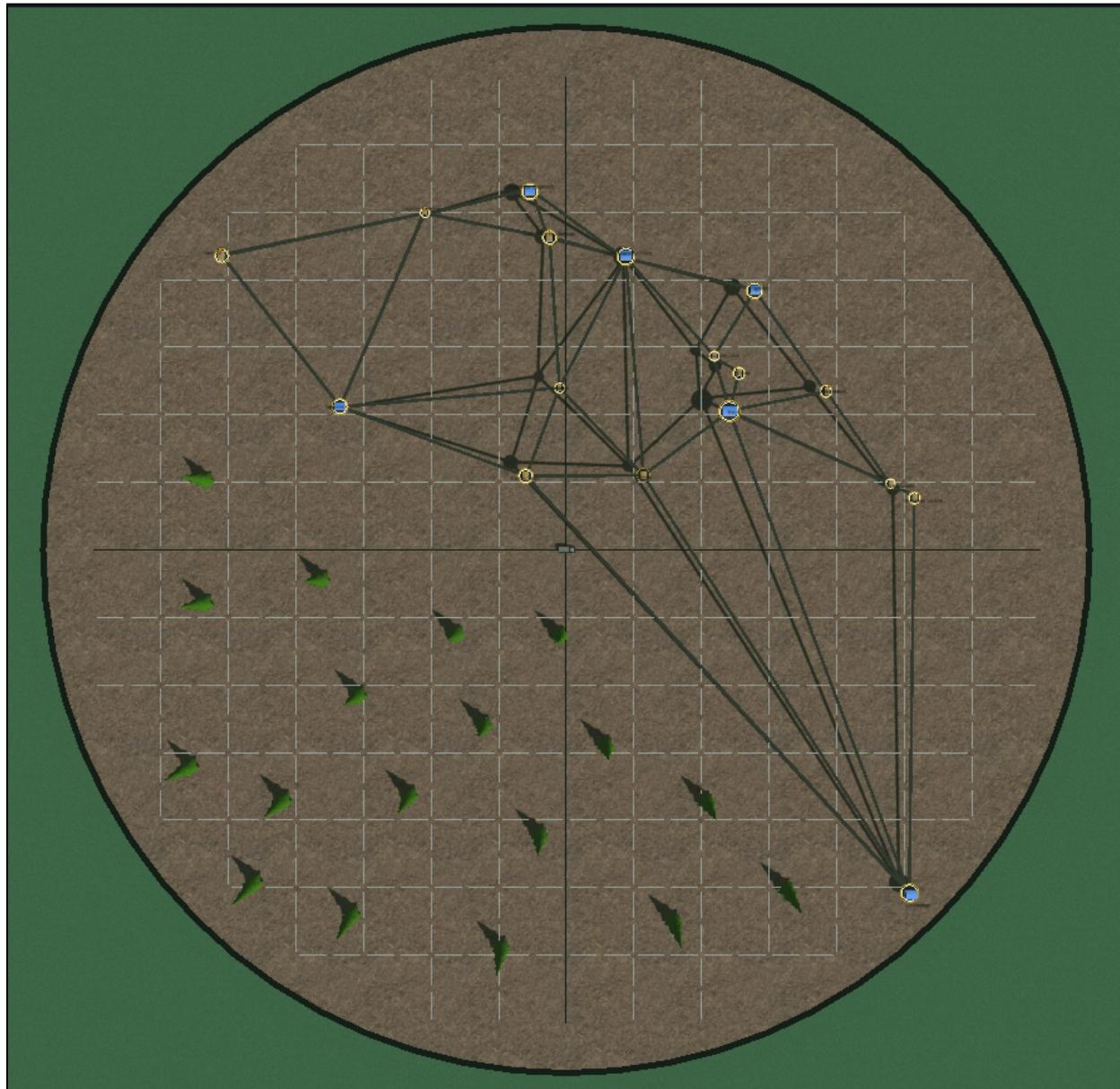


US 1

Como gestor de logística pretendo visualizar graficamente em 3D a rede viária. A visualização deverá ocupar a totalidade da área útil do browser. Modelar a rede viária (ver Tutorial).



Primeiramente foram criados dois cilindros de grande largura e pequena altura para serem a base da visualização. Um deles com uma textura do chão, e outro, um pouco maior que o anterior e apenas com a cor preta (para dar a ilusão de uma borda).

```
///////////
// Create the graph plane //
///////////

mesh = new THREE.Mesh(
  new THREE.CylinderGeometry( radiusTop: 77, radiusBottom: 77, height: 0.1, radialSegments: 64),
  new THREE.MeshStandardMaterial( parameters: {color: 0xffffffff, map: texture})
);

this.scene.add(mesh);

///////////
// Create a slightly larger plane to serve as a border //
///////////

mesh = new THREE.Mesh(
  new THREE.CylinderGeometry( radiusTop: 78, radiusBottom: 78, height: 0.1, radialSegments: 64),
  new THREE.MeshStandardMaterial( parameters: {color: 0x000000})
);

this.scene.add(mesh);
```

De seguida são adicionadas as linhas sólidas que representam o eixo do X e do Z.

```
///////////
// Draw solid axis lines //
/////////

this.drawSolidLines( points: [
  new THREE.Vector3( x: 0, y: 0, z: 70),
  new THREE.Vector3( x: 0, y: 0, z: -70),
  new THREE.Vector3( x: 70, y: 0, z: 0),
  new THREE.Vector3( x: -70, y: 0, z: 0)
], new THREE.LineBasicMaterial( parameters: {color: 0x000000}));
```

```
// Draws solid lines
drawSolidLines(points: any[], material: THREE.LineBasicMaterial) {

  const geometry = new THREE.BufferGeometry().setFromPoints(points);
  this.scene.add(new THREE.LineSegments(geometry, material));
}
```

Depois umas linhas guia a tracejado:

```
///////////
// Draw dashed helper lines //
///////////

this.drawDottedLines( points: [
  [new THREE.Vector3( x: -60, y: 0, z: 39.5), new THREE.Vector3( x: -60, y: 0, z: -40)],
  [new THREE.Vector3( x: -50, y: 0, z: 49.5), new THREE.Vector3( x: -50, y: 0, z: -50)],
  [new THREE.Vector3( x: -40, y: 0, z: 59.5), new THREE.Vector3( x: -40, y: 0, z: -60)],
  [new THREE.Vector3( x: -30, y: 0, z: 69.5), new THREE.Vector3( x: -30, y: 0, z: -70)],
  [new THREE.Vector3( x: -20, y: 0, z: 69.5), new THREE.Vector3( x: -20, y: 0, z: -70)],
  [new THREE.Vector3( x: -10, y: 0, z: 69.5), new THREE.Vector3( x: -10, y: 0, z: -70)],
  [new THREE.Vector3( x: 60, y: 0, z: 39.5), new THREE.Vector3( x: 60, y: 0, z: -40)],
  [new THREE.Vector3( x: 50, y: 0, z: 49.5), new THREE.Vector3( x: 50, y: 0, z: -50)],
  [new THREE.Vector3( x: 40, y: 0, z: 59.5), new THREE.Vector3( x: 40, y: 0, z: -60)],
  [new THREE.Vector3( x: 30, y: 0, z: 59.5), new THREE.Vector3( x: 30, y: 0, z: -60)],
  [new THREE.Vector3( x: 20, y: 0, z: 69.5), new THREE.Vector3( x: 20, y: 0, z: -70)],
  [new THREE.Vector3( x: 10, y: 0, z: 69.5), new THREE.Vector3( x: 10, y: 0, z: -70)],
  [new THREE.Vector3( x: 39.5, y: 0, z: -60), new THREE.Vector3( x: -40, y: 0, z: -60)],
  [new THREE.Vector3( x: 49.5, y: 0, z: -50), new THREE.Vector3( x: -50, y: 0, z: -50)],
  [new THREE.Vector3( x: 59.5, y: 0, z: -40), new THREE.Vector3( x: -60, y: 0, z: -40)],
  [new THREE.Vector3( x: 59.5, y: 0, z: -30), new THREE.Vector3( x: -60, y: 0, z: -30)],
  [new THREE.Vector3( x: 69.5, y: 0, z: -20), new THREE.Vector3( x: -70, y: 0, z: -20)],
  [new THREE.Vector3( x: 69.5, y: 0, z: -10), new THREE.Vector3( x: -70, y: 0, z: -10)],
  [new THREE.Vector3( x: 39.5, y: 0, z: 60), new THREE.Vector3( x: -60, y: 0, z: 60)],
  [new THREE.Vector3( x: 49.5, y: 0, z: 50), new THREE.Vector3( x: -50, y: 0, z: 50)],
  [new THREE.Vector3( x: 59.5, y: 0, z: 40), new THREE.Vector3( x: -60, y: 0, z: 40)],
  [new THREE.Vector3( x: 59.5, y: 0, z: 30), new THREE.Vector3( x: -60, y: 0, z: 30)],
  [new THREE.Vector3( x: 69.5, y: 0, z: 20), new THREE.Vector3( x: -70, y: 0, z: 20)],
  [new THREE.Vector3( x: 69.5, y: 0, z: 10), new THREE.Vector3( x: -70, y: 0, z: 10)]
], new THREE.LineDashedMaterial( parameters: {
  color: 0xa0a0a0,
  dashSize: 4,
  gapSize: 1
}));
```

```
// Draws dotted lines
drawDottedLines(points: any[], material: THREE.LineDashedMaterial) {

  let geometry, line;

  for (let i = 0; i < points.length; i++) {
    geometry = new THREE.BufferGeometry().setFromPoints(points[i]);
    line = new THREE.LineSegments(geometry, material);
    line.computeLineDistances();
    this.scene.add(line);
  }
}
```

Assim como os cilindros que compõem a base da vista, foi criado um par de cilindros, mais uma vez, um deles com textura de asfalto e outro ligeiramente maior de cor preta para fazer de borda. Isto servirá para representar a rotunda que pertence a cada armazém.

```

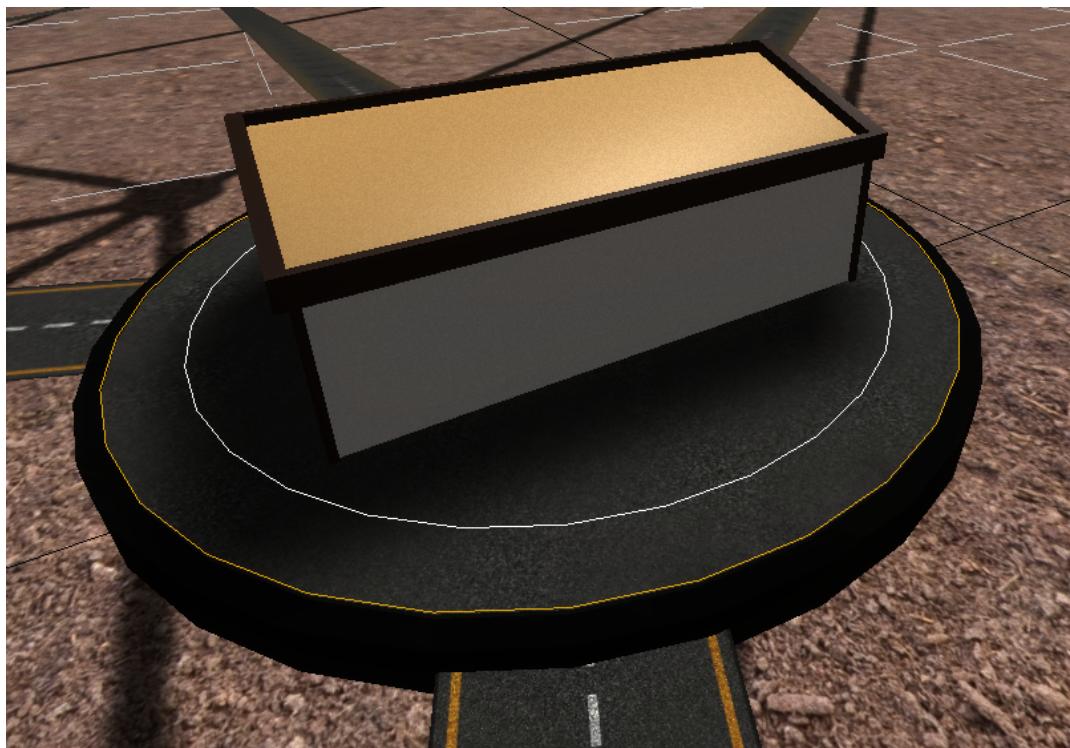
// Create the main cylinder that represents the warehouse
geometry = new THREE.CylinderGeometry(this.radius, this.radius, height: 0.1, segments);
material = new THREE.MeshStandardMaterial( parameters: {color: 0xffffffff, map: texture});
mesh = new THREE.Mesh(geometry, material);

this.object.add(mesh);

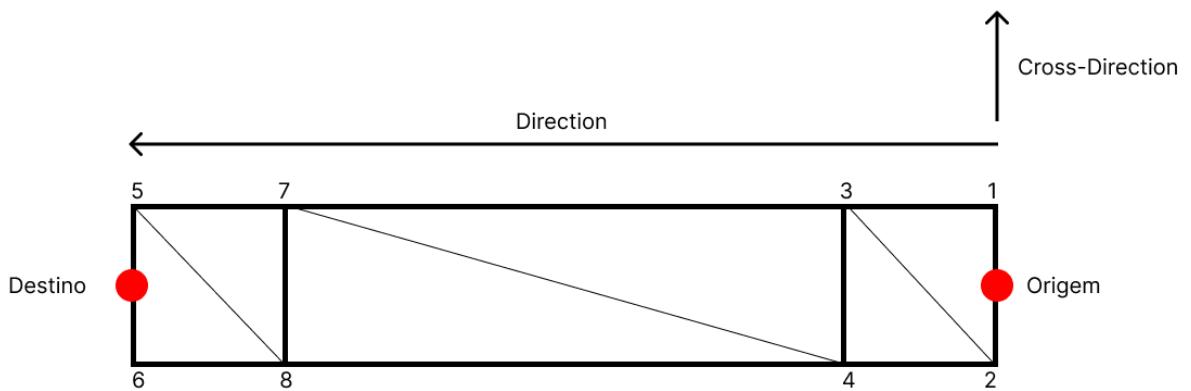
// Create a slightly larger cylinder (black) to serve as a border
geometry = new THREE.CylinderGeometry( radiusTop: this.radius * 1.05, radiusBottom: this.radius * 1.05, height: 0.1, segments);
material = new THREE.MeshStandardMaterial( parameters: {color: 0x000000});
mesh = new THREE.Mesh(geometry, material);

this.object.add(mesh);

```



Por fim temos as ligações entre armazéns que foram criados da seguinte forma:



Começando com as posições dos armazéns de origem e destino, calcula-se um vetor “direction” subtraindo as coordenadas de um pelo outro. A partir desse vetor calcula-se outro vetor “cross-direction” perpendicular ao “direction”. Aplica-se o “cross-direction” às coordenadas do armazém de origem e obtém-se o ponto nº1. Aplica-se o inverso do “cross-direction” às coordenadas do armazém de origem e obtém-se o ponto nº2. Com o vetor “direction” e os pontos 1 e 2 obtêm-se os pontos 3 e 4. Depois de fazer o mesmo para o armazém de destino temos os pontos todos necessários para criar a geometria da ligação.



US 2

Como gestor de logística pretendo visualizar graficamente os armazéns existentes. Criar ou importar os modelos 3D correspondentes (por exemplo, OBJ, GLTF, 3DS ou outros).

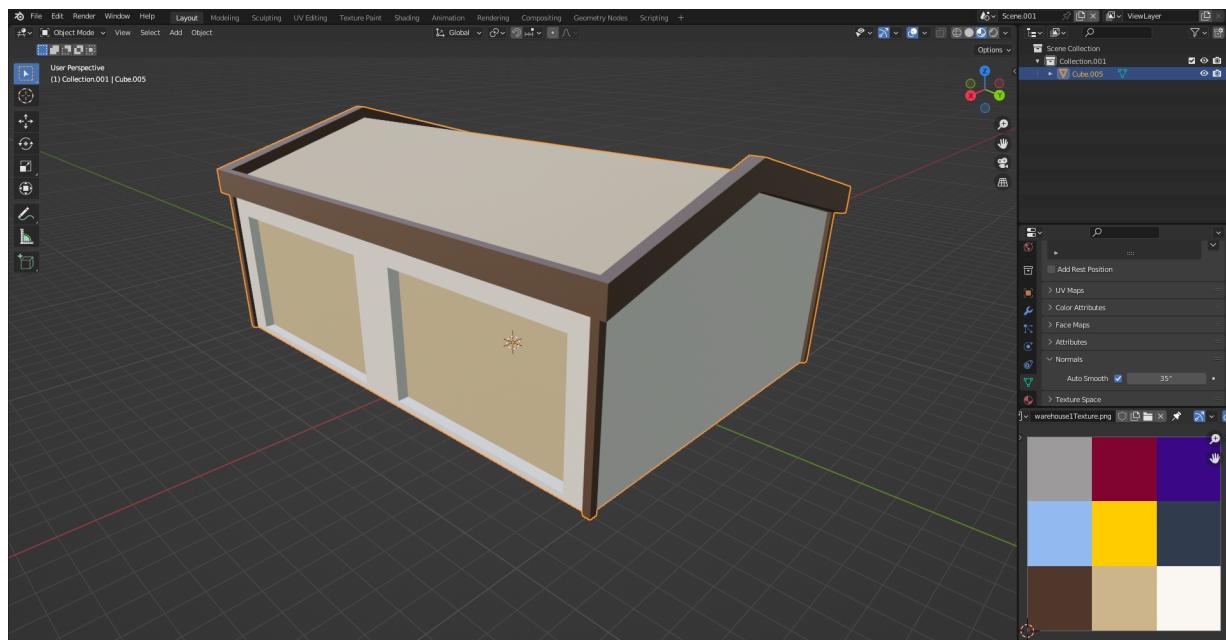
```
// Imports the warehouse model
createWarehouse(model: string) {

    new GLTFLoader().load(model, onLoad: (model) => {

        model.scene.traverse( callback: (node : Object3D ) => {
            // @ts-ignore
            if (node.isMesh) node.castShadow = true;
        });

        const mesh = model.scene;
        mesh.scale.set(this.radius / 16, this.radius / 16, this.radius / 16);
        this.object.add(mesh);
    });
}
```

Os modelos e as texturas dos armazéns foram criados com recurso ao software de modelação 3D chamado Blender.



US 3

Como gestor de logística pretendo controlar a visualização. Adicionar os comandos da câmera pan, zoom e orbit (sugestão: botão direito do rato – pan; roda do rato – zoom; botão esquerdo do rato – orbit).

```
// Camera
this.camera = new THREE.PerspectiveCamera( fov: 75, aspect: window.innerWidth / (window.innerHeight - 90), near: 0.1, far: 10000);
this.camera.position.set( x: 0, y: 90, z: 0);

// Orbit controls
this.controls = new OrbitControls(this.camera, this.renderer.domElement);
this.controls.enableDamping = true;
this.controls.dampingFactor = 0.2;

this.controls.maxPolarAngle = Math.PI;
this.controls.target.set(0, 2, 0);
this.controls.update();
```

US 4

Como gestor de logística pretendo visualizar graficamente um camião de distribuição. Criar ou importar o modelo 3D correspondente (por exemplo, OBJ, GLTF, 3DS ou outro).

```
// Import the truck model
createTruck() {

  new GLTFLoader().load( url: '/assets/models/truck1.gltf', onLoad: (model:GLTF) => {

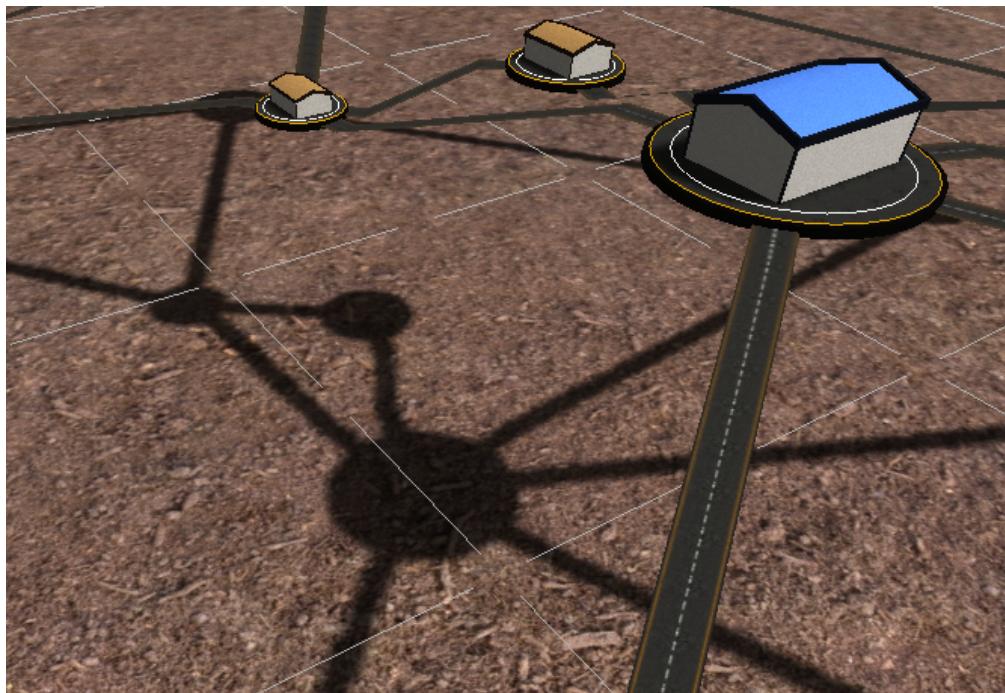
    model.scene.traverse( callback: (node: Object3D) => {
      // @ts-ignore
      if (node.isMesh) node.castShadow = true;
    });

    const mesh = model.scene;
    mesh.position.y = 1.05;
    const scaleFactor = 0.01;
    mesh.scale.set(scaleFactor, scaleFactor, scaleFactor);
    this.object.add(mesh);
  });
}
```



US 5a

Iluminar adequadamente a cena, contemplando pelo menos uma fonte de luz ambiente e uma fonte de luz direccional, com projecção de sombras.



```
///////////
// Lighting //
///////////
this.scene.add(new THREE.AmbientLight( color: 0x404040));

const sun = new THREE.DirectionalLight( color: 0xffffffff, intensity: 1);
sun.position.set( x: 25, y: 50, z: 25);

sun.castShadow = true;

sun.shadow.mapSize.width = 2048;
sun.shadow.mapSize.height = 2048;

sun.shadow.camera.top = 60;
sun.shadow.camera.bottom = -60;
sun.shadow.camera.left = -75;
sun.shadow.camera.right = 75;

this.scene.add(sun);
```

Para a iluminação foi criada uma luz ambiente com um tom ligeiramente amarelado, e uma luz direcional com um shadow map com a resolução de 2048x2048.

Nos objetos da cena ativamos opções de receber e projetar sombras.

```
// Mesh
const mesh = new THREE.Mesh(geometry, material);
mesh.receiveShadow = true;
mesh.castShadow = true;
```

US 5b

Mapear texturas apropriadas nos objetos constituintes da cena.



```
// Material
const texture = new THREE.TextureLoader().load( url: "/assets/textures/road.jpg");

texture.magFilter = THREE.LinearFilter;
texture.minFilter = THREE.LinearMipMapLinearFilter;

texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;

texture.repeat.set(1, totalLength * (1 / 0.45));

const material = new THREE.MeshStandardMaterial( parameters: {
  color: 0xffffffff,
  side: THREE.DoubleSide,
  map: texture
});
```

US 6b

Movimento interativo (controlado pelo utilizador): usar quatro teclas de navegação (sugestão: tecla ‘A’ – rodar para a esquerda; tecla ‘D’ – rodar para a direita; tecla ‘W’ – avançar; tecla ‘S’ – recuar). Implementar a detecção de colisões, de modo a impedir que o camião ultrapasse os limites da rede viária (ver Tutorial).

```
private keyCodes = {  
    forward: "ArrowUp",  
    backward: "ArrowDown",  
    right: "ArrowRight",  
    left: "ArrowLeft"  
}  
  
public keyStates = {  
    forward: false,  
    backward: false,  
    right: false,  
    left: false  
};
```

```
keyChange(event: any, state: any) {  
  
    if (document.activeElement == document.body) {  
        if (event.code == this.keyCodes.left) {  
            this.keyStates.left = state;  
        } else if (event.code == this.keyCodes.right) {  
            this.keyStates.right = state;  
        }  
        if (event.code == this.keyCodes.backward) {  
            this.keyStates.backward = state;  
        } else if (event.code == this.keyCodes.forward) {  
            this.keyStates.forward = state;  
        }  
    }  
}
```

Foi criada uma variável para armazenar o nome das teclas, e outra para guardar os seus estados

```
const deltaTime = component.clock.getDelta();
const currentDirection = THREE.MathUtils.degToRad(component.truck.direction);

if (component.keyStates.forward) {
  component.truck.acceleration = interpolate(component.truck.acceleration, {end: 5 * deltaTime, amount: deltaTime * 5});
} else if (component.keyStates.backward) {
  component.truck.acceleration = interpolate(component.truck.acceleration, {end: -2.5 * deltaTime, amount: deltaTime * 5});
} else {
  component.truck.acceleration = interpolate(component.truck.acceleration, {end: 0, amount: deltaTime * 5});
}

component.truck.position = new THREE.Vector3( x: component.truck.acceleration * Math.sin(currentDirection), y: 0, z: component.truck.acceleration * Math.cos(currentDirection));

if (component.keyStates.right && (component.truck.acceleration > 0.001 || component.truck.acceleration < -0.001)) {
  component.truck.direction -= 1000 * deltaTime * component.truck.acceleration;
} else if (component.keyStates.left && (component.truck.acceleration > 0.001 || component.truck.acceleration < -0.001)) {
  component.truck.direction += 1000 * deltaTime * component.truck.acceleration;
}

component.truck.updatePosition();
```

```
function interpolate(start: number, end: number, amount: number) {
  return (1 - amount) * start + amount * end;
}
```

A função “interpolate” serve para criar um atraso entre um momento em que o carro está parado até que está à velocidade máxima, ou seja, simula a aceleração.