultados são referentes à operação do jogo por períodos de 3 a 4 minutos. São apresentados os tempos mínimos com os quais o controlo do jogo foi testado. Foi possível controlar o Helicóptero no modo 2, precisando apenas de 1.5 s de dados EEG para deteção de comando, o que dá uma sensação de controlo quase contínuo semelhante ao que é obtido com teclas ou joystick.

Jogo	Taxa de sucesso (%) e tempo de controlo	
	Participante 1	Participante 2
Carros 3D	90% - 100% (3 s)	-
Helicóptero modo 1	90% - 100% (3 s)	90% - 100% (3 s)
Helicóptero modo 2	90% - 100% (1.5 s)	-

Tabela 1 – Taxa de sucesso de controlo

## 5 Conclusões

Desde o começo que este projeto se tornou um verdadeiro desafio para nós, muito interessante e motivador, tendo em conta as diferentes e novas tecnologias por nós aprendidas durante todo o seu desenvolvimento.

Apesar da sua complexidade e das dificuldades encontradas deu-nos um enorme prazer concebê-lo, e o sentimento de criação de algo inovador e inexistente torna-se numa sensação ótima, sendo este o motivo maior da nossa motivação que nos levou ao sucesso.

Conseguimos assim verificar que, é possível a criação de jogos utilizando várias tecnologias ainda com um vasto campo de exploração como é o caso do Unity, EEG e SSVEP. Da conjugação destas várias tecnologias nasceu assim o DriveByMind, um projeto que tem como principal objetivo atingir o maior público alvo possível e quebrar barreiras com o que a sociedade já conhece no mundo dos jogos de vídeo.

### 5.1 Contribuição para a comunidade

O facto de se tratar de um conceito inovador e inexistente abre várias portas aos diversos mundos das diferentes tecnologias por nós abordadas.

A conjugação destas tecnologias e o modo como o jogo é controlado pode também sair em benefício de jogadores com dificuldades motoras que não conseguem jogar os jogos tradicionais. Abrindo assim horizontes a futuros desenvolvimentos de jogos com este intuito.

A nossa dificuldade inicial ao abordar esta tecnologia por nós desconhecida em termos de desenvolvimento, o Unity, e reportar-mos os conhecimentos que adquirimos na construção do projeto para o tutorial de iniciação ao Unity 3D e também com a realização deste relatório com detalhes mais pormenorizados é uma boa base de conhecimento para futuros desenvolvedores de jogos nesta plataforma. Para tal suporte a futuros trabalhos disponibilizamos também todo o projeto num repositório de código aberto *GitHub* possível de ser descarregado por qualquer pessoa [33].

## 5.2 Desenvolvimento Futuro

Dadas as características e objetivo deste projeto não podemos concluir que o mesmo esteja findado.

Os limites deste projeto prendem-se apenas na imaginação e sentido criativo do desenvolvedor. Ainda assim, julgamos que a introdução de mais algumas animações, assim como colocar tráfego na cidade daria outra vida ao jogo. A possibilidade de interação com mais objetos dentro do cenário de jogo aliada às animações dos mesmos poderia ser um desenvolvimento bastante impulsionador deste projeto.

Outra questão a ter em conta em futuros desenvolvimentos são, a inserção dos estímulos visuais em objetos do próprio jogo a piscar como por exemplo sinais de trânsito ao invés das atuais setas diretivas.

## 5.3 Apreciação final

Dum modo geral estamos bastante satisfeitos e orgulhosos do nosso esforço e portanto do nosso produto final, o DriveByMind.

Desde o início do projeto, ao “mergulharmos” no desconhecido que era para nós o Unity e a construção de jogos de vídeo, passando pela exploração das mais variadas tecnologias que utilizámos para a concretização do projeto, foram muitos os conhecimentos adquiridos ao longo destes meses. Com isto provámos a nossa autonomia e capacidade de autoaprendizagem na concretização de diferentes tarefas e na tomada de decisões perante os mais diversos problemas que foram surgindo no decorrer do projeto. O que acaba também por ser uma mais-valia muito grande para a nossa carreira profissional.

Foram de facto muitas horas de trabalho investidas, muitas dores de cabeça sentidas, porém tudo é justificado quando temos a oportunidade de ver o fruto o nosso árduo trabalho terminado e cem por cento funcional.

# Bibliografia

- [1] VITA - Vida Assistida em Ambientes Inteligentes;  
[http://portal2.ipt.pt/pt/ipt/unidades\\_de\\_i\\_d\\_tecnologico\\_e\\_artistico/vita/](http://portal2.ipt.pt/pt/ipt/unidades_de_i_d_tecnologico_e_artistico/vita/) (Outubro 2014)
- [2] Unity Game Engine ; <https://unity3d.com/pt> (Outubro 2014)
- [3] R. Parafita, **G. Pires**, U. Nunes, and M. Castelo-Branco (2013), "A spacecraft game controlled with a brain-computer interface using SSVEP with phase tagging", IEEE 2nd Int.l Conf. on Serious Games and Applications for Health, SeGAH2013
- [4] Jian Ding, George Sperling, and Ramesh Srinivasan, "Attentional modulation of SVEP power depends on the network tagged by the flicker frequency", Cereb Cortex. 2006 July ; 16(7): 1016–1029
- [5] Indie Games – the WebLog - <http://indiegames.com/index.html> (outubro 2014)
- [6] LiveBioFeedback
- [7] g.USBamp USB Biosignal Amplifier - <http://www.gtec.at/Products/Hardware-and-Accessories/g.USBamp-Specs-Features> (Outubro 2014)
- [8] Esri CityEngine - <http://www.esri.com/software/cityengine> (Outubro 2014)
- [9] CityEngine - <http://en.wikipedia.org/wiki/CityEngine> (Outubro 2014)
- [10] SketchUp - <http://www.sketchup.com/> (Outubro 2014)
- [11] SketchUp - <http://en.wikipedia.org/wiki/SketchUp> (Outubro 2014)
- [12] Home of the Blender Project – Free and Open 3D Creation Software - <http://www.blender.org/> (Outubro de 2014)
- [13] Blender(Software) - [http://en.wikipedia.org/wiki/Blender\\_\(software\)](http://en.wikipedia.org/wiki/Blender_(software)) (Outubro 2014)
- [14] thinkgear\_connector [NeuroSky developer – Docs] -  
[http://developer.neurosky.com/docs/doku.php?id=thinkgear\\_connector\\_tgc](http://developer.neurosky.com/docs/doku.php?id=thinkgear_connector_tgc) (Outubro 2014)
- [15] ThinkGear Connector User's Guide (PDF) -  
[http://developer.neurosky.com/docs/lib/exe/fetch.php?media=thinkgear\\_connector\\_user\\_guide.pdf](http://developer.neurosky.com/docs/lib/exe/fetch.php?media=thinkgear_connector_user_guide.pdf) (Outubro 2014)

- [16] NeuroSky Store – <http://store.neurosky.com> (Outubro 2014)
- [17] EPOC - <http://www.emotiv.com/epoc.php> (Outubro 2014)
- [18] Blog Archive»Emotiv EPOC Neuroheadset Update -  
<http://grinding.be/2008/03/22/emotiv-epoch-neuroheadset-update/> (Outubro 2014)
- [19] Emotiv | EEG System | Electoencephalography - <http://www.emotiv.com/>  
(Outubro 2014)
- [20] Mindflex - <http://mindflexgames.com/> (Outubro 2014)
- [21] Electroencefalografia - <http://pt.wikipedia.org/wiki/Eletroencefalografia>  
(Outubro 2014)
- [22] Desenvolvimento de um simulador controlado por interface cérebro-computador  
não invasiva para treinamento na utilização de cadeira de rodas -  
<http://seer.ufrgs.br/index.php/renote/article/view/44716> (Outubro 2014)
- [23] BodyWave Reads Brain Activity | Space Foundation -  
<http://www.spacefoundation.org/media/space-watch/bodywave-reads-brain-activity>  
(Outubro 2014)
- [24] OCZ Storage Solutions – A Toshiba Group Company - <http://ocz.com/> (Outubro 2014)
- [25] BCInet – Redefining human interaction - <http://www.bcinet.com/products/>  
(Outubro 2014)
- [26] MindFlex hoax? - <https://www.youtube.com/watch?v=HsmLA9PqTGM>  
(Outubro 2014)
- [27] BCI Sensor Based Environment Changing System for Immersion of 3D Games -  
<http://www.hindawi.com/journals/ijdsn/2014/620391/> (Outubro 2014)
- [28] 3D Warehouse - <https://3dwarehouse.sketchup.com/> (Outubro 2014)
- [29] Unity – Sprite Renderer -  
<https://unity3d.com/pt/learn/tutorials/modules/beginner/2d/sprite-renderer> (Outubro 2014)
- [30] Unity – Scripting API: Transform -  
<http://docs.unity3d.com/ScriptReference/Transform.html> (Outubro 2014)

- [31] Unity – Scripting API: Rigidbody2D -  
<http://docs.unity3d.com/ScriptReference/Rigidbody2D.html> (Outubro 2014)
- [32] Unity – Scripting API: BoxCollider2D -  
<http://docs.unity3d.com/ScriptReference/BoxCollider2D.html> (Outubro 2014)
- [33] RAMCosta/Unity3DFinalProject -  
<https://github.com/RAMCosta/Unity3DFinalProject> (Novembro 2014)
- [34] Mixamo – 3D Animation Online Services, 3D Characters, and Character Rigging  
- <https://www.mixamo.com/> (Novembro 2014)
- [35] TF3DM – 3D Models for free - <http://tf3dm.com/> (Maio 2014)
- [36] Asset Store Unity – Asset Store - <https://www.assetstore.unity3d.com/en/> (Junho 2014)
- [37] Wii | Nintendo - <http://www.nintendo.pt/Wii/Wii-94559.html> (Novembro 2014)
- [38] Kinect para Xbox 360 - <http://www.xbox.com/pt-pt/kinect/> (Novembro 2014)



# Anexo 1 Enunciado do projeto



Instituto Politécnico de Tomar  
Escola Superior de Tecnologia  
Licenciatura em Engenharia Informática

## Proposta de Projecto Final

Ano lectivo: 2013 / 2014

### Controlo de jogos 3D através de ondas cerebrais

#### Sumário:

Pretende-se com este projeto fazer o desenvolvimento de um jogo controlado por ondas cerebrais captadas através do sistema Mindwave da Neurosky (ou g.USBBamp da g.tec.). O jogo deve ser desenvolvido usando a plataforma Unity 3D para PC (S.O. Windows) e para dispositivos móveis (S.O. Android). O projeto será compreendido pelas seguintes partes: 1) desenvolvimento em Unity 3D para PC e exportação para Android; 2) desenvolvimento de módulo de comunicação entre mindwave e Unity 3D; 3) controlo do jogo com sinais já processados fornecidos pelo mindwave e numa segunda fase com processamento próprio; 4) Análise da possibilidade de integração com sistema g.USBBamp.

Este projeto pretende dar continuidade ao sistema LiveBioFeedback desenvolvido no ano anterior.

Nº de alunos a envolver: <2>

#### Objectivos específicos:

1. Levantamento do estado da arte no âmbito do projecto.
2. Análise da framework Unity 3D e respetivos requisitos para utilização em PC, Android e protocolos de comunicação de dados.
3. Desenvolvimento do jogo em Unity 3D para PC e exportação para Android.
4. Comunicação e comando do jogo com mindwave.
5. Integração de sistema g.USBBamp com comunicação UDP/IP.
6. Teste e operacionalização do sistema.
7. Documentação do projecto.

#### Pré-requisitos:

- Sólidos conhecimentos de sistemas distribuídos e programação
- Ter realizado uma entrevista com os orientadores para aferir do interesse e disponibilidade para a realização deste projecto.

#### Orientador(es):

António Manso

Gabriel Pires

O(s) Proponente(s)

Disciplina de Projecto Final

## Anexo 2 Atas de reunião de projeto

**Acta de reunião de P.F. nº: 1  
2014**

**Data: 06 / Março /**

### **Controlo de Jogos 3D por EEG**

#### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. António Manso	
Prof. Gabriel Pires	

#### **Documentos da reunião:**

1. Enunciado do Projeto

#### **Ordem de trabalho:**

1. Explicação sobre o projeto
2. Explicação sobre o equipamento a utilizar na realização do mesmo

#### **Acta da reunião:**

- Breve explicação sobre o funcionamento da tecnologia *MindWave* e como será integrada no projeto.
- Discussão sobre as diferentes fases do projeto

#### **Objetivos Semanais:**

1. Estudo do *Unity*

**Acta de reunião de P.F. nº: 2  
2014**

**Data: 13 / Março /**

## **Controlo de Jogos 3D por EEG**

### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. António Manso	

### **Documentos da reunião:**

- 2. Demo de Cenário em Unity

### **Ordem de trabalho:**

- 3. Apresentação da Demo.
- 4. Estudo de tecnologias de comunicação TCP.
- 5. Implementação de um servidor TCP em Java.
- 6. Implementação de um cliente C# em Unity.
- 7. Comunicação entre Servidor e Cliente através de Sockets TCP.

### **Acta da reunião:**

- Apresentação da Demo do jogo 3D onde foram demonstrados os seguintes itens:
  - Cenário básico em 3D
  - Movimento de um boneco através das setas direcionais
  - Introdução de alguns objetos no cenário.
- Exportação para Android

### **Objetivos Semanais:**

- 1. Construir um cenário de uma cidade em 3D
- 2. Introduzir um helicóptero

Movimentar o helicóptero com valores recebidos pelo socket através do servidor.

**Acta de reunião de P.F. nº: 3  
2014**

**Data: 20 / Março /**

## **Controlo de Jogos 3D por EEG**

### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. António Manso	
Prof. Gabriel Pires	

### **Documentos da reunião:**

3. Demo de Jogo com Helicóptero 3D

### **Ordem de trabalho:**

8. Apresentação da Demo
9. Discussão sobre as técnicas envolvidas na demonstração

### **Acta da reunião:**

- Apresentação da Demo do Jogo 3D onde foram demonstrados os seguintes itens:
  - Construção de um cenário 3D
  - Utilização do *Blender* para criação dos prédios
  - Movimento do helicóptero na vertical através de sockets
- Investigação do modo de funcionamento do *Unity* em 2D
- Visualização de jogos em 2D para uma melhor percepção do seu funcionamento

### **Objetivos Semanais:**

1. Jogo de Carros em 2D:
  - a. Cenário em movimento
  - b. Introdução de som
  - c. Deteção de colisões

**Acta de reunião de P.F. nº: 4  
2014**

**Data: 27 / Março /**

## **Controlo de Jogos 3D por EEG**

### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. António Manso	

### **Documentos da reunião:**

4. Demo de Jogo de Carros 2D

### **Ordem de trabalho:**

10. Apresentação da Demo
11. Discussão sobre as técnicas envolvidas na demonstração
12. Discussão da possibilidade de efetuar uma Demo de um jogo de carros em 3D

### **Acta da reunião:**

Apresentação da Demos do Jogo 2D onde foram demonstrados os seguintes itens:

- Construção de um cenário 2D
- Movimento de objetos
- Colocação de obstáculos na pista de modo aleatório
- Detetor de colisões
- Exportação da Demo para *Android*

### **Objetivos Semanais:**

2. Construção de um cenário para um jogo de carros em 3D
3. Realização de um menu de jogo
4. Criação de animações

**Acta de reunião de P.F. nº: 5  
2014**

**Data: 03 / Abril /**

## **Controlo de Jogos 3D por EEG**

### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. António Manso	

### **Documentos da reunião:**

5. Demo de Jogo de Carros 3D

### **Ordem de trabalho:**

13. Apresentação da Demo
14. Discussão sobre as técnicas envolvidas na demonstração
15. Aprimoramento do jogo

### **Acta da reunião:**

Apresentação da Demo do Jogo 3D onde foram demonstrados os seguintes itens:

- Animações
- Desenho de cenários em 3D
- Utilização do *Blender* para construção dos prédios
- Utilização do *EasyRoads* para desenho da estrada
- Utilização de Som no Jogo
- Construção de Menus
- Utilização de Motor de físicas do *Unity*

### **Objetivos Semanais:**

1. Introdução de níveis.
2. Desenvolver um Jogo controlado com movimentos verticais.
3. Elaboração do relatório.

**Acta de reunião de P.F. nº: 6  
2014**

**Data: 10 / Abril /**

## **Controlo de Jogos 3D por EEG**

### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. António Manso	

### **Documentos da reunião:**

- 6. Demo de Jogo de Helicóptero 3D

### **Ordem de trabalho:**

- 16. Apresentação da Demo
- 17. Discussão sobre as técnicas envolvidas na demonstração
- 18. Conexão entre Unity3D e o Mindwave

### **Acta da reunião:**

- Apresentação da Demo do Jogo 3D onde foram demonstrados os seguintes itens:
  - Animações
  - Desenho de cenários em 3D
- Investigação sobre conexão entre Unity3D e o Mindwave
  - Exploração do ThinkGear Connector

### **Objetivos Semanais:**

1. Utilização do NeuroSky para controlo de um objeto do jogo
2. Ligação para controlar o helicóptero
3. Documentação sobre a conexão do Unity com o Mindwave
4. Menu de Jogo
5. Níveis

**Acta de reunião de P.F. nº: 7**

Data: 06 / Maio / 2014

**Controlo de Jogos 3D por EEG****Pessoas presentes na reunião**

Nome	Rubrica
Rafael Costa	
Prof. Gabriel Pires	

**Documentos da reunião:**

7.

**Ordem de trabalho:**

19. Discussão do ponto de situação do trabalho efectuado até à data.
20. Discussão dos pontos a melhorar no trabalho.
21. Definição do trabalho a efectuar.

**Acta da reunião:**

- Explicação de todo o trabalho já efectuado
  - Detalhes de todo o trabalho que já foi feito
  - Evoluções no trabalho e no unity3D
- Definição de aspectos a melhorar no trabalho já desenvolvido
  - Qual o próximo ponto a melhorar
- Definição do trabalho a efectuar (evoluções a realizar)
  - Qual o próximo ponto a implementar (Conexão do *MindWave*)

**Objetivos Semanais:**

6. Conexão do jogo carros2D com o *MindWave* através de um servidor *java*.
7. Continuação de desenvolvimento do jogo 3D (helicóptero).

**Acta de reunião de P.F. nº: 8**

Data: 13 / Maio / 2014

**Controlo de Jogos 3D por EEG****Pessoas presentes na reunião**

Nome	Rubrica
Jorge Martins	
Rafael Costa	
Prof. Gabriel Pires	
Prof. António Manso (via Skype)	

**Documentos da reunião:**

8. Demo de Jogo de Helicóptero 3D
9. Demo do Jogo Carros 2D
10. Servidor Java para conexão entre Unity e Mindwave

**Ordem de trabalho:**

22. Apresentação das Demos
23. Discussão sobre as técnicas envolvidas na demonstração
24. Resolução de problemas de programação no jogo de Carros 2D

**Acta da reunião:**

- Apresentação da Demo do Jogo Helicóptero 3D onde foram demonstrados os seguintes itens:
  - Desenho de cenários em 3D
- Apresentação da Demo do Jogo Carros 2D onde foram demonstrados os seguintes itens:
  - Conexão entre o Unity 3D e o Mindwave através do Servidor Java
  - Controlo do carro através do Mindwave
  - Construção de Menu de Jogo

**Objetivos Semanais:**

8. Construção de uma cidade em 3D
9. Movimentar o Helicóptero de um ponto A para B
10. Conectar o Mindwave com o jogo Helicóptero 3D
11. Melhorar o movimento do carro (Carros 2D)

**Acta de reunião de P.F. nº: 9****Data: 20 / Maio / 2014****Controlo de Jogos 3D por EEG****Pessoas presentes na reunião**

Nome	Rubrica
Jorge Martins	
Rafael Costa	
Prof. Gabriel Pires	

**Documentos da reunião:**

11. Demo de Jogo de Helicóptero 3D
12. Demo de Jogo de Carros 2D

**Ordem de trabalho:**

25. Apresentação das Demos
26. Discussão sobre as técnicas envolvidas na demonstração

**Acta da reunião:**

- Apresentação da Demo do Jogo Helicóptero 3D onde foram demonstrados os seguintes itens:
  - Pequena construção de uma cidade
  - Movimentação do Helicóptero na cidade
- Apresentação da Demo do Jogo Carros 2D onde foram demonstrados os seguintes itens:
  - Movimentação de carro aperfeiçoada

**Objetivos Semanais:**

12. Continuar construção da cidade
13. Movimentar Helicóptero de um Ponto A para B
14. Adicionar ao jogo Carros 2D (controlado pelo Mindwave) alguns objetos para colisão
15. Implementar no Carros 2D (controlado pelo Mindwave) um sistema de pontuação

**Acta de reunião de P.F. nº: 10**

Data: 27 / Maio / 2014

**Controlo de Jogos 3D por EEG****Pessoas presentes na reunião**

Nome	Rubrica
Jorge Martins	
Rafael Costa	
Prof. Gabriel Pires	

**Documentos da reunião:**

- 13. Demo de Jogo de Helicóptero 3D
- 14. Demo de Jogo de Carros 2D

**Ordem de trabalho:**

- 27. Apresentação das Demos
- 28. Discussão sobre as técnicas envolvidas na demonstração

**Acta da reunião:**

- Apresentação da Demo do Jogo 3D onde foram demonstrados os seguintes itens:
  - Construção de uma pequena parte da cidade
  - Utilização do CityEngine3D para a construção de prédios e casas
- Apresentação da Demo do Jogo 2D onde foram demonstrados os seguintes itens:
  - Sistema de pontuação
  - Objetos para colisão

**Objetivos Semanais:**

- 16. Continuar construção da cidade
- 17. Movimentar Helicóptero de um Ponto A para B
- 18. Criar estímulos laterais a piscar a diferentes frequências (8 a 20 Hz)

**Acta de reunião de P.F. nº: 11  
2014**

**Data: 03 / Junho /**

## **Controlo de Jogos 3D por EEG**

### **Pessoas presentes na reunião**

<b>Nome</b>	<b>Rubrica</b>
Jorge Martins	
Rafael Costa	
Prof. Gabriel Pires	

### **Documentos da reunião:**

- 15. Demo de Jogo de Helicóptero 3D

### **Ordem de trabalho:**

- 29. Apresentação da Demos e evoluções efectuadas
- 30. Discussão sobre as técnicas envolvidas na demonstração
- 31. Discussão sobre problemas nas técnicas envolvidas

### **Acta da reunião:**

- o Apresentação da Demo do Jogo 3D onde foram demonstrados os seguintes itens:
  - Evolução ao nível do aspecto gráfico da cidade
  - Utilização do CityEngine3D para a construção de prédios, casas e estradas
  - Introdução do modo de estímulos para controlo do helicóptero (Setas indicativas com diferentes frequências)
  - Validação das frequências dos estímulos (verificação de problemas com as frequências implementadas, onde falha a precisão das mesmas)

### **Objetivos Semanais:**

- 19. Continuar construção gráfica da cidade
- 20. Resolução do problema de precisão das frequências dos estímulos implementados nas setas indicativas.

## Anexo 3    Tutorial Unity 3D

### 1 Tutorial Unity3D

Neste capítulo faremos uma introdução ao Unity3D. Indicando e explicando as principais ferramentas desta aplicação.

Para concluir demonstramos alguns dos principais passos para a criação de um projeto em Unity3D.

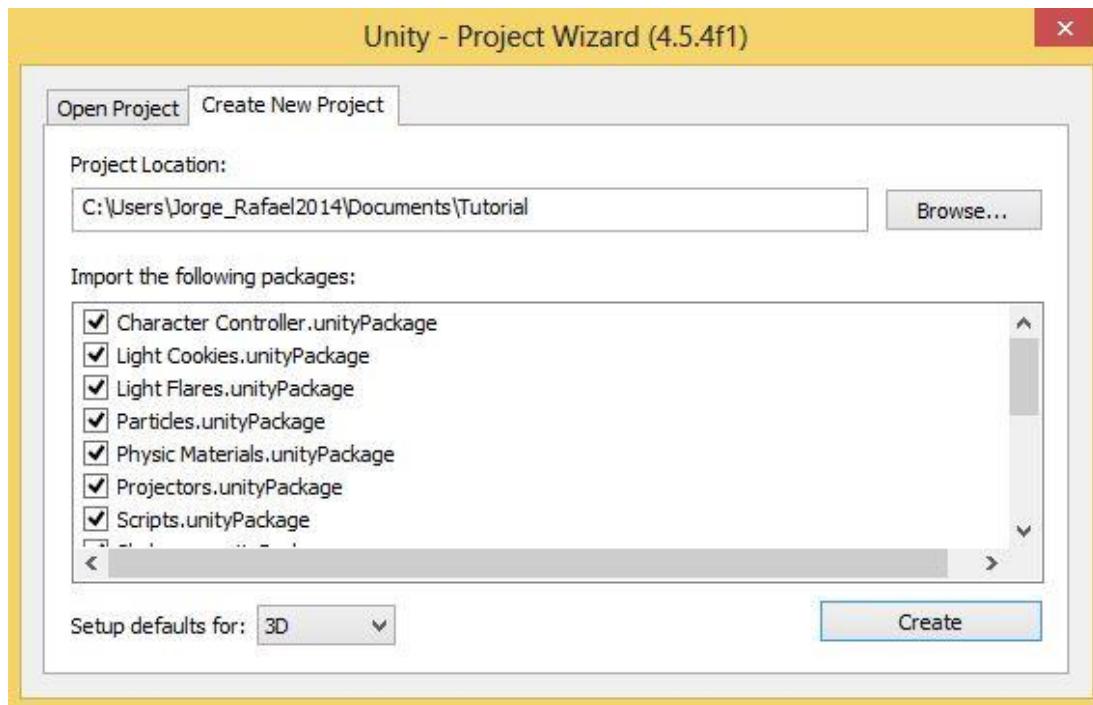
#### 1.1 Criação de um projeto em Unity

Ao executar o Unity pela primeira vez, será exibida no ecrã uma janela para a criação de um projeto.

É nesta que se irá definir a localização do projeto, os *packages* de objetos que pretendemos importar, assim como o tipo de projeto a realizar (2D ou 3D).

Depois da sua criação o Unity carregará todos os *Packages* escolhidos e é-nos mostrado o ambiente de trabalho da aplicação.

É aconselhado salvaguardar o trabalho quando se inicia o projeto, para tal devemos no menu escolher a opção *File>Save Scene*.



*Figura 118 – Janela inicial para a criação de um projeto em Unity*

## 1.2 Componentes

Introdução e descrição detalhada dos principais componentes existentes em Unity que compõem um cenário de jogo.

### 1.2.1 Física

Descrição dos componentes do motor de física existentes em Unity.

Estes componentes tornam o jogo bastante mais realista e, assim, mais agradável para o jogador.

#### 1.2.1.1 Rigidbody

Característica possível de adicionar a um determinado objeto.

Os motores de física do Unity efetuam cálculos para que durante o jogo, os objetos sejam afetados pelas leis da física, tornando-os mais realistas. A gravidade, massa, velocidade e atrito são as propriedades que esta característica possui.

Para adicionar este componente a um objeto deveremos escolher o menu *Component>Physics*.

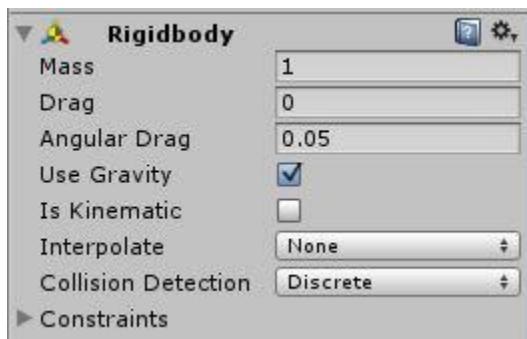


Figura 119 – Janela das definições do componente Rigidbody

### 1.2.1.2 Colisões

As colisões entre objetos durante o jogo são da responsabilidade do motor de física do Unity. A sua função é alterar o comportamento de um determinado objeto quando colide com outro, alterando a sua velocidade ou a direção do movimento.

Para o Unity detetar uma colisão entre objetos, estes necessitam conter o componente *Collider*, para isso é necessário adicioná-lo ao objeto através do menu *Component>Physics*.

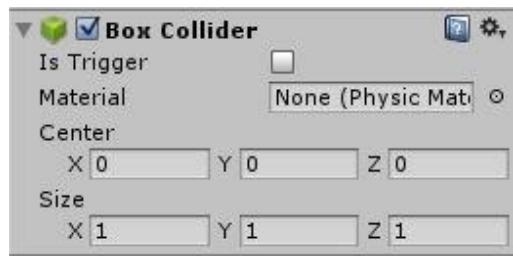
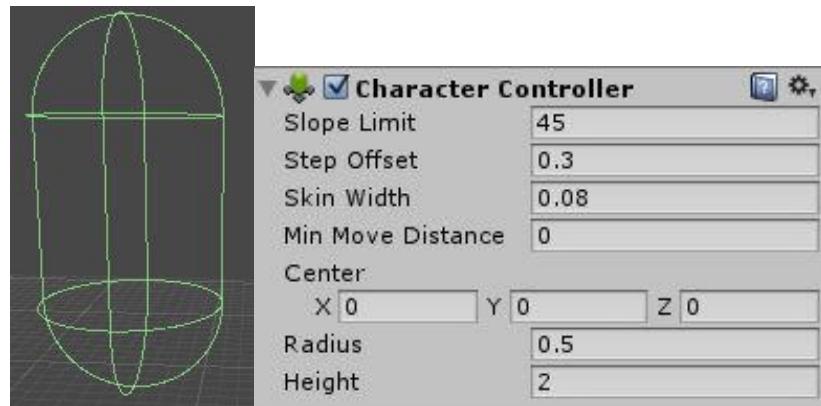


Figura 120 – Janela das definições da Box Collider

### 1.2.1.3 Character Controller

Este componente é um simulador de personagens de jogo, utilizado em jogos em terceira ou primeira pessoa que não usam a componente Rigidbody.

Este componente atribui ao personagem um *Collider* simples em forma de cápsula, que se posiciona na vertical, simulando assim uma personagem de jogo.

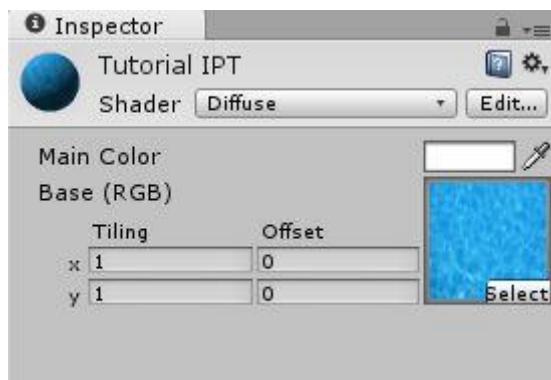


*Figura 121 – Componente Character Controller(Esquerda) e respectiva janela de configurações (direita)*

### 1.2.2 Material

Utilizados para adicionar diferentes texturas a objetos.

Para a criação do material será através do menu *Assets>Create* e escolher a opção *Material*. Para adicionar o respetivo material a um determinado objeto, apenas é necessário arrastá-lo para o objeto em questão.



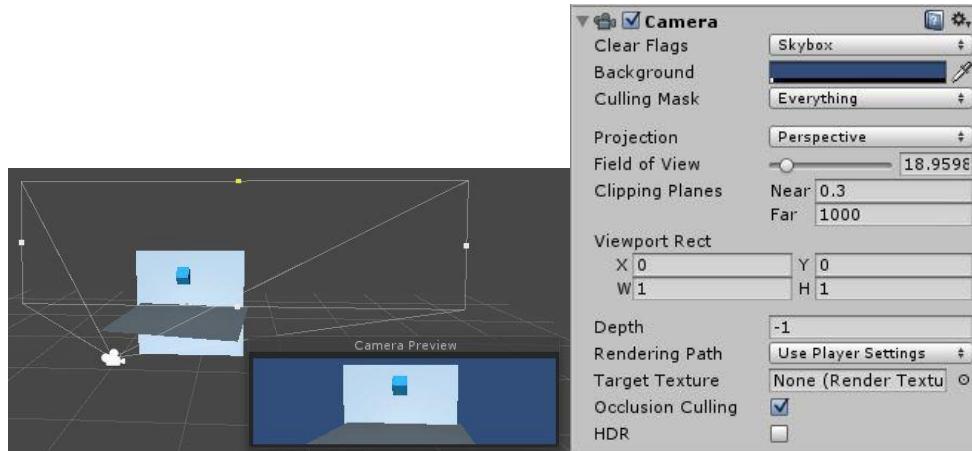
*Figura 122 – Janela das definições dos materiais dos objectos*

### 1.2.3 Câmeras

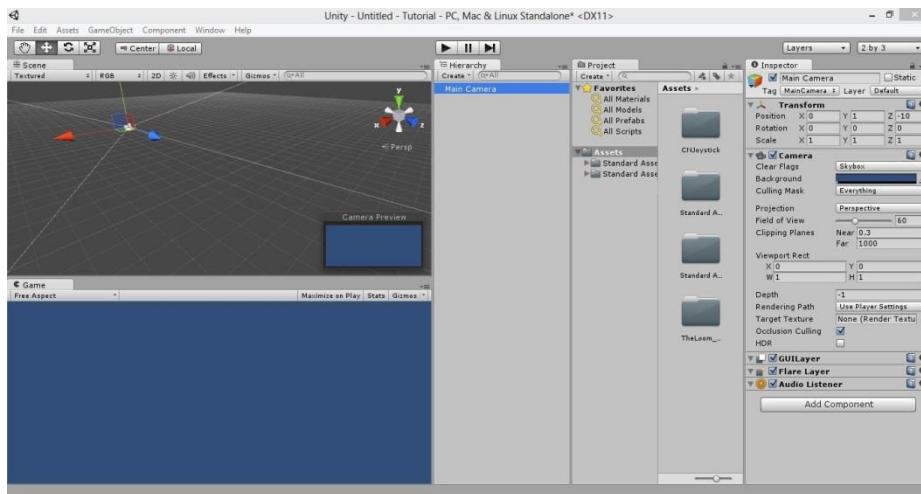
Em Unity estas são utilizadas com intuito de mostrar ao jogador o cenário de jogo.

É necessária a utilização de pelo menos uma câmara para o utilizador poder ver o seu jogo. Mas existe também a possibilidade de serem utilizadas mais do que uma, criando assim efeitos personalizados e avançados, como dividir o ecrã para dois jogadores por exemplo.

Para adicionar câmaras de jogo acedemos ao menu *GameObject>Create Other* e selecionamos a opção *Camera*.



*Figura 123 – Câmera de jogo e respetiva janela de configurações*



*Figura 124 – Câmera principal (Main) do jogo*

#### 1.2.4 Luzes

Têm a função de definir a cor e o tipo de iluminação do cenário de jogo.

Estas luzes podem ser utilizadas em conjunto e em diversos locais do cenário, podendo assim criar um ambiente mais adequado para o jogo em questão.

Esta função do Unity encontra-se no menu *GameObject>Create Other*.



Figura 125 – Janela de configurações do componente Luz

### 1.2.5 GameObject

Objetos que fazem parte do cenário de jogo.

Os mesmos, inicialmente não contêm nenhum tipo de componente ou característica, o que pode ser alterado adicionando-lhes qualquer componente Unity.

A criação destes objetos encontra-se no menu *GameObject*.

### 1.2.6 Prefabs

Servem de armazenamento para alocar todas as características e componentes de um *GameObject*, criando assim objetos pré-fabricados. Impedindo assim que as características dos objetos se percam.

Podem ser criados através do menu *Assets>Create* e escolher a opção *Prefab*. Para associar o objeto desejado a um *Prefab* apenas é necessário arrastar o objeto desejado para o *Prefab* criado.

#### 1.2.6.1 Importar/Exportar

Uma das grandes vantagens dos *Prefabs* é a possibilidade de importação e exportação para diferentes projetos sem que estes percam as suas características.

Existem duas possibilidades para exportar um *Prefab*. Uma opção será clicar o botão direito do rato em cima do armazenamento e selecionar a opção *Export Package*. Outro método é aceder ao menu *Assets>Export Package*.

Para a importação acedemos ao menu *Assets* e seleccionamos a opção *Import Package*, escolhendo seguidamente o *Prefab* armazenado localmente ou disco ou noutra dispositivo de armazenamento.

### 1.3 Ambiente Unity

Uma grande vantagem do Unity face a outros programas de desenvolvimento é o seu ambiente gráfico intuitivo e de fácil utilização. A possibilidade de moldar objetos dentro do cenário de jogo facilita bastante o desenvolvimento como poderemos contactar na imagem seguinte.

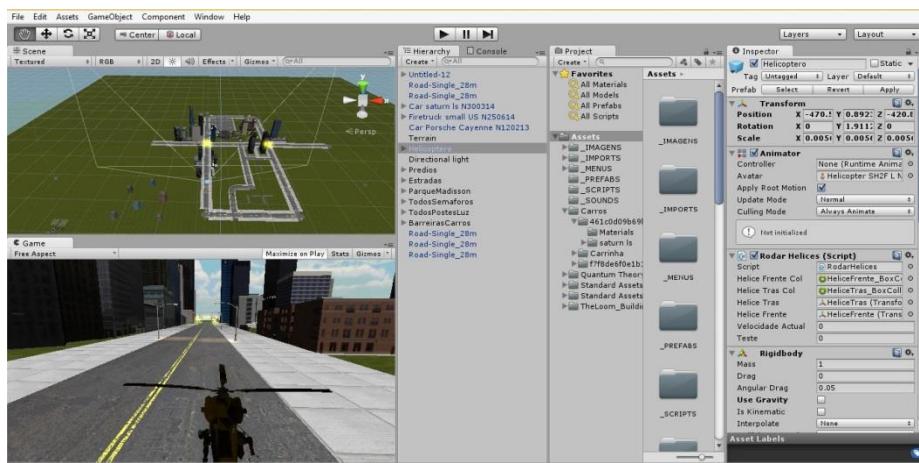


Figura 126 - Janela do ambiente de desenvolvimento do Unity

#### 1.3.1 Hierarchy

Local onde se encontram todos os objetos que fazem parte do cenário de jogo.

Neste separador o utilizador pode organizar os seus objetos como se fossem pastas, criando uma hierarquia.

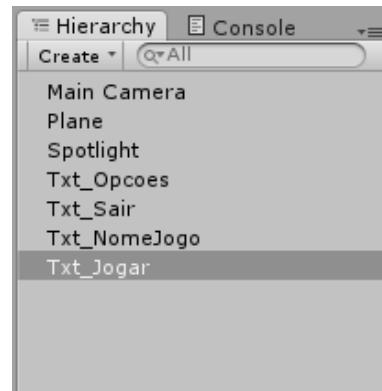


Figura 127 – Hierarquia de componentes do cenário

### 1.3.2 Console

Onde são exibidas todas as mensagens de erro, avisos ou até algum texto de *Debug* escolhido pelo programador.

Para inserir esta opção no ambiente de trabalho Unity pode aceder à barra de ferramentas, em *Window>Console* ou poderá fazê-lo através do atalho *Ctrl+Shift+C*.

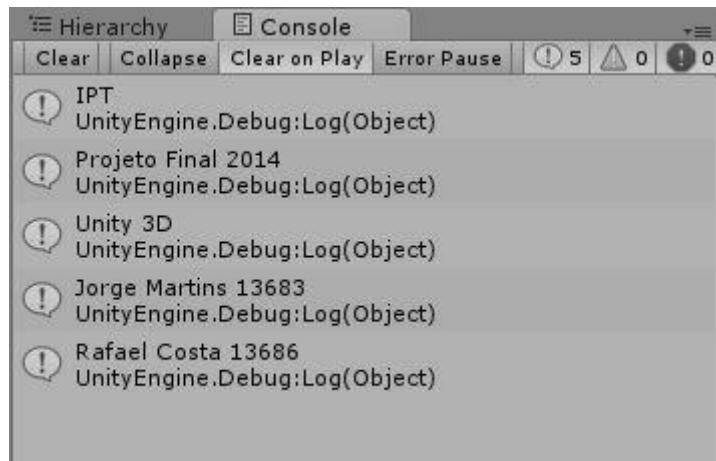


Figura 128 – Consola Unity

### 1.3.3 Inspector

Janela onde o utilizador pode visualizar as definições que o objeto inserido no cenário de jogo contém.

Nesta janela poderemos mover o objeto através de coordenadas, adicionar *scripts*, *sounds*, *colliders*, entre outras opções.

### 1.3.4 Project

Este separador contem todos os ficheiros e pastas pertencentes ao projeto.

A este conjunto de pastas e ficheiros dá-se o nome de *Assets*.

#### 1.3.4.1 Assets

Contém toda a informação referente ao projeto.

Dentro desta pasta encontram-se duas subpastas denominadas por *Standard Assets* e *Standard Assets (Mobile)*.

##### 1.3.4.1.1 Standard assets

Aqui encontram-se todos os elementos que o utilizador importou quando criou o projecto.

- Character controller

Pasta onde se encontra o tipo de controlo de jogo. Este controlo pode ser *3rd Person Controller* ou *First Person Controller*.

- Light cookies

Pasta onde se encontra os diversos tipos de texturas que o Unity disponibiliza para efeitos de luz. Estas texturas são utilizadas como uma máscara que determina o quão brilhante será a luz num determinado local.

- Light flares

Pasta que contém luzes bastante brilhantes possíveis de adicionar a um objecto. Estas servem para simular efeitos de luz a refletir na câmara de jogo.

- Particles

Pasta que contém as partículas que o Unity disponibiliza.

Estas partículas são pequenas animações que simulam alguns materiais como fumo, chamas, explosões, faíscas, fogo-de-artifício, neve a cair, entre muitas outras.

Para adicionar estas partículas é necessário apenas arrastar o efeito desejado para o objeto onde se pretende inserir a partícula.

- Physic materials

Pasta que contém os diferentes tipos de materiais possíveis de inserir nos objetos.

Estes materiais têm o intuito de ajustar a fricção e os efeitos de objetos a saltar ou arrastar quando estes colidem com outros objetos.

Os tipos de materiais existentes são madeira, gelo, material saltitante, metal e borracha.

- Projectors

Pasta que contém alguns tipos de projetores (lâmpadas de iluminação) que se podem aplicar a diferentes objetos.

Estes permitem projetar um determinado material sobre todos os objetos que o cruzem.

Para definir este tipo de material apenas é necessário arrastar o projetor desejado para o objeto ao qual se pretende aplicar.

- Scripts

Pasta onde se encontram todos os *Scripts* criados pelo Unity na criação do projeto.

- SkyBoxes

Pasta que contém as texturas relativas ao céu.

Estas texturas são um invólucro à volta de toda a cena, que exibe o céu.

- Terrain Assets

Pasta onde se encontram as diferentes texturas relativas ao terreno de jogo.

Estas texturas servem para o utilizador criar as suas próprias paisagens para o jogo.

- Tesselation shaders/Toon shading

Pasta que contém *Shaders*. Estes são pequenos *Scripts* que permitem configurar o modo como o hardware gráfico é moldado.

*Tesselation Shader* apenas estão presentes em versões do Unity superiores à 4 e é compatível apenas com DirectX11.

- Tree creator

Pasta que contém texturas para criação de árvores.

- Water (basics)

Pasta que contém os tipos de águas que podem ser adicionadas ao projeto.

#### **1.3.4.1.2 Standard assets (mobile)**

Nesta pasta encontramos todos os elementos que o utilizador importou quando procedeu à criação de um projeto para a vertente móvel (dispositivos móveis).

- Control setups

Pasta que contém os diferentes tipos de controlos de jogo que o desenvolvedor pode criar.

Estes controlos estão sempre visíveis no ecrã de jogo, para que o jogador consiga controlar o mesmo.

- **Prefabs**

Pasta que contém *Prefabs* para a versão *Mobile*.

Estes *Prefabs*, são objetos já preparados com as definições e componentes necessários para os seu funcionamento neste tipo de plataformas.

- **Scripts**

Pasta que contém os *Scripts* criados pelo Unity quando a criação do projeto, para a versão *Mobile*.

- **Shaders**

Pasta que contém os pequenos *Scripts Shaders*, a fim de configurar a forma como o hardware gráfico é moldado na plataforma móvel.

- **Textures**

Pasta que contém texturas que servem para controlo da versão *mobile* do jogo.

### 1.3.5 Scene

Local onde são organizados todos os objetos do cenário de jogo.

Nesta janela é possível alterar a posição da câmara de jogo, jogadores, assim como todos os objetos do cenário.

#### 1.3.5.1 Posicionamento dos objetos

Um objeto dentro do cenário de jogo (Scene), pode ser alterado relativamente ao seu posicionamento, rotação ou zoom.

Para efetuar estas alterações utilizamos as *Trasform Tools*, que contêm cada uma dessas opções.

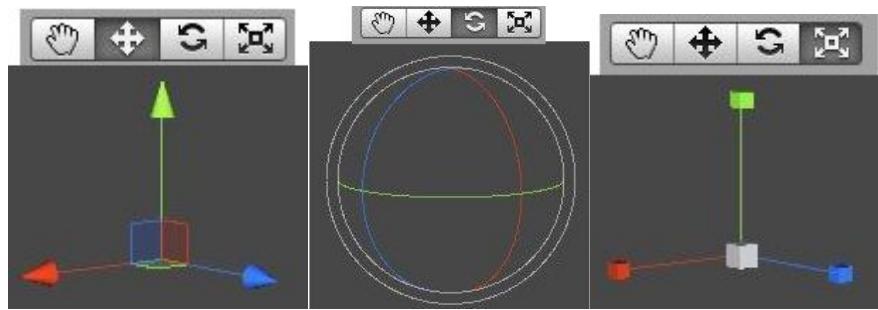


Figura 129 – Transform Tools

#### 1.3.5.2 Gizmo (eixo)

No canto superior direito da janela do cenário de jogo, encontramos a ferramenta *Gizmo*, um tipo de eixo.

Este mostra-nos a orientação da câmara em relação ao cenário, o que nos possibilita uma mudança rápida e fácil da mesma, podendo assim ter uma perspetiva diferente do cenário de jogo.

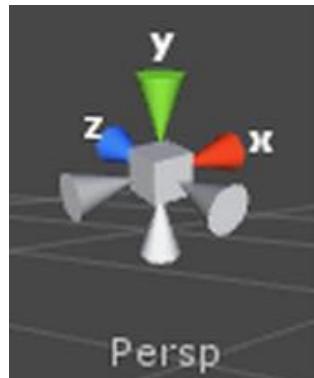


Figura 130 – Ferramenta Gizmo

#### 1.3.5.3 Main camera

Câmara que mostra o mundo do jogo ao jogador.

O desenvolvedor poderá inserir um número ilimitado de câmaras, sendo estas configuradas para processar em qualquer ordem ou lugar sobre o cenário, de modo a mostrar o que o utilizador pretende.

Esta vem pré-definida após a criação do projeto.

### 1.3.6 Game

Janela onde é possível visualizar o resultado final do jogo que está a ser desenvolvido.

O jogo é executado neste separador sempre que se clica sobre o botão *Play*, e mostrado segundo a perspetiva da câmara de jogo que se encontra no cenário de jogo.

### 1.3.7 Play/Pause/Step

Encontram-se no topo central do ambiente Unity ou na barra de menus *Edit*.

Estes três botões são utilizados para iniciar, pausar e correr o jogo passo a passo, respetivamente. Os botões *Pause* e *Step* ficam ativos quando se clica no botão *Play*.

Para sair do jogo é necessário clicar novamente sobre o botão *Play*.



Figura 131 – Botões Play/Pause/Stepa

### 1.3.8 Barra de estado

Barra situada no canto inferior esquerdo da janela Unity, que tem como funcionalidade mostrar erros de compilação, avisos ou mensagens de *Debug*.

Estes erros podem ser visualizados no componente em que este se encontra através de um duplo clique na mensagem.

## 1.4 Criação de um cenário de jogo

Neste capítulo explicaremos os vários processos e componentes necessários para a construção de um cenário de jogo em Unity.

### 1.4.1 Terrain

O terreno de jogo é um dos componentes essenciais para a construção de um cenário, pois é a base onde a maioria dos outros objetos irão assentar, exemplo de estradas, prédios, veículos, etc.

Este objeto pode ser adicionado através do menu *GameObject>Create Other>Terrain*.

Este objecto pode ser moldado, com as opções de subir o terreno, baixar terreno, alisar terreno, adicionar árvores, criar detalhes (relva, areia, etc.).

Para efetuar alterações neste objeto seleccionamos o objeto *Terrain* na janela *Hierarchy*, e deslocamo-nos ao separador *Inspector*. Nesta janela encontra-se o

componente *Terrain*(*Script*), onde se encontram as opções do terreno referidas anteriormente.

Após as modificações do terreno devemos utilizar uma *Directional Light*, para dar alguma luz ao cenário não o tornando demasiado escuro. Para a criação da mesma selecionamos o menu *GameObject>Create Other>Directional Light*.

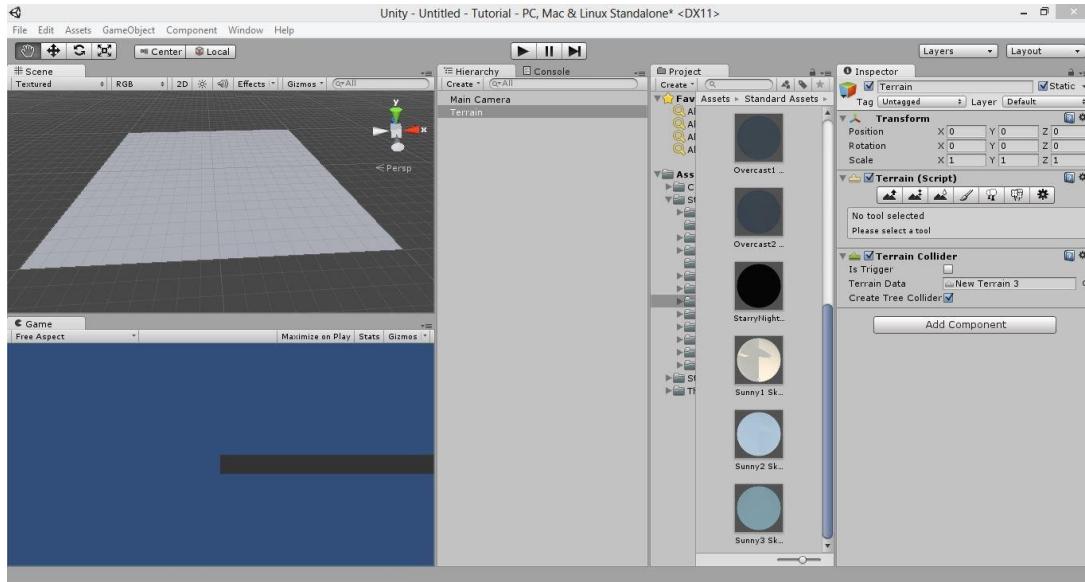


Figura 132 – Terreno do jogo e respetiva janela de definições

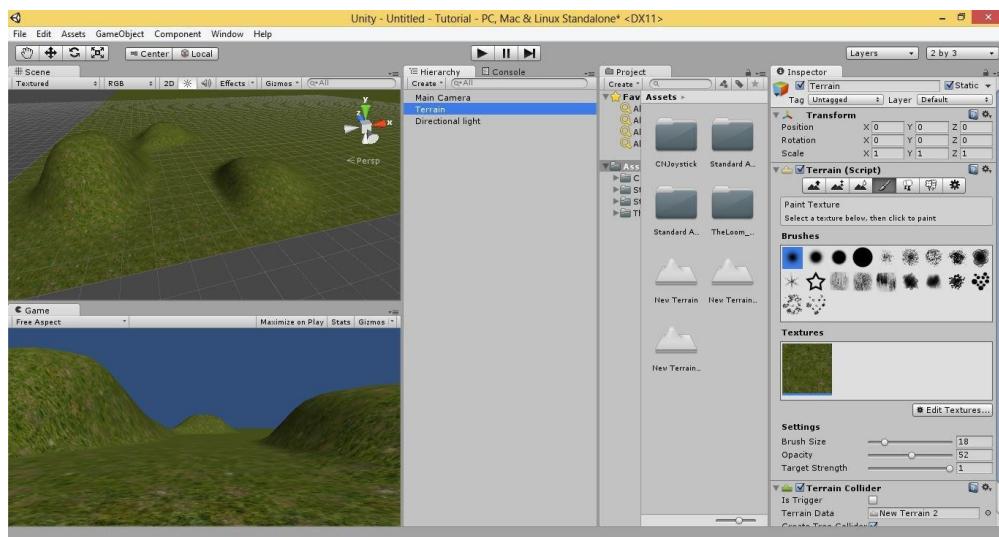


Figura 133 – Edição do terreno de jogo

#### 1.4.2 Sky Box

Para adicionar estas texturas referentes ao céu do cenário de jogo, dando-lhe assim um aspecto mais realista, selecionamos o menu *Edit>Render Settings*.

Na janela *Inspector* existem várias opções para alterarmos o céu do cenário de jogo, como podemos ver em *Skybox Material* (Fig. X). Existem ainda outros detalhes possíveis de adicionais, como o nevoeiro. Podendo definir-se a sua cor, densidade entre outras opções.

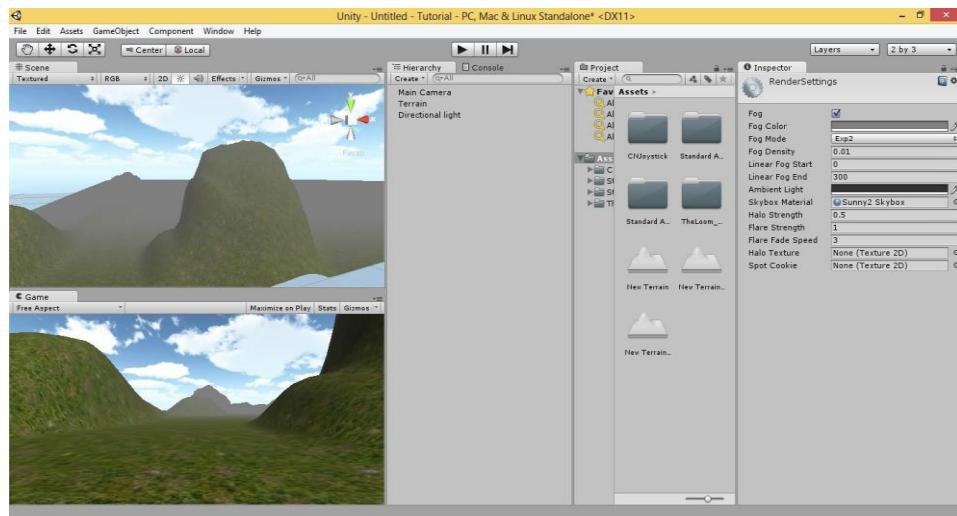


Figura 134 – Janela de definições da componente Sky Box

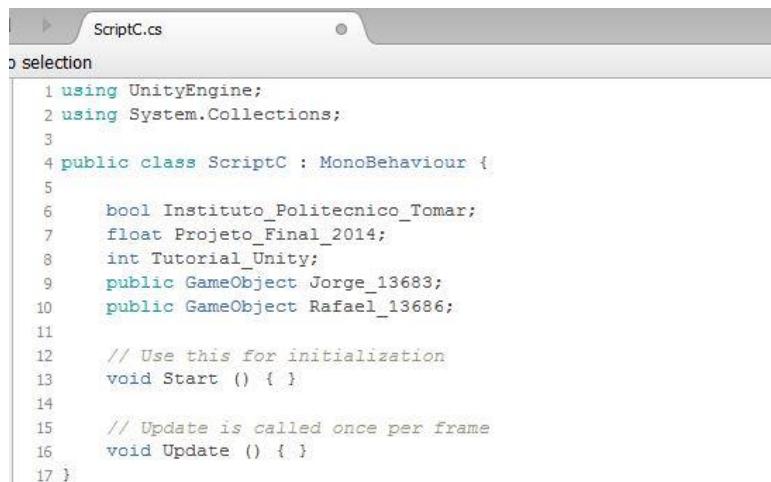
### 1.4.3 Scripts

Para a criação de Scripts em Unity acedemos ao menu *Assets>Create* e seleccionamos um dos Scripts disponíveis. Estes para executarem o código desenvolvido deverão estar associados a um determinado objeto. Para tal, apenas é necessário arrastar o Script para cima do objeto correspondente. Outra opção será através do menu *Hierarchy*, na janela *Inspector* clicar em *Add Component*, escrever o nome do Script desenvolvido e clicar sobre o mesmo.

Em Unity, os Scripts são executados por um *Software* denominado por *MonoDevelop*.

### 1.4.3.1 Estrutura de um Script

Os Scripts em Unity são estruturados por Bibliotecas, Classes, Variáveis e duas funções padrão, a função **void Start()** e **void Update()**.



```

1 using UnityEngine;
2 using System.Collections;
3
4 public class ScriptC : MonoBehaviour {
5
6     bool Instituto_Politecnico_Tomar;
7     float Projeto_Final_2014;
8     int Tutorial_Unity;
9     public GameObject Jorge_13683;
10    public GameObject Rafael_13686;
11
12    // Use this for initialization
13    void Start () { }
14
15    // Update is called once per frame
16    void Update () { }
17 }

```

Figura 135 – Exemplo de um script em C# para Unity

#### 1.4.3.1.1 Bibliotecas

Bibliotecas são uma coleção de serviços que o Unity disponibiliza para o programador poder produzir os seus *Scripts*.

Estas bibliotecas em Unity iniciam sempre por **using**.

#### 1.4.3.1.2 Variáveis

As variáveis em Unity são criadas antes da função **void Start()**.

Estas podem ser públicas ou privadas. As variáveis do tipo *public* podem ser visualizadas no ambiente Unity, através da janela *inspector*. Este tipo de variáveis são bastante interessante e úteis caso o utilizador pretenda associar vários objetos ao *Script*, porque ao criar uma variável *public GameObject*, podemos a partir do separador *inspector*, visualizar a variável *GameObject* criada e assim, arrastar o objeto pretendido para a respetiva caixa.

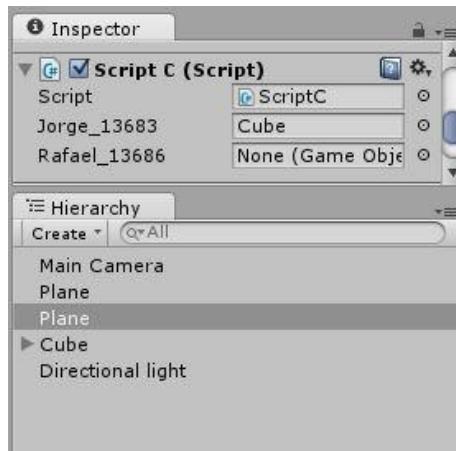


Figura 136 – Janela de definições das variáveis públicas referentes a um Script

#### 1.4.3.1.3 Funções

Para desenvolvimento em Unity existem duas funções principais, pois é nelas que recaem as principais instruções a executar no cenário de jogo, estas funções são:

- A função `void Start ()` é uma função executada sempre que o programa é iniciado, ou seja, sempre que iniciamos o jogo, todas as instruções que se encontram dentro desta função serão executadas.
- A função `void Update ()` é executada a cada frame do jogo.

Existem ainda outras funções que complementam as anteriores, estas são fundamentais para a interação dos diferentes objetos dentro do cenário de jogo.

- A função `void OnCollisionEnter (Collision colisao)`, serve para detetar quando um objeto colide com outro, onde a variável `colisao` contém as características do objeto em que colidiu.
- A função `void OnCollisionExit (Collision Exitcolisao)`, serve para detetar quando um objeto deixa de colidir com outro, onde a variável `Exitcolisao` contém as características do objeto em que colidiu.
- A função `void OnTriggerEnter (Collider Triggercolisao)`, serve para detetar a colisão com um objecto que tenha a propriedade de ser `trigger`, onde a variável `Triggercolisao` contém as características do objeto em que colidiu.
- A função `void OnTriggerExit (Collider Triggercolisao)`, serve para detetar quando um objeto deixa de colidir com outro que tenha a propriedade de ser

*trigger*, onde a variável Triggercolisao contem as características do objeto em que colidiu.

- A função `void OnGUI()`, serve para desenhar no ecrã através de *Scripts*. Esta função pode ser utilizada para que seja mostrado no ecrã de jogo algo que o desenvolvedor necessite, como mensagens ou valores para *debug*, ou o próprio jogador como barra de energia por exemplo.

#### 1.4.4 Fatores a ter em conta no desenvolvimento

Durante o desenvolvimento de um projeto em Unity, existem alguns fatores a ter em conta que poderão afetar o desempenho do jogo ou a jogabilidade do jogo por exemplo.

##### 1.4.4.1 Dimensão dos objetos

Em cenários de jogo muito extensos e com muitos objetos, não é recomendado o tamanho exagerado dos objetos (edifícios, estradas, pontes, carros, etc.).

Ao criarmos um cenário de jogo, com a exportação dos diferentes objetos para o cenário, é bastante comum estes ficarem demasiado grandes. Com o desenvolvimento do projeto e à medida que lhe são acrescentados objetos a complexidade deste aumenta, assim como os detalhes ou numero de objetos no ecrã de jogo. Devido a isso o jogo torna-se por vezes demasiado lento, o que pode ser um problema para jogadores com máquinas menos potentes capazes de correr o jogo ou até com a exportação para dispositivos móveis, pois não têm tanta capacidade. Ainda outro problema com a utilização de objetos com dimensões enormes é a incapacidade da placa gráfica desenhar todos esses objetos durante o jogo. Fazendo com que por vezes, durante o jogo o cenário não apareça por completo, aparecendo conforme o jogador se desloca.

Por isso, é recomendado que os objetos tenham tamanhos reduzidos. Para o redimensionamento dos objetos o desenvolvedor poderá recorrer às *Transform Tools*.

##### 1.4.4.2 Objetos com demasiado detalhe

Ao utilizarmos num cenário de jogo objetos que contenham texturas muito detalhadas poderá ser um problema.

Quando criamos um cenário grande como por exemplo uma cidade, existem muitos detalhes (passeios, estradas, prédios, carros, etc.), as texturas que dão o detalhe ao objeto não deverão ser muito elevadas, pois tornam o jogo por vezes demasiado lento retirando *performance* e piorando a jogabilidade.

Então é recomendado ao desenvolvedor que, diminua as texturas dos objetos. Recorrendo aos materiais do objeto na janela *Inspector*.

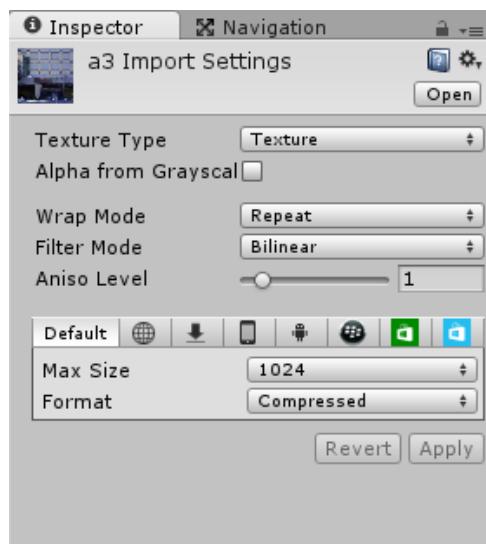


Figura 137 – Definições das texturas de um objeto

#### 1.4.4.3 Posição dos objetos em relação ao terreno do jogo

Devido ao motor de física do Unity, se um objeto tiver um *Rigidbody* com gravidade a sua posição em relação ao solo deverá ser positiva, isto é, o objeto deverá estar acima do solo. Caso contrário, este objeto “cairá” no vazio, para o infinito.