Relatório Final do Trabalho Prático

da Unidade Curricular de

Computação Gráfica

do curso

Licenciatura em Engenharia Informática

Nome dos docentes
Maximino Esteves Correia Bessa
Miguel Ângelo Correia de Melo

Nome dos discentes	Número do aluno
André Filipe Correia Pereira	74066
Filipe José Serra Pantoja Gomes da Silva	63555

Título do trabalho
Shooting Humanoids 3D

Ano Letivo:
2022/2023

Grupo:
Grupo 32

Índice

1. Introdução

1.1 Objetivos do Trabalho

No âmbito da Unidade Curricular de Computação gráfico foi pedido aos alunos que criassem uma aplicação gráfica implementada recorrendo à Biblioteca WebGL *"three.js"*. Como requisitos, o cenário deverá contemplar os seguintes elementos:
- Construção de objetos 3D complexos com recurso às primitivas básicas de three.js. Os objetos 3D criados devem ser texturizados com recurso tanto a materiais nativos como texturas importadas (o ambiente pode ser complementado com objetos 3D importados).
- Configuração de Câmara: deverá ser possível alternar entre uma configuração de câmara em perspetiva e de câmara ortográfica.
- Configuração de luzes: deverão existir diversos tipos de luzes (*PointLight, DirectionalLight e/ou AmbientLight*), sendo possível desligar cada tipo de luz Individualmente.
- Interação com a cena: o utilizador deve ser capaz de interagir com a cena através do rato e/ou teclado.
-Animação: os objetos devem ser animados de forma a demonstrar um conceito a ilustrar. Deverá ser possível repetir/rever a animação.

1.2 Cenário Simplificado

No desenvolvimento desta aplicação, serão utilizadas as funcionalidades da biblioteca *three.js* para uma demonstração da aplicação. O cenário envolverá as 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 por nós. Em relação à configuração das luzes, o cenário será iluminado por uma *DirectionalLight*, uma *AmbientLight*, bem como por seis luzes *PointLight* e será permitido desligar/ligar cada tipo individualmente. Para movimentar o personagem, são utilizadas as teclas *'WASD'* bem como o movimento do rato 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 humanoides a vaguearem e a árvore do ecrã de carregamento.
Assim, temos o desenvolvimento de uma aplicação cujo cenário em que decorre a ação é um parque com umas figuras humanoides presentes, onde o personagem têm como objetivo eliminar os humanoides espalhados pelo parque utilizando a sua arma.
Aqui presente 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.

2. Desenvolvimento

Neste capítulo vamos explicar e demonstrar os métodos tomados no desenvolvimento de cada um dos requisitos e a especificação de tudo o que foi feito dentro de cada um.
Como dito anteriormente, o cenário retrata um parque e este contém vários objetos 3D. Foram importados e elaborados por nós 3 objetos 3D (bancos, vasos e árvores), seis luzes *PointLight* (uma em cada spawnpoint dos humanoides), uma *AmbientLight* e uma *DirectionalLight*. Utilizamos a biblioteca *PointerLockControls*, tecla 'M' para abrir minimapa, teclas 'WASD' para movimentar o personagem, rato e tecla 'F' para disparar a arma, teclas 'OPL' para desligar cada luz individualmente e a tecla 'SHIFT' para aumentar a velocidade de movimentação do personagem. Para as animações foram importados humanoides a vaguearem pelo parque e a árvore do ecrã de carregamento.

2.1 Construção dos 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 ao parque, também foram criados objetos 3D, sendo estes: banco, vasos 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 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 Banco:

```
function Banco() {
    const banco = new THREE.Group();
    var loadTexturaBanco = new THREE.TextureLoader(loadingManager);
    var texturaBanco = loadTexturaBanco.load("Textures/Trees/base.jpg");

    //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 1 - Criação do Objeto Banco (pt.1)

```
//Base
const base = new THREE.Group();
const tabua1 = new THREE.Mesh(
    new THREE.BoxGeometry(0.1, 0.1, 1),
    new THREE.MeshPhongMaterial({ map: texturaBanco })
);
const tabua2 = new THREE.Mesh(
    new THREE.BoxGeometry(0.1, 0.1, 1),
    new THREE.MeshPhongMaterial({ map: texturaBanco })
);
const tabua3 = new THREE.Mesh(
    new THREE.BoxGeometry(0.1, 0.1, 1),
    new THREE.MeshPhongMaterial({ map: texturaBanco })
);
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 2 -Criação do Objeto Banco (pt.2)

Como podemos observar na Figura 1, 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.
Em relação à Figura 2, 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 3, adicionamos os subgrupos ao grupo principal formando assim o banco, obtendo o resultado final apresentado na Figura 4.

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

Figura 3 – Criação do Objeto Banco (pt. 3)

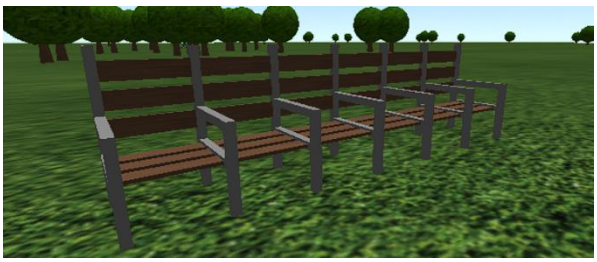

Figura 4 – Resultado do objeto 3D Banco.

Vamos agora analisar o processo de criação do objeto Arvore:

```
/*--------------ARVORES--------------*/
function Arvore() { // Função para criar uma árvore

    const arvore = new THREE.Group();  // Cria um grupo vazio para representar a árvore

    const folhasLoader = new THREE.TextureLoader(loadingManager); // Criando um loading manager de textura para as folhas da árvore
    var folhasTexture = folhasLoader.load("Textures/Cercas/leaves.jpg"); // Carrega a textura das folhas
    const folhas = new THREE.Mesh(
        new THREE.SphereGeometry(1, 10, 10),// Criando uma geometria esférica para representar as folhas (10 é o nmr de segmentos, quantos
        new THREE.MeshLambertMaterial({ map: folhasTexture })// Aplica o material de Lambert com a textura das folhas

    );

    folhas.position.y = 1.4; // Define a posição vertical das folhas em relação à árvore

    const cercaTexturaLoader = new THREE.TextureLoader(loadingManager);// Cria um loadingmanager de textura para o tronco da árvore
    var texturaCerca = cercaTexturaLoader.load("Textures/Cercas/cerca.jpg"); //Carrega a textura do tronco
    const tronco = new THREE.Mesh(
        new THREE.CylinderGeometry(0.1, 0.2, 1), // Cria uma geometria cilíndrica para representar o tronco
        new THREE.MeshLambertMaterial({ map: texturaCerca })// Aplica o material de Lambert com a textura do tronco
    );

    folhas.castShadow = true;// Habilita a projeção de sombras para as folhas
    folhas.receiveShadow = false; // Desabilita a receção de sombras para as folhas
    tronco.castShadow = true;// Habilita a projeção de sombras para o tronco
    tronco.receiveShadow = true;// Desabilita a receção de sombras para o tronco

    arvore.add(folhas); //Adiciona as folhas à arvore
    arvore.add(tronco); //Adiciona as folhas ao tronco

    return arvore;
}
/*--------------ARVORES--------------*/
```

Figura 5 - Criação do Objeto Árvore.

Figura 6 - Resultado do objeto 3D Árvore.

2.2 Configuração da câmara

Em Computação Gráfica, 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. 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 e uma para o ecrã de vitória.

```
// Camera
camera = new THREE.PerspectiveCamera(90, 1280 / 720, 0.01, 1000); // Criação da câmera perspectiva
camaraOrtografica = new THREE.OrthographicCamera(-100, 100, 100, -100, 0.1, 1000); // Criação da câmera ortográfica
camaraOrtografica.position.set(0, 10, 0); // Definindo a posição da câmera ortográfica
camaraOrtografica.rotation.set(-Math.PI / 2, 0, 0); // Definindo a rotação da câmera ortográfica
```

Figura 7 - Configuração das Câmaras do Personagem e do Minimapa.

A Figura 7 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 8) é efetuada, é explicada detalhadamente no subcapítulo "Interação com a Cena". Na Figura 10 são apresentadas as configurações das câmaras em perspetiva dos ecrãs de carregamento e de final de jogo.

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

Figura 8 - Alternância das câmaras

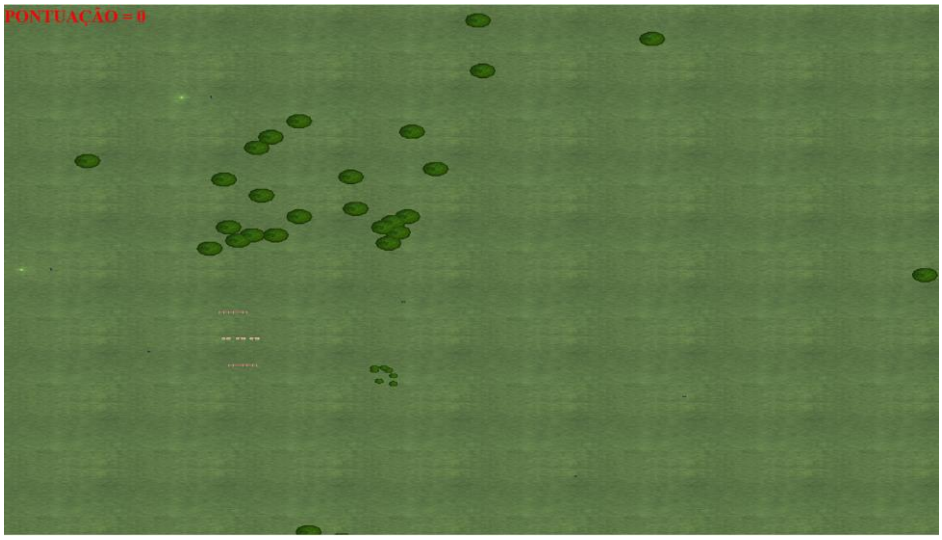

Figura 9 – Minimapa

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

Figura 10 - Configuração das Câmaras em Perspetiva de Ecrã de Carregamento e de Final de Jogo

2.3 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 (seis *PointLight*, uma *DirectionalLight* e uma *AmbientLight*), sendo possível desligar cada tipo de luz individualmente. Relativamente às *PointLight*, estas estão presentes nos spawnpoints dos humanoides, como podemos observar na Figura 11. 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.

```
// Luzes
ambientLight = new THREE.AmbientLight(0xffffff, 0.3);// Cria uma luz ambiente com cor branca e intensidade 0.3
scene.add(ambientLight); // Adiciona a luz ambiente à cena

// Cria uma luz direcional com cor branca e intensidade 0.5
directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
directionalLight.castShadow = true; // Permite que a luz direcional projete sombras
scene.add(directionalLight); // Adiciona a luz direcional à cena
// // Criação das PointLights para os humanoide

// Cria uma luz pontual com cor branca, intensidade 2 e posição específica
light = new THREE.PointLight(0xffffff, 2, 18);
// light.position.set(6, 5, -3); // Define a posição da luz pontual
light.position.set(-96, 0.1, 0);
light.castShadow = true; // Permite que a luz pontual projete sombras
light.shadow.camera.near = 0.1; // Configura o plano de corte próximo da sombra
light.shadow.camera.far = 25; // Configura o plano de corte distante da sombra
scene.add(light); // Adiciona a luz pontual à cena

// Cria outra luz pontual com cor branca, intensidade 3 e posição específica
luzPoint = new THREE.PointLight(0xFFFFFF, 2, 18);
luzPoint.position.set(-62, 0.1, -65); // Define a posição da segunda luz pontual
luzPoint.castShadow = true; // Permite que a segunda luz pontual projete sombras
luzPoint.shadow.camera.near = 0.1; // Configuran o plano de corte próximo da sombra
luzPoint.shadow.camera.far = 25; // Configura o plano de corte distante da sombra
scene.add(luzPoint); // Adiciona a segunda luz pontual à cena
```

Figura 11 - Configuração das Luzes

2.4 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 12.

```
controls = new THREE.PointerLockControls(camera, renderer.domElement);
//Movimento
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 (Correr)
    controls.moveForward(player.speed * 2); // o 2 define a velocidade x2
}
```

Figura 12 - Movimento da 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 13 - 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 14.

```
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 14 - Teclas 'OLP' para as Luzes

O nosso cenário requere que o personagem elimine os humanoides 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 15. 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* (Figura 16).

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

Figura 15 - Alternativa a Disparo

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

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

Figura 16 - Botões do Rato

2.5 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 duas animações: a rotação de uma árvore no ecrã de carregamento e o caminho tomado pelos humanoides no parque. 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 17 - Definição da Animação da Árvore no Ecrã de Carregamento

A outra animação presente no nosso cenário que vamos apresentar é a animação dos humanoides que vagueiam pelo parque. 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 20 e 19)

```
//Movimentação humanoids
//humanoid 1
if (humanoidAndarFrente == true) {// Verifica se o humanoid está a andar para a frente
    if (objetoImportado.position.z > 59) { // Verifica se a posição z do objetoImportado é maior que 59
        humanoidAndarFrente = false;// Caso seja maior, indica que o humanoid deve começar a andar para trás
        objetoImportado.rotation.set(0, Math.PI, 0); // Define a rotação do objetoImportado para que ele fique de costas
    } else {
        // Caso contrário, incrementa a posição z do objetoImportado pela velocidade dos humanoids
        objetoImportado.position.z += velocidadehumanoids;
    }
}
if (humanoidAndarFrente == false) {
    if (objetoImportado.position.z < 5) {
        objetoImportado.rotation.set(0, 2 * Math.PI, 0);
        humanoidAndarFrente = true;
    } else {
        objetoImportado.position.z -= velocidadehumanoids;
    }
}
```

Figura 18 - Animação Humanoide

```
//humanoids
// Cria uma instância do FBXLoader para importar objetos 3D no formato FBX
var importer = new THREE.FBXLoader();

// Variáveis para controlar a animação e os objetos importados
var mixerAnimacao, objetoImportado;

// Carrega o arquivo Walking.fbx e executa uma função de callback quando o carregamento estiver completo
importer.load('./3D Objects/Walking.fbx', function (object) {
    // Cria uma instância do AnimationMixer e associa-a ao objeto importado
    mixerAnimacao = new THREE.AnimationMixer(object);
    // Obtém a primeira animação do objeto importado e cria uma ação de animação
    var action = mixerAnimacao.clipAction(object.animations[0]);
    // Inician a reprodução da animação
    action.play();
    // Percorre todos os elementos filho do objeto importado
    object.traverse(function (child) {
        if (child.isMesh) {
            // Habilita o objeto filho para receber e projetar sombras
            child.castShadow = true;
            child.receiveShadow = true;
        }
    });
    //Adiciona o objeto à cena
    scene.add(object);

    //Configuração da escala e posição do objeto importado
    object.scale.x = 0.0125;
    object.scale.y = 0.0125;
    object.scale.z = 0.0125;
    object.position.set(-15, 0.1, 6);

    //Guarda o objeto importado na variável objetoImportado
    objetoImportado = object;
```

Figura 10 - Animação Humanoide

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

Figura 19 - Animação Humanoide (pt. 2)

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 18)

3. Conclusão

Este projeto foi uma experiência desafiadora que também estimulou nossa criatividade e originalidade, uma vez que nosso objetivo principal era criar um cenário imersivo.

Ao longo do desenvolvimento, encontramos algumas adversidades, sendo a principal delas a dificuldade na utilização da biblioteca '*PointerLockControls'*. Essa ferramenta específica apresentou desafios significativos, exigindo esforço extra para superar as dificuldades técnicas associadas. Além disso, também enfrentamos uma pequena dificuldade ao tentar alterar a trajetória dos humanoides para um percurso mais complexo. Infelizmente, não conseguimos encontrar uma alternativa satisfatória para contornar esse obstáculo, o que nos levou a abandonar a ideia e ficar pela trajetória inicial.

Apesar dos desafios encontrados, consideramos essa experiência extremamente valiosa, pois permitiu que melhorássemos as nossas habilidades práticas, desenvolvendo soluções criativas e eficientes para os problemas que surgiram ao longo do caminho.

A nossa auto avaliação tendo em conta a grelha do protocolo é uma nota entre o [15-17], visto que o nosso trabalho cumpre todos os requisitos e o relatório está bem conseguido.

4. Bibliografia

- https://threejs.org/
- https://kenney.nl/
- https://www.mixamo.com/#/
- https://www.youtube.com/watch?v=X2Y7fcQHRjw&list=PLCTVwBLCNozSGfxhCIiEH26tbJrQ2_Bw3&index=1&ab_channel=xSaucecode
- https://www.youtube.com/watch?v=EkPfhzIbp2g&t=46s
- https://www.youtube.com/watch?v=SBfZAVzbhCg
- https://sketchfab.com/feed
- https://free3d.com/
- https://www.cgtrader.com/
- Documentos disponibilizados no SIDE e MOODLE (protocolos, tutoriais, pdf´s das aulas)
- Interactive Computer Graphics - A Top Down Approach With Shader-Based OpenGL, Angel E., Shreiner D., 6th Edition, 2012.