Zombie Village: Find and Kill

Relatório do Trabalho Prático

Licenciatura em Engenharia Informática

Computação Gráfica

2020/2021

Trabalho realizado por:

Diana Ferreira – 68938
Vítor Neto – 68717

Junho de 2021

Índice

Introdução 2
Objetivos do Trabalho 2
Estrutura do Trabalho 2
Cenário Simplificado 2
Metodologias 3
Desenvolvimento 4
Construção de Objetos 3D 4
Configuração de Câmara 8
Configurações de Luzes 9
Interações com a Cena 10
Animações 12
Conclusão 13
Melhorias 13
Autoavaliação 13
Bibliografia 14

Introdução

Objetivos do Trabalho

No âmbito da unidade curricular de Computação Gráfica, foi proposto aos alunos a elaboração de uma aplicação gráfica composta por elementos 3D num espaço 3D que fosse de encontro com todo o conteúdo lecionado ao longo do semestre, através do auxílio da biblioteca *three.js*. A aplicação deve demonstrar um cenário prático, que contempla os seguintes objetivos/requisitos: O primeiro requisito é a construção de objetos 3D complexos, utilizando as primitivas básicas de *three.js*, e texturizados, utilizando texturas importadas. O segundo requisito é a configuração de câmara, onde é possível alternar entre uma configuração de câmara em perspetiva e uma câmara ortográfica. O terceiro requisito é a configuração de luzes, onde devem existir diversos tipos de luzes como *PointLight*, *DirectionalLight*, *AmbientLight* e *SpotLight* sendo possível desligar cada tipo de luz individualmente. O quarto requisito é a interação com a cena, aqui o utilizador deve ser capaz de interagir com a cena através de dispositivos de *input*, para isto, foi utilizada a biblioteca *PointerLockControls*. O quinto requisito são as animações: os objetos devem ser animados, com animações importadas, de forma a demonstrar um conceito a ilustrar.

Estrutura do Trabalho

O relatório apresentado está estruturado em quatro capítulos (Introdução, Desenvolvimento, Conclusão e Bibliografia) e onze subcapítulos (Objetivos do Trabalho, Estrutura do Trabalho, Cenário, Metodologias, Construção de Objetos 3D, Configuração de Câmara, Configuração de Luzes, Interação com a Cena, Animações, Melhorias e Autoavaliação).

No capítulo da introdução constam os objetivos do trabalho, a estrutura do trabalho, o cenário, e a metodologia, onde descrevemos os objetivos/requisitos pedidos no protocolo proposto pelos docentes e todo o material e ferramentas utilizadas para o desenvolvimento deste trabalho. Para além disso temos a descrição do cenário desenvolvido.

No capítulo do desenvolvimento estão enumerados os requisitos que vão ser abordados e explicados detalhadamente no contexto do funcionamento da aplicação. Foram apresentados *printscreens* do código desenvolvido para uma melhor explicação de cada requisito.

No capítulo da conclusão, foram abordadas as considerações retiradas de todo o desenvolvimento desta aplicação e as dificuldades encontradas durante este processo. Para além disso, também constam as melhorias e a autoavaliação consoante a tabela fornecida no protocolo do trabalho.

No quarto e último capítulo, são expostas as referências que foram acedidas para tirar informações úteis para ultrapassar os obstáculos que foram surgindo.

Cenário Simplificado

No desenvolvimento desta aplicação, serão utilizadas as funcionalidades permitidas pela biblioteca *three.js* para uma demonstração das suas capacidades. O cenário envolverá quatro câmaras de perspetiva cujo controlo pelo cenário será permitido com a utilização das funcionalidades da biblioteca *PointerLockControls.js*, sendo também permitida a alternação de uma delas (câmara do personagem) entre uma câmara ortográfica. O cenário em questão envolverá objetos 3D desenvolvidos e importados pelos alunos. Em relação à configuração das luzes, o cenário será iluminado por uma *DirectionalLight*, uma *AmbientLight*, duas *SpotLights* bem como por duas luzes *PointLight* e será permitido desligar/ligar cada uma delas individualmente. Para movimentar o personagem, são utilizadas as teclas 'WASD' bem como o movimento do rato/setas para rodar a câmara. Será também permitido correr com a tecla 'Shift' e disparar com o botão esquerdo do rato/tecla 'F'. Neste trabalho estarão também presentes algumas animações, como, por exemplo, os zombies a vaguearem, a câmara cinematográfica e a árvore do ecrã de carregamento.

Assim, temos o desenvolvimento de uma aplicação cujo cenário em que decorre a ação é uma vila infestada por zombies, onde o personagem principal é um sobrevivente com o objetivo de eliminar os zombies espalhados pela vila, para este efeito, utiliza uma arma.

Neste subcapítulo não se encontra toda a informação sobre o cenário, visto que é apenas uma contextualização, sendo que o cenário será abordado mais detalhadamente no capítulo do desenvolvimento.

Metodologias

A aplicação foi desenvolvida com recurso ao software utilizado ao longo das aulas práticas, *Visual Studio Code*.

Para o desenvolvimento do presente relatório utilizou-se o *Microsoft Office Word* e o *Adobe Acrobat Reader* para a leitura do mesmo em formato pdf.

Devido à pandemia, para facilitar as reuniões tidas entre os integrantes do grupo, foi utilizado o *Discord*.

Por último, foi utilizado todo o material teórico disponibilizado pelos docentes, como os acetatos e a bibliografia recomendada, de maneira a servir de fundamentação na explicação dos objetivos que vão ser abordados mais pormenorizadamente no capítulo do desenvolvimento.

Desenvolvimento

A computação gráfica é a área que se preocupa com todos os aspetos da produção e representação de informações através de imagens, animações e vídeos usando computadores. Pode ser definida como o conjunto de métodos e técnicas para criação e manipulação de imagens. Ela pode possuir uma infinidade de aplicações para diversas áreas, desde a própria informática, ao produzir interfaces gráficas para software, sistemas operacionais e sites na Internet, quanto para produzir animações e jogos.

Em relação à estruturação deste capítulo, constam subcapítulos que vão ser retratados em seguida, onde são explicados e demonstrados os métodos tomados para o desenvolvimento de cada um dos requisitos, tal como a especificação de tudo o que foi feito dentro de cada um.

Como dito anteriormente, o cenário retrata uma pequena vila infestada por zombies, que contém vários objetos 3D importados e sete objetos 3D elaborados por nós (banco, cruzes, vasos, automóvel, cenouras, árvores e cercas), duas luzes *PointLight* (uma no cemitério e uma no candeeiro central), duas luzes *SpotLight* (faróis do automóvel), uma *AmbientLight* e uma *DirectionalLight*. Utilizamos a biblioteca *PointerLockControls*, tecla 'M' para abrir minimapa, teclas 'WASD' para movimentar o personagem, teclas das setas para rodar o personagem, botões do rato e tecla 'F' para disparar a arma, teclas 'OLP' para desligar cada uma das respetivas luzes individualmente, teclas de espaço e 'C' para subir e descer, respetivamente, e 'SHIFT' para aumentar a velocidade de movimentação do personagem. Para as animações foram importados zombies a vaguearem pela cidade, a câmara cinematográfica e a árvore do ecrã de carregamento.

Construção de Objetos 3D

Na construção de objetos 3D, que é uma forma que pode ser definida como um sólido ou objeto que tem três dimensões, a largura, altura e profundidade, recorremos a uma das áreas fundamentais da computação gráfica, a modelação. A modelação trata-se de fazer uma especificação, normalmente matemática (desenho vetorial), da forma e aparência de um objeto, e essa mesma definição deve poder ser armazenada num computador. Nesta área, os objetos devem ser fielmente representados, não devendo existir ambiguidades, as representações devem ser únicas, universais e precisas, e devem ser compactas para que os processamentos sejam rápidos.

Para a construção dos objetos que formam o nosso cenário, recorreu-se ao método de instanciação de primitivas, que consiste na representação das primitivas fornecidas pela biblioteca 3D utilizada, que, no nosso caso, é o *three.js*. Este método de instanciação de primitivas é uma das representações mais usadas devido à sua grande simplicidade e flexibilidade, tendo por base a definição de objetos geométricos tridimensionais, as primitivas, que possuem atributos, os parâmetros, cujos valores são definidos pelo utilizador no momento da criação de uma nova instância.

Nalguns objetos criados para a cena, foram utilizados métodos de mapeamento de texturas. Tendo em conta que as superfícies “no mundo real” são muito complexas, uma das maneiras para obtermos cenas sintetizadas por computador altamente realistas é através da utilização de texturas, que simulam efeitos de luz avançados sem necessitar de gastar computacionalmente recursos que levariam a que fosse possível obter isto. Para efetuar o mapeamento de texturas, existem três tipos de textura, a textura em si (*environment mapping*), o *bump map* e o *normal map* (*texture map*). O *environment mapping* basicamente aplica as cores ao objeto, o *normal map* é utilizado para calcular de que forma as luzes são refletidas e refratadas e o *bump map* serve para adicionar pequenos “solavancos” nos polígonos sem alterar a geometria do mesmo, para acrescentar realismo ao objeto. Para a criação dos nossos objetos, foi apenas utilizada a primeira abordagem, ou seja, o *environment mapping*.

Para além de todos os objetos importados na cena, para proporcionar um nível elevado de vivacidade e realismo à vila, também foram criados objetos 3D, sendo estes: banco, automóvel, vasos, cercas, cenouras, cruzes e árvores.

Para uma melhor compreensão da abordagem utilizada e de como todo o processo de criação de um objeto se desenvolve, vamos selecionar o objeto 3D automóvel e banco para demonstrar e explicar este processo, visto que são os objetos mais complexos da cena.

Vamos começar por explicar o processo de criação do automóvel.

```
/*---------------Carro---------------*/
function Carro(){
    const carro = new THREE.Group();
    const carroTextureLoader = new THREE.TextureLoader(loadingManager);
    var texturaCarro= carroTextureLoader.load("Textures/Carro/carroMetal.jpg");
    var vidroCarro= carroTextureLoader.load("Textures/Carro/glass.jpg");
```

Figura 1 - Criação do Objeto Automóvel (Parte 1)

Como podemos observar na Figura 1, para que o objeto possa ser instanciado, utilizamos uma função "Carro()", onde começamos por criar um *THREE.Group* para que as instâncias das primitivas possam ser adicionadas e unidas hierarquicamente como um todo ao grupo. Foi também criado um *TextureLoader* para que possam ser importadas as texturas e armazenadas numa variável.

```
const basePreta = new THREE.Mesh(
    new THREE.BoxGeometry(2, 0.5, 5.6),
    new THREE.MeshPhongMaterial({color: 0x424140})
);
basePreta.position.set(0, 1, 0.7);
const baseRosa = new THREE.Mesh(
    new THREE.BoxGeometry(1.8, 0.4, 1.6),
    new THREE.MeshPhongMaterial({map: texturaCarro})
);
baseRosa.position.set(0, 1.4, -1.2);
const baseRosaMeio = new THREE.Mesh(
    new THREE.BoxGeometry(1.75, 1.6, 1.95),
    new THREE.MeshPhongMaterial({map: texturaCarro})
);
baseRosaMeio.position.set(0, 1.7, 0);
const traseira1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.2, 0.4, 3.1),
    new THREE.MeshPhongMaterial({map: texturaCarro})
);
traseira1.position.set(0.8, 1.4, 1.948);
const traseira2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.2, 0.4, 3.1),
    new THREE.MeshPhongMaterial({map: texturaCarro})
);
traseira2.position.set(-0.8, 1.4, 1.948);
const traseira3 = new THREE.Mesh(
    new THREE.BoxGeometry(0.3, 0.4, 1.4),
    new THREE.MeshPhongMaterial({map: texturaCarro})
);
traseira3.position.set(0, 1.4, 3.348);
traseira3.rotation.set(0, Math.PI/2, 0);
```

Figura 2 - Criação do Objeto Automóvel (Parte 2)

```
const farol1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.3, 0.37, 1.4),
    new THREE.MeshPhongMaterial({color: 0xEAED98})
);
farol1.position.set(0.74, 1.4, -1.35);
const farol2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.3, 0.37, 1.4),
    new THREE.MeshPhongMaterial({color: 0xEAED98})
);
farol2.position.set(-0.74, 1.4, -1.35);
const radiador = new THREE.Mesh(
    new THREE.BoxGeometry(0.5, 0.15, 0.5),
    new THREE.MeshPhongMaterial({color: 0x424140})
);
radiador.position.set(0, 1.3, -1.8);
const retrovisor1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.4, 0.15, 0.2),
    new THREE.MeshPhongMaterial({color: 0x8B8A89})
);
retrovisor1.position.set(0.9, 1.55, -0.8);
const retrovisor2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.4, 0.15, 0.2),
    new THREE.MeshPhongMaterial({color: 0x8B8A89})
);
retrovisor2.position.set(-0.9, 1.55, -0.8);
```

Figura 3 - Criação do Objeto Automóvel (Parte 3)

Em relação à Figura 2, para começar a instanciação das primitivas, dividimos o automóvel por partes: a "basePreta", que representa o para-choques; a "baseRosa", que representa o capô; a "baseRosaMeio", que é o *cockpit*; e as "traseira1", "traseira2" e "traseira3", que juntas formam a mala. Em cada uma delas utilizamos uma *BoxGeometry* cujas dimensões variam de acordo com o que representam e uma *MeshPhongMaterial*, que representa a textura aplicada ao carro. Decidimos utilizar a *MeshPhongMaterial*, visto que esta utiliza o modelo de reflexão de *Phong*, permitindo que a luz interaja com a textura do objeto. Para que as primitivas ocupem os lugares pretendidos no contexto do grupo, foram modificados os parâmetros das suas posições, efetuando translações das suas coordenadas. É de notar que estas translações modificam a posição das primitivas dentro do contexto das coordenadas do grupo e não das coordenadas mundo.

Em relação à Figura 3, são demonstradas as partes: "farol1" e "farol2", que representam os faróis dianteiros; "radiador", que representa o radiador; e "retrovisor1" e "retrovisor2", que representam os retrovisores. Foi seguida a mesma ordem de ideias descrita no parágrafo relativo à Figura 2, no entanto, na *MeshPhongMaterial*, as texturas foram substituídas por cores.

Em relação à Figura 4, são demonstradas as partes: “rodaTraseira” e “rodaDianteira”, que representam dois cilindros que vão formar as quatro rodas; e os “pneu1”, “pneu2”, “pneu3” e “pneu4”, que representam os quatro pneus. Foi seguida a mesma ordem de ideias descrita no parágrafo relativo à Figura 2 e 3, no entanto, no caso das rodas foram utilizadas primitivas correspondentes a um cilindro (*CylinderGeometry*) e, no caso dos pneus, foram utilizadas primitivas correspondentes a um *donut* (*TorusGeometry*).

```
const rodaTraseira = new THREE.Mesh(
    new THREE.CylinderGeometry(0.25, 0.25, 2.4, 30),
    new THREE.MeshPhongMaterial({color: 0xFFFFFF})
);
rodaTraseira.position.set(0, 1, -1.2);
rodaTraseira.rotation.set(Math.PI/2, 0, Math.PI/2);
const rodaDianteira = new THREE.Mesh(
    new THREE.CylinderGeometry(0.25, 0.25, 2.4, 30),
    new THREE.MeshPhongMaterial({color: 0xFFFFFF})
);
rodaDianteira.position.set(0, 1, 2.3);
rodaDianteira.rotation.set(Math.PI/2, 0, Math.PI/2);
const pneu1 = new THREE.Mesh(
    new THREE.TorusGeometry(0.35, 0.15, 30, 100),
    new THREE.MeshPhongMaterial({color: 0x000000})
);
pneu1.position.set(1.1, 0.97, -1.19);
pneu1.rotation.set(0, Math.PI/2,0);
const pneu2 = new THREE.Mesh(
    new THREE.TorusGeometry(0.35, 0.15, 30, 100),
    new THREE.MeshPhongMaterial({color: 0x000000})
);
pneu2.position.set(-1.1, 0.97, -1.19);
pneu2.rotation.set(0, Math.PI/2,0);
const pneu3 = new THREE.Mesh(
    new THREE.TorusGeometry(0.35, 0.15, 30, 100),
    new THREE.MeshPhongMaterial({color: 0x000000})
);
pneu3.position.set(1.1, 0.97, 2.3);
pneu3.rotation.set(0, Math.PI/2,0);
const pneu4 = new THREE.Mesh(
    new THREE.TorusGeometry(0.35, 0.15, 30, 100),
    new THREE.MeshPhongMaterial({color: 0x000000})
);
pneu4.position.set(-1.1, 0.97, 2.3);
pneu4.rotation.set(0, Math.PI/2,0);
```

Figura 4 - Criação do Objeto Automóvel (Parte 4)

```
const paraBrisas = new THREE.Mesh(
    new THREE.BoxGeometry(1.4, 0.8, 0.1),
    new THREE.MeshPhongMaterial({map: vidroCarro})
);
paraBrisas.position.set(0, 1.95, -0.96);
const vidro1 = new THREE.Mesh(
    new THREE.BoxGeometry(1.4, 0.65, 0.1),
    new THREE.MeshPhongMaterial({map: vidroCarro})
);
vidro1.position.set(0.86, 2, 0);
vidro1.rotation.set(0, Math.PI / 2, 0);
const vidro2 = new THREE.Mesh(
    new THREE.BoxGeometry(1.4, 0.65, 0.1),
    new THREE.MeshPhongMaterial({map:vidroCarro})
);
vidro2.position.set(-0.86, 2, 0);
vidro2.rotation.set(0, Math.PI / 2, 0);
const retrovisorVidro1 = new THREE.Mesh(
    new THREE.PlaneGeometry(0.18, 0.1, 10, 10),
    new THREE.MeshPhongMaterial({map:vidroCarro})
);
retrovisorVidro1.position.set(-1, 1.55, -0.69);
const retrovisorVidro2 = new THREE.Mesh(
    new THREE.PlaneGeometry(0.18, 0.1, 10, 10),
    new THREE.MeshPhongMaterial({map:vidroCarro})
);
retrovisorVidro2.position.set(1, 1.55, -0.69);
const farolTras1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.2, 0.3, 0.3),
    new THREE.MeshPhongMaterial({color: 0xFF0000})
);
farolTras1.position.set(-0.74, 1.415, 3.35);
const farolTras2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.2, 0.3, 0.3),
    new THREE.MeshPhongMaterial({color: 0xFF0000})
);
farolTras2.position.set(0.74, 1.415, 3.35);
```

Figura 5 - Criação do Objeto Automóvel (Parte 5)

Em relação à Figura 5, são demonstradas as partes: “paraBrisas”, que representa o para-brisas; “vidro1” e “vidro2”, que representam os vidros das portas; “retrovisorVidro1” e “retrovisorVidro2”, que representam os vidros dos retrovisores; e “farolTras1” e “farolTras2”, que representam os faróis traseiros. Foi seguida a mesma ordem de ideias descrita no parágrafo relativo às Figuras 2, 3 e 4, no entanto, no caso dos vidros dos retrovisores, foram utilizadas primitivas correspondentes a um plano (*PlaneGeometry*).

Em relação à Figura 6, são demonstradas as partes: “targetFarol1” e “targetFarol2”, que foram utilizados para auxiliar a orientação das luzes dos faróis; e “farolLuz1” e “farolLuz2”, que são as luzes dos faróis. Nesta figura, são criadas luzes, utilizando luzes referentes a holofotes (*SpotLight*), porque achamos que são as que melhor representam os faróis de um carro. A definição das luzes será abordada detalhadamente no subcapítulo “Configurações de Luzes”. Os *targets* foram definidos para auxiliar na orientação das luzes, não sendo importante a sua geometria, tamanho e textura, sendo que utilizamos uma *MeshBasicMaterial*, porque a reflexão nestes objetos não interessa, existindo só para controlar o local para onde as luzes estão a direcionar.

```
const targetFarol1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.001, 0.001, 0.001),
    new THREE.MeshBasicMaterial({color: 0xffffff})
);
farol1.add(targetFarol1);
targetFarol1.position.set(0, 0, -2);

const targetFarol2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.001, 0.001, 0.001),
    new THREE.MeshBasicMaterial({color: 0xffffff})
);
farol2.add(targetFarol2);
targetFarol2.position.set(0, 0, -2);

const farolLuz1 = new THREE.SpotLight(0xFFFFFF, 2, 18, Math.PI/5);
farolLuz1.castShadow = true;
farolLuz1.target = targetFarol1;

const farolLuz2 = new THREE.SpotLight(0xFFFFFF, 2, 18, Math.PI/5);
farolLuz2.castShadow = true;
farolLuz2.target = targetFarol2;
```

Figura 6 - Criação do Objeto Automóvel (Parte 6)

Finalmente, adicionamos todas as partes ao carro, exceto as luzes dos faróis, que foram adicionadas aos respetivos faróis (Figura 7). O desenvolvimento deste objeto 3D, teve como resultado final (Figura 8):

```
carro.add(radiador);
carro.add(farol1);
farol1.add(farolLuz1);
farol2.add(farolLuz2);
```

Figura 7- Criação do Objeto Automóvel (Parte 7)

Figura 8 - Resultado Objeto 3D Automóvel

Agora, vamos explicar o processo de criação do banco, que demonstra detalhadamente o funcionamento das hierarquias de grupo e que seguirá a mesma ordem de ideias da criação do objeto 3D automóvel.

Como podemos observar na Figura 9, para que o objeto possa ser instanciado, utilizamos uma função "Banco()", onde começamos por criar um *THREE.Group* para que as instâncias das primitivas possam ser adicionadas e unidas hierarquicamente como um todo ao grupo. Para a instanciação das primitivas, dividimos o banco por partes: "suporteEsquerdo", que representa a lateral esquerda do banco e é constituída pelos quatro paralelepípedos; "suporteDireito", que representa a lateral direita do banco, que, tal como o suporte esquerdo, é constituída por quatro paralelepípedos. Estes paralelepípedos foram adicionados aos respetivos suportes do banco.

```
function Banco(){
    const banco = new THREE.Group();
    //Suporte Esquerdo
    const suporteEsquerdo = new THREE.Group();
    const paralelepipedo1 = new THREE.Mesh(
        new THREE.BoxGeometry(0.1, 0.1, 1),
        new THREE.MeshPhongMaterial({color: 0xB3B3B3})
    );
    const paralelepipedo2 = new THREE.Mesh(
        new THREE.BoxGeometry(0.1, 0.1, 1),
        new THREE.MeshPhongMaterial({color: 0xB3B3B3})
    );
    const paralelepipedo3 = new THREE.Mesh(
        new THREE.BoxGeometry(0.1, 0.1, 1),
        new THREE.MeshPhongMaterial({color: 0xB3B3B3})
    );
    const paralelepipedo4 = new THREE.Mesh(
        new THREE.BoxGeometry(0.1, 0.1, 1),
        new THREE.MeshPhongMaterial({color: 0xB3B3B3})
    );
    suporteEsquerdo.add(paralelepipedo1);
    suporteEsquerdo.add(paralelepipedo2);
    suporteEsquerdo.add(paralelepipedo3);
    suporteEsquerdo.add(paralelepipedo4);
    paralelepipedo1.position.set(0, 2, 0);
    paralelepipedo2.position.set(0 ,1.6 ,0);
    paralelepipedo3.position.set(0 ,1.9 ,0.5);
    paralelepipedo3.scale.set(1, 1, 1.7);
    paralelepipedo3.rotation.set(Math.PI/2,0 ,0 );
    paralelepipedo4.position.set(0 ,1.55 ,-0.5);
    paralelepipedo4.rotation.set(Math.PI/2,0 ,0 );
```

Figura 9 - Criação do Objeto Banco (Parte 1)

```
const base = new THREE.Group();
const tabua1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.1, 0.1, 1),
    new THREE.MeshPhongMaterial({color: 0x9E5D14})
);
const tabua2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.1, 0.1, 1),
    new THREE.MeshPhongMaterial({color: 0x9E5D14})
);
const tabua3 = new THREE.Mesh(
    new THREE.BoxGeometry(0.1, 0.1, 1),
    new THREE.MeshPhongMaterial({color: 0x9E5D14})
);
base.add(tabua1);
base.add(tabua2);
base.add(tabua3);
tabua1.rotation.set(0, Math.PI / 2, 0);
tabua1.scale.set(2.1, 0.4, 2.05);
tabua1.position.set(1, 1.6, 0);
tabua2.rotation.set(0, Math.PI / 2, 0);
tabua2.scale.set(2.1, 0.4, 2.05);
tabua2.position.set(1, 1.6, 0.3);
tabua3.rotation.set(0, Math.PI / 2, 0);
tabua3.scale.set(2.1, 0.4, 2.05);
tabua3.position.set(1, 1.6, -0.3);
base.position.set(0, 0, 0);
```

Figura 10 – Criação do Objeto Banco (Parte 2)

Em relação à Figura 10, é demonstrada a construção da base do banco, sendo constituída por três tábuas que vão ser adicionadas ao grupo "base". O encosto do banco segue exatamente a mesma ordem de ideias da construção da base, sendo, por isso, omitida essa parte da explicação.

Na Figura 11, adicionamos os subgrupos ao grupo principal formando assim o banco, obtendo o resultado final apresentado na Figura 12.

```
banco.add(suporteEsquerdo);
banco.add(suporteDireito);
banco.add(encosto);
banco.add(base);
```

Figura 11 - Criação do Objeto Banco (Parte 3)

Figura 12 - Resultado Objeto 3D Banco

Configuração de Câmara

Em CG, todo o processo de visualização tem subjacente o denominado modelo de câmara virtual, em que uma câmara fotográfica virtual é posicionada e orientada no espaço do mundo da cena. Este conceito é fundamental, pois representa o ponto de vista sobre o qual os objetos da cena vão ser visualizados e define um novo referencial: sistema de coordenadas de câmara.

O volume de visualização representa a delimitação da região do espaço do mundo que se pretende visualizar e projetar no plano de visualização.

Na projeção em perspetiva, o volume de visualização é definido pelo tronco de pirâmide infinita, com vértice no centro de projeção (VRP) e lados sobre a janela de visualização. Ao proceder ao recorte da cena sobre este volume de visualização antes da projeção, não ficam projetados objetos que se encontrem atrás do centro de projeção. O Volume de Visualização contém tudo o que é 'visível' pela câmara.

Na projeção ortogonal, a projeção e recorte são mais simples. Esta é definida pelo paralelepípedo infinito, passando pelos lados da janela de visualização de arestas paralelas à direção VPN. O FOV (*field-of-view*) nesta vista é 0.

Os planos de recorte delimitam o que a câmara desenha, sendo o volume entre os planos de recorte que determinam o que a câmara vê. A este volume chama-se também de *frustum*. Se um objeto for "cortado" por um dos planos de recorte, a sua porção que fica do lado de fora do volume de visualização é ignorada e não é desenhada pela câmara.

No cenário, tal como é requisitado no protocolo, temos a existência de uma configuração de câmara em perspetiva (câmara do personagem) que alterna com a configuração da câmara ortográfica (minimapa). Para além da câmara em perspetiva do personagem, também temos uma câmara em perspetiva para o ecrã de carregamento, uma para o ecrã de final de jogo e uma cinematográfica.

```
camera = new THREE.PerspectiveCamera(90, 1600/920, 0.1, 1000); //1280/720
camaraOrtografica = new THREE.OrthographicCamera(-100, 100, 100, -100, 0.1, 1000);
camaraOrtografica.position.set(0, 10, 0);
camaraOrtografica.rotation.set(-Math.PI/2, 0, 0);
```

Figura 13 - Configuração das Câmaras do Personagem e do Minimapa

A Figura 13 apresenta a configuração das câmaras do personagem (câmara em perspetiva) e do minimapa (câmara ortográfica). Para configurar a câmara do personagem, são passados como parâmetros: o *field-of-view*, que indica o tamanho do ecrã que é desenhado pela câmara; o *aspect ratio*, que são a quantidade de pixéis que constituem o ecrã que é desenhando pela câmara; e os últimos dois parâmetros representam a distância dos planos *near* e *far* à câmara. Para configurar a câmara do Minimapa, são passados como parâmetros: os valores correspondentes aos planos de recorte da esquerda, direita, baixo e cima; e os valores da distância dos planos *near* e *far* à câmara. Para que a aplicação reconheça a câmara que deve ser apresentada, esta deve ser passada como parâmetro da função *render*, correspondente ao renderizador. A forma como esta alternância (Figura 14) é efetuada, é explicada detalhadamente no subcapítulo "Interação com a Cena". Na Figura 16 são apresentadas as configurações das câmaras em perspetiva dos ecrãs de carregamento e de final de jogo. A câmara cinematográfica é abordada no subcapítulo "Animações".

```
if(c == 0)
renderer.render(scene, camera);
else if(c == 1)
renderer.render(scene, camaraOrtografica);
else if(c == 2)
renderer.render(vitoria, cameraVitoria);
```

Figura 14 - Alternância das Câmaras

Figura 15 - Minimapa

```
var camera = new THREE.PerspectiveCamera(90, 1600/920, 0.1, 1000);
var cameraVitoria = new THREE.PerspectiveCamera(90, 1600/920, 0.1, 1000);
```

Figura 16 - Configuração das Câmaras em Perspetiva de Ecrã de Carregamento e de Final de Jogo

Configurações de Luzes

A interação da luz com os materiais dá a origem à cor. Esta é uma energia luminosa que é refletida num determinado ponto e essa energia vai ao encontro da câmara virtual (no caso da síntese de imagem) e através dessa luz que chega à câmara virtual é que conseguimos calcular a cor de um determinado pixel. O resultado é diferente consoante a posição da câmara, objeto e fonte de luz.

Na biblioteca *three.js* são disponibilizados quatro tipos de luz:

- *PointLight*: Fonte de luz localizada numa posição do espaço que radia igualmente em todas as direções, com atenuação da intensidade luminosa em função da distância;
- *SpotLight*: Fonte de luz semelhante à "*PointLight*", em que a emissão de luz se encontra limitada a um ângulo sólido de abertura variável, cujo eixo é a direção da emissão. A intensidade luminosa é atenuada radialmente em função do ângulo entre os raios luminosos emitidos e a direção central da emissão, podendo também ser atenuada com a distância – diminuindo com esta. Para a definir temos de utilizar a posição, cor e o ângulo;
- *AmbientLight*: Fonte de luz sem posição nem direção, de intensidade constante, independente da direção e sem atenuação;
- *DirecionalLight*: Fonte de luz sem localização precisa (ou no infinito), sem atenuação da intensidade luminosa com a distância e cujos raios luminosos são paralelos, possuindo uma direção precisa, muito utilizada na modelação da luz solar.

No cenário, tal como é requisitado no protocolo, temos a existência de diversos tipos de luzes (Duas *PointLight*, uma *DirectionalLight*, uma *AmbientLight* e duas *SpotLight*), sendo possível desligar cada tipo de luz individualmente. A forma como estas são desligadas/ligadas é abordada no subcapítulo "Interação com a Cena". Relativamente às *PointLight*, estas estão presentes na lanterna do cemitério ("luzLanterna") e no centro da cena ("light"), como podemos observar na Figura 17. A configuração desta fonte de luz simplificada tem como parâmetros: cor, que define a cor que a luz irá emitir; intensidade, que define a força com que a luz será transmitida; e a distância, que define o ponto de decaimento da luz. Colocar a propriedade *castShadow* da luz a *true*, possibilita que os objetos nos quais a luz incide emitam uma sombra (esta propriedade está presente em todas as fontes de luz simplificadas exceto na *AmbientLight*). Já as propriedades *shadow.camera.near* e *shadow.camera.far* indicam a intensidade da sombra que vai ser emitida. Em relação à *AmbientLight*, esta é global e tem como parâmetros: cor, que define a cor que a luz irá emitir; e a intensidade, que define a força com que a luz será transmitida. Relativamente à *DirectionalLight*, esta é global e tem como parâmetros: cor, que define a cor que a luz irá emitir; e a intensidade, que define a força com que a luz será transmitida. Por último, como podemos observar na Figura 18, as duas *SpotLights* ("farolLuz1" e "farolLuz2") estão presentes nos faróis do automóvel. Estas têm como parâmetros: cor, que define a cor que a luz irá emitir; a intensidade, que define a força com que a luz será transmitida; a distância, que define o ponto de decaimento da luz; e o ângulo de dispersão da luz.

```
ambientLight = new THREE.AmbientLight(0xffffff, 0.3);

scene.add(ambientLight);

directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
directionalLight.castShadow = true;
scene.add(directionalLight);

light = new THREE.PointLight(0xffffff, 0.8, 18);
light.position.set(6, 5, -3);
light.castShadow = true;
light.shadow.camera.near = 0.1;
light.shadow.camera.far = 25;
scene.add(light);

luzLanterna = new THREE.PointLight(0xFFFFFF, 3, 18);
luzLanterna.position.set(49.6, 0.65, -54.5);
luzLanterna.castShadow = true;
luzLanterna.shadow.camera.near = 0.1;
luzLanterna.shadow.camera.far = 25;
scene.add(luzLanterna);
```

Figura 17 - Configuração das Luzes (Parte 1)

```
const farolLuz1 = new THREE.SpotLight(0xFFFFFF, 2, 18, Math.PI/5);
farolLuz1.castShadow = true;
farolLuz1.target = targetFarol1;

const farolLuz2 = new THREE.SpotLight(0xFFFFFF, 2, 18, Math.PI/5);
farolLuz2.castShadow = true;
farolLuz2.target = targetFarol2;
```

Figura 18 - Configuração das Luzes (Parte 2)

Interações com a Cena

No cenário, tal como é requisitado no protocolo, o utilizador é capaz de interagir com a cena através de dispositivos de *input*, como o rato e o teclado. Para possibilitar a interação, utilizamos a biblioteca *PointerLockControls*, relacionada com os movimentos do rato e facilitando também os controlos de movimentação do personagem com as teclas 'WASD' e com as teclas 'Shift + W' para que este se desloque a uma velocidade superior, como podemos observar na Figura 19.

```
controls = new THREE.PointerLockControls(camera, renderer.domElement);
//Movement
if(keyboard[87]){ //W
    controls.moveForward(player.speed);
}
if(keyboard[83]){ //S
    controls.moveForward(-player.speed);
}
if(keyboard[65]){ //A
    controls.moveRight(-player.speed);
}
if(keyboard[68]){ //D
    controls.moveRight(player.speed);
}
if(keyboard[16] && keyboard[87]){ //Shift + W
    controls.moveForward(player.speed * 10);
}
```

Figura 19 - Movimento do Personagem

Como especificado no subcapítulo "Configuração de Câmara", no cenário é possível alternar entre as câmaras em perspetiva e ortográfica, sendo que a ortográfica simula um minimapa. Para esta alternância, utilizamos a tecla 'M', implementada da seguinte forma:

```
if(teclado[77]){ //M
    if(c == 0)
    c++;
    else
    c--;
}
```

Figura 20 - Tecla 'M' para Minimapa

Como especificado no subcapítulo "Configurações de Luzes", no cenário é possível desligar/ligar cada uma das luzes individualmente. Para isto, foram utilizadas as teclas 'OLP', sendo que a tecla 'O' permite desligar/ligar a *DirectionalLight*, a tecla 'P' permite desligar/ligar a *PointLight* e a tecla 'L' permite desligar/ligar a *AmbientLight*. A implementação desta abordagem pode ser observada na Figura 21.

```
if(teclado[80]){ //P (Point Lights)
    if(light.visible == true){
        light.visible = false;
        luzLanterna.visible = false;
    }
    else{
        light.visible = true;
        luzLanterna.visible = true;
    }
}
if(teclado[76]){ //L (Ambient Light)
    if(ambientLight.visible == true)
    ambientLight.visible = false;
    else
    ambientLight.visible = true;
}
if(teclado[79]){ //O (Directional Light)
    if(directionalLight.visible == true)
    directionalLight.visible = false;
    else
    directionalLight.visible = true;
}
```

Figura 21 - Teclas 'OLP' para as Luzes

```
if(keyboard[37]){ //Left arrow key
    camera.rotation.y -= player.turnSpeed;
}
if(keyboard[39]){ //Right arrow key
    camera.rotation.y += player.turnSpeed;
}
```

Figura 22 - Teclas das Setas para a Rotação do Personagem

Caso os *PointerLockControls* não funcionem corretamente, utilizamos a alternativa apresentada na Figura 22, que consiste no uso das teclas das setas, para mudar a rotação da câmara conforme a tecla selecionada.

Como um extra criado para a testagem da aplicação durante o seu desenvolvimento, implementamos duas teclas que possibilitam o voo do personagem (tecla 'C' para descer e 'espaço' para subir). Esta implementação pode ser observada na Figura 23.

O nosso cenário requere que o personagem elimine os zombies através do uso de uma arma. Para este efeito, ou seja, em relação ao disparo da arma, implementamos a tecla 'F', como podemos observar na Figura 24.

```
if(keyboard[32]){ //Spacebar
    camera.position.y += 0.3;
}
if(keyboard[67]){ //C
    camera.position.y -= 0.3;
}
```

Figura 23 - Teclas para Voo

```
if(teclado[70]){ //F
    if(player.canShoot <= 0){
        isClicked = 1;
    }
}
```

Figura 24 - Tecla para Disparo

Como alternativa à tecla 'F', implementamos também os botões do rato (ambos) como forma de disparar através deste. O uso dos botões do rato permite também o uso da API dos *PointerLockControls*.

```
var isClicked = 0;
document.addEventListener('mousedown', ev =>{
    if(player.canShoot <= 0){
        isClicked = 1;
    }

    controls.lock();
});
```

Figura 25 - Botões do Rato

Animações

A animação é um método em computação gráfica que permite que as figuras sejam manipuladas para que aparentem possuir vida própria. Esta manipulação pode ser atingida ao utilizar as transformações geométricas de translação, rotação e escala. A translação é uma soma de vetores enquanto a rotação e a escala são produtos de matrizes. A translação é o deslocamento dos objetos segundo um determinado vetor e tem como propriedades preservar comprimentos e ângulos. Para deslocar polígonos a transformação é aplicada a cada um dos vértices. A escala consiste na alteração do tamanho de um objeto, através da multiplicação das coordenadas x e y por constantes e não preserva comprimentos nem ângulos, excetuando se a escala for uniforme. A rotação é o deslocamento circular de um objeto sobre a origem e tem como propriedades preservar comprimentos e ângulos.

Em termos de animação, no nosso cenário, constam três animações: a rotação de uma árvore no ecrã de carregamento; a entrada cinematográfica na cena; e o caminho tomado pelos zombies na vila.

A primeira animação presente no nosso cenário que vamos apresentar é a animação de uma árvore no ecrã de carregamento. Esta animação consiste numa transformação geométrica de rotação no eixo do y.

```
if(RESOURCES_LOADED == false){
    requestAnimationFrame(Update);

    loadingScreen.box.rotation.y += 0.05;
    loadingScreen.box.scale.set(0.8, 0.8, 0.8);
    renderer.render(loadingScreen.scene, loadingScreen.camera);
    return;
}
```

Figura 26 - Definição da Animação da Árvore no Ecrã de Carregamento

A segunda animação presente no nosso cenário que vamos apresentar é a animação da câmara cinematográfica introdutória, que permite que o mapa seja mostrado. Esta câmara encontra-se a uma posição mais elevada do que a posição da câmara do personagem e vai descendo efetuando uma translação negativa no eixo do y, efetuando também uma ligeira rotação no eixo do x e uma rotação mais acentuada no eixo do y, para que possa ser mostrado o mapa na sua totalidade. Esta câmara é outra vez alternada para a do personagem assim que acabar a sua animação.

```
if(cameraCinematica.position.distanceTo(meshFloor.position) > 2.5){
    cameraCinematica.position.y -= 1;
    cameraCinematica.rotation.x -= Math.PI / 1000
    cameraCinematica.rotation.y -= Math.PI / 25
    renderer.render(scene, cameraCinematica);
```

Figura 27 - Definição da Animação da Câmara Cinematográfica

A terceira animação presente no nosso cenário que vamos apresentar é a animação dos zombies que vagueiam pela Vila. Para esse efeito vamos começar por definir um *FBXImporter* que servirá para importar os objetos 3D com as animações incorporadas. Para controlar estas animações será também definido um novo *AnimationMixer* que servirá para correr as animações (Figuras 28 e 29).

```
var importer = new THREE.FBXLoader();
var mixerAnimacao, objetoImportado;
importer.load('./3D Objects/Walking.fbx', function(object){
    mixerAnimacao = new THREE.AnimationMixer(object);
    var action = mixerAnimacao.clipAction(object.animations[0]);
    action.play();
    object.traverse(function(child){
        if(child.isMesh){
            child.castShadow = true;
            child.receiveShadow = true;
        }
    });
    scene.add(object);
    object.scale.x = 0.01;
    object.scale.y = 0.01;
    object.scale.z = 0.01;
    object.position.set(0, 0.1, 6);

    //Guardar o objeto importado na variável objetoImportado
    objetoImportado = object;
});
```

Figura 28 - Definição da Animação dos Zombies (Parte 1)

```
var delta = clock.getDelta();
if(mixerAnimacao){
    mixerAnimacao.update(delta);
}
```

Figura 29 - Definição da Animação dos Zombies (Parte 2)

```
if(zombieAndarFrente == true){
    if(objetoImportado.position.z > 59){
        zombieAndarFrente = false;
        objetoImportado.rotation.set(0, Math.PI, 0);
    }else{
        objetoImportado.position.z += velocidadeZombies;
    }
}
if(zombieAndarFrente == false){
    if(objetoImportado.position.z < 5){
        objetoImportado.rotation.set(0, 2 * Math.PI, 0);
        zombieAndarFrente = true;
    }else{
        objetoImportado.position.z -= velocidadeZombies;
    }
}
```

Figura 30 - Definição da Animação dos Zombies (Parte 3)

Ainda sobre a terceira animação, para definir a rota dos zombies foram aplicadas transformações geométricas de translação aos objetos importados, caso estes atinjam o limite estabelecido é-lhes aplicada uma rotação de 180° no eixo do y para que estes façam a rota no sentido oposto à que vieram (Figura 30).

Conclusão

Com o desenvolvimento deste trabalho, pusemos em prática todos os fundamentos apresentados nas aulas teóricas, cimentando melhor os nossos conhecimentos relacionados com a computação gráfica, com o auxílio da biblioteca *three.js*. Este trabalho fomentou a nossa criatividade e originalidade, pois temos como objetivo tornar a vila, ou seja, todo o cenário, o mais atrativa e realista possível.

Ao longo do desenvolvimento deste trabalho surgiram algumas adversidades, sendo a nossa principal dificuldade a utilização da biblioteca *PointerLockControls*. Para além disso, tivemos ainda uma outra pequena dificuldade no uso da biblioteca *Reflector*. Ao contrário da primeira não arranjamos uma alternativa levando ao abandono da ideia.

Melhorias

Tendo em conta a fraqueza das nossas animações no momento da apresentação, decidimos aceitar a sugestão do professor e implementar uma câmara cinematográfica, de maneira a aumentar a complexidade do fator “Animação” e melhorar ainda mais a qualidade do projeto no geral.

Autoavaliação

Para a autoavaliação utilizamos como partido a grelha de autoavaliação fornecida pelo professor no protocolo do trabalho. Tendo em conta esta grelha, autoavaliamo-nos na escala do excelente [18-20], visto que o nosso trabalho cumpre todos os requisitos, o relatório está excelente e o projeto possui complexidade e originalidade elevada. Para além disso, fizemos melhorias após a apresentação que reforçam a qualidade do projeto.

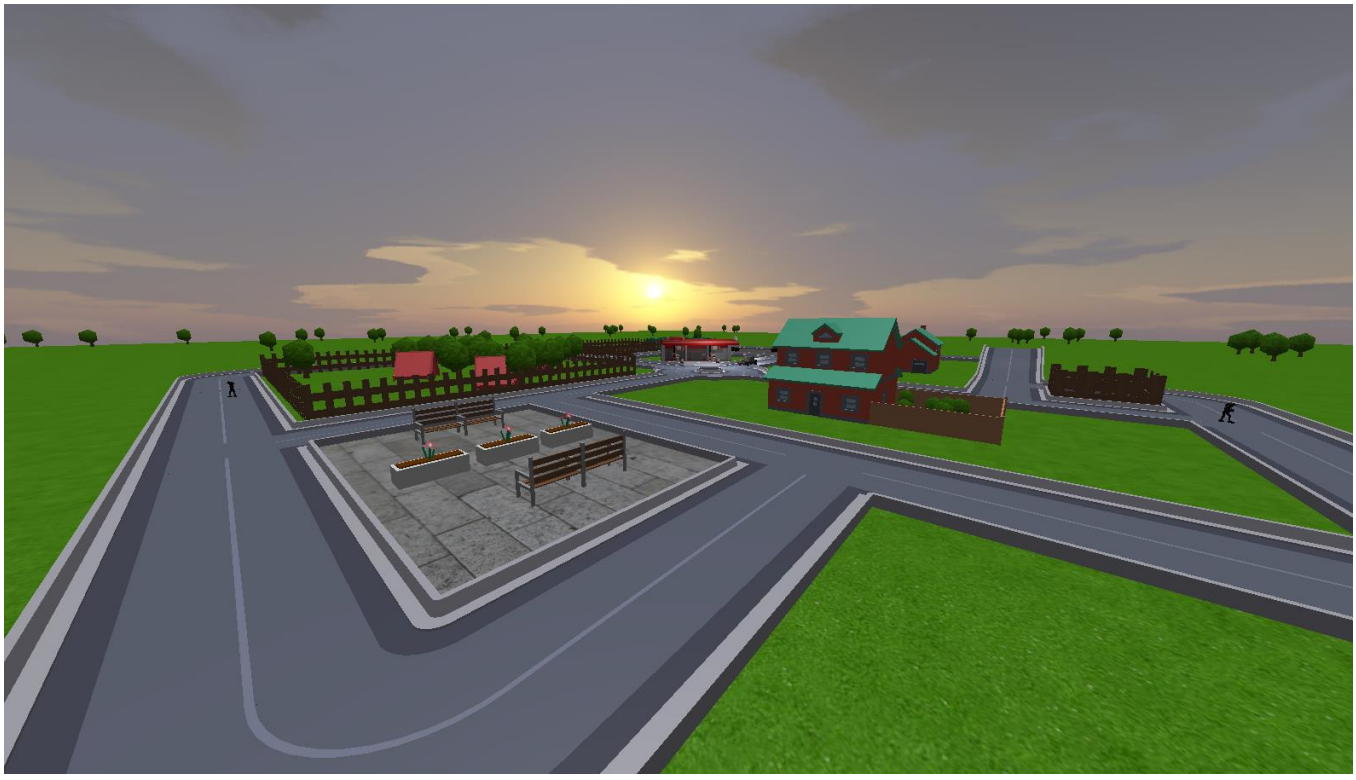

Figura 31 - Produto Final

Bibliografia

https://threejs.org/
https://kenney.nl/
https://www.mixamo.com/#/
https://www.youtube.com/watch?v=X2Y7fcQHRjw&list=PLCTVwBLCNozSGfxhCIiEH26tbJrQ2_Bw3
https://sketchfab.com/feed
https://free3d.com/
https://www.cgtrader.com/
Bessa M., Melo M. (2021), Acetatos das Aulas Teóricas, UTAD, Vila Real.
Bessa M., Melo M. (2021). Protocolo do Trabalho Prático. UTAD, Vila Real.
Bessa M., Melo M. (2021). Tutoriais das Aulas Práticas. UTAD, Vila Real.
Interactive Computer Graphics - A Top Down Approach With Shader-Based OpenGL, Angel E., Shreiner D., 6th Edition, 2012 – ISBN: