INSTITUTO POLITÉCNICO DE VIANA DO CASTELO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

ENGENHARIA INFORMÁTICA

TECNOLOGIAS MULTIMÉDIA

2022/2023

TugaTraffic

Aluno:
Rafael André - 28234
Diogo Reis - 28239

Docente:
Marcelo Fernandes

Instituto Politécnico
de Viana do Castelo

10 de abril de 2023

Conteúdo

1 Objetivos **1**
1.1 Introdução 1

2 Desenvolvimento do jogo **2**
2.1 Manual de Instruções 2
2.2 Configuração de ambientes 2
2.3 mainScene 3

3 Dificuldades **10**

4 Conclusões **11**

Capítulo 1

Objetivos

1.1 Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Tecnologias Multimédia, onde o grupo decidiu criar um jogo de corridas em duas dimensões, utilizando a framework do Phaser, e desenvolvendo o jogo em HTML, JavaScript e CSS. O deploy do jogo foi feito através do netlify, ou seja, o jogo pode ser feito através de um link.

É possível acompanhar o processo de criação do jogo através do seguinte link do GitHub: link.

O GitHub é uma ferramenta que permite criar um repositório, e com a ajuda da aplicação desktop, é possível ir atualizando sempre o código, facilitando o trabalho em equipa, e evitando derivados envios de ficheiros entre o grupo.

Capítulo 2

Desenvolvimento do jogo

2.1 Manual de Instruções

No ecrã inicial aparece o logótipo do jogo, e para o iniciar é necessário pressionar o **ESPAÇO**, de seguida o jogo irá dar load aos seus assets, e depois começa o jogo, para jogar usam-se as setas do teclado, no canto superior esquerdo tem também uma lista de cheats, que aumenta se pressionar a tecla **I**, e fica escondida com a letra **O**.

Por fim, quando bater, ou ficar sem combustível, basta pressionar o botão de recomeçar o jogo, carregando com o **botão esquerdo** do rato.

2.2 Configuração de ambientes

Para iniciar o trabalho em Phaser temos de criar um ficheiro HTML (**index.html**), que representa a nossa página web.

```
<!doctype html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <title>Tuga Traffic | HTML5 Game</title>
        <script type="text/javascript"></script>
        <script src="//cdn.jsdelivr.net/npm/phaser@3.11.0/dist/phaser.js"></script>
        <script type="module" src="/src/game.js"></script>
        <script type="module" src="/src/menuScene.js"></script>
        <script type="module" src="/src/mainScene.js"></script>
        <script type="module" src="/src/loadScene.js"></script>
        <style type="text/css">
            canvas {
                display : block;
                margin : auto;
            }
        </style>
    </head>
    <body>
    </body>
</html>
```

FIGURA 2.1: Ficheiro HTML do jogo

O nosso jogo está dividido em quatro cenas. A "**mainScene**", onde está presente o código fundamental do jogo, ou seja, geração de carros, inimigos gasolina, sons, movimentos, entre outros, a "**menuScene**", que corresponde ao ecrã inicial, onde aparece o logo do jogo e uma instrução, a "**loadScene**", que é onde se dá preload à maior parte dos assets, por fim a "**gameScene**", onde corre o jogo, ou seja, a sua configuração.

A **gameScene**, que é onde o jogo corre está composta da seguinte forma:

```
import MainScene from '/src/mainScene.js';
import MenuScene from '/src/menuScene.js';
import LoadScene from '/src/loadScene.js';

var config = {
    type: Phaser.AUTO,
    width: 1080,
    height: 720,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 0 },
            debug: false
        }
    },
    scene: [MenuScene, LoadScene, MainScene]
};

window.game = new Phaser.Game(config);
```

FIGURA 2.2: Ficheiro **gameScene.js** do jogo

2.3 mainScene

Nesta secção serão explicadas as partes mais importantes do código que formam o jogo, que estão presentes na "***mainScene***".

Para criar um ambiente mais profundo, utilizando o **Tiled**, o grupo criou um sprite para a estrada (no GOD MODE, existe outra), para simular o movimento dos carros.

```
this.skysprite = this.physics.add.sprite(0,0, 'skysprite')
this.skysprite.setCollideWorldBounds(true);
this.anims.create({
    key: 'backMov',
    frames: this.anims.generateFrameNumbers('skysprite', { start: 0, end: 11 }),
    frameRate:8,
    repeat: -1
});
this.skysprite.anims.play('backMov', true);
```

FIGURA 2.3: Criação do background do jogo

Primeiramente para um jogo de carros funcionar é necessário haver movimento do carro do jogador. Inicialmente é necessário criar o carro do jogador, inserindo as coordenadas (x e y respetivamente), e de seguida ativando a sua física, ou seja, fazendo com que o carro bata em objetos.

```
// Adicionar o carro do jogador
this.player = this.physics.add.sprite(560, 690, 'carroplayer')
this.player.setCollideWorldBounds(true);
this.player.setScale(0.275)
```

FIGURA 2.4: Criação do carro do player

O código presente a seguir, representa os movimentos do carro com as setas de direção.

```
// Movimentar o carro do jogador com as teclas de controle
if (this.cursors.up.isDown) {
    this.player.setVelocityY(-300);
    this.playSound(this.acelararSom);
} else if (this.cursors.down.isDown) {
    this.player.setVelocityY(175);
    this.stopSound(this.acelararSom);
} else if (this.cursors.right.isDown) {
    this.player.setVelocityX(225);
    this.stopSound(this.acelararSom);
} else if (this.cursors.left.isDown) {
    this.player.setVelocityX(-225);
    this.stopSound(this.acelararSom);
} else {
    this.player.setVelocityY(-25);
    this.player.setVelocityX(0);
    this.stopSound(this.acelararSom);
}
```

FIGURA 2.5: Movimentos do carro do player

De seguida, temos de ter inimigos, não poderia ser um jogo de traffic se não existissem inimigos, para isso criou-se um grupo onde se juntaram diversos inimigos, e também o grupo dos barris de combustível, que é outra variável que o jogador precisa de ter em conta.

```
// criar o grupo dos carros inimigos
this.cars = this.physics.add.group()
this.gasolGroup = this.physics.add.group()
this.gasolGroup.setVelocityY(50)
```

FIGURA 2.6: Criação do grupo de carros inimigos e combustível

A criação dos carros inimigos é a função mais complexa do jogo, uma vez que, os carros das três vias da esquerda são invertidos, ou seja, para fazer notar que estão em sentido contrário. Depois as lanes, em cada lane o carro pode nascer em pixeis diferentes, porque no início quando os carros eram gerados, estavam sempre na mesma posição, apesar de a ordem ser aleatória, a posição onde iriam nascer era sempre a mesma, e conseguimos dar a volta a isso, para evitar que o jogador fique sempre numa mesma posição.

```
createCar() {
    // Escolher uma via aleatória para adicionar o carro inimigo
    let transito = Phaser.Math.Between(1, 6);
    if (transito != 1){

        let lane = Phaser.Math.Between(1,6)
        while (lane == this.verificalane[this.verificalane.length-1]|| lane == this.verificalane[this.verificalane.length-2]){
            lane = Phaser.Math.Between(1,6);
        }
        this.verificalane.push(lane);
        let c = Phaser.Math.Between(1, 40);
        let y;
        //252 360 486 612 738 846
        switch (lane){
            case 1:
                y=250;
                break;
            case 2:
                y=370;
                break;
            case 3:
                y=500;
                break;
            case 4:
                y=620;
                break;
            case 5:
                y=750;
                break;
            case 6:
                y=870;
                break;

        }
```

FIGURA 2.7: Criação de carros inimigos

```
    //Mudar a posição dos carros na lane
    let lane2 = Phaser.Math.Between(1, 3);
    let y2;
    switch(lane2){
        case 1:
            y2 = 15;
            break;
        case 2:
            y2 = 0;
            break;
        case 3:
            y2 = -15;
            break;
    }

    // Adicionar o carro inimigo na posição aleatória
    if (c==1){
        this.car = this.cars.create(y+y2, 0, 'ambuinimiga');
        this.anims.create({
            key: 'sirene',
            frames: this.anims.generateFrameNumbers('ambuinimiga', { start: 0, end: 3 }),
            frameRate: 10,
            repeat: -1
        });
        this.car.anims.play('sirene', true);

        if (lane === 1 || lane === 2 || lane === 3) {
            this.car.setAngle(180); // inverte a sprite horizontalmente
          } else {
            this.car.setAngle(0); // mantém a escala normal
          }
          this.car.setScale(0.875);
```

FIGURA 2.8: Criação de carros inimigos

```
} else if (c>=1 && c<6) {
    this.car = this.cars.create(y+y2, 0, 'carroinimigo');
    if (lane === 1 || lane === 2 || lane === 3) {
        this.car.setAngle(180); // inverte a sprite horizontalmente
    } else {
        this.car.setAngle(0); // mantém a escala normal
    }
    this.car.setScale(0.275);
} else if (c>=6 && c<16){
    this.car = this.cars.create(y+y2, 0, 'carroinimigo2');
    if (lane === 1 || lane === 2 || lane === 3) {
        this.car.setAngle(180); // inverte a sprite horizontalmente
    } else {
        this.car.setAngle(0); // mantém a escala normal
    }
    this.car.setScale(0.9);
}else if (c>=16 && c<23){
    this.car = this.cars.create(y+y2, 0, 'motainimiga');
    if (lane === 1 || lane === 2 || lane === 3) {
        this.car.setAngle(180); // inverte a sprite horizontalmente
    } else {
        this.car.setAngle(0); // mantém a escala normal
    }
    this.car.setScale(0.8);
}else if(c>=23 && c<30){
    this.car = this.cars.create(y+y2, 0, 'camiao');
    if (lane === 1 || lane === 2 || lane === 3) {
        this.car.setAngle(180); // inverte a sprite horizontalmente
    } else {
        this.car.setAngle(0); // mantém a escala normal
    }
    this.car.setScale(0.7);
```

FIGURA 2.9: Criação de carros inimigos

```
}else{
    this.car = this.cars.create(y+y2, 0, 'carroinimigo3');
    if (lane === 1 || lane === 2 || lane === 3) {
        this.car.setAngle(180); // inverte a sprite horizontalmente
    } else {
        this.car.setAngle(0); // mantém a escala normal
    }
    this.car.setScale(0.9);
}
this.car.setCollideWorldBounds(true);
this.car.setBounce(1);
this.car.setCollideWorldBounds(false);
```

FIGURA 2.10: Criação de carros inimigos

Portanto, as peças principais do jogo estão alinhadas, o background, o carro principal, e os inimigos. Uma outra peça fundamental que o grupo pensou para aumentar a dificuldade do jogo, foi o combustível, que é algo semelhante com a realidade.

```
EventoGas(){
    let lane = Phaser.Math.Between(1, 6);
    let y;
    //252 360 486 612 738 846
    switch (lane){
        case 1:
            y=250;
            break;
        case 2:
            y=370;
            break;
        case 3:
            y=500;
            break;
        case 4:
            y=620;
            break;
        case 5:
            y=750;
            break;
        case 6:
            y=870;
            break;
    }
    const barrilGas = this.physics.add.sprite(y, 0, 'gas');
    this.anims.create({
        key: 'gasmov',
        frames: this.anims.generateFrameNumbers('gas', { start: 0, end: 3 }),
        frameRate: 6,
        repeat: -1
    });
    barrilGas.anims.play('gasmov', true);
    barrilGas.setCollideWorldBounds(false);
    barrilGas.setScale(0.15);
    barrilGas.setBounce(1)
    this.gasolGroup.add(barrilGas)

    let dl= Phaser.Math.Between(4000,10000);
    this.GasEvent.delay = dl;
}
```

FIGURA 2.11: Criação de gasolina

Depois, aquando da colisão, é atribuído 30 pontos de gasolina ao carro, e sempre que o jogador pressione a tecla para avançar mais rápido, o consumo aumenta.

```
//Colisões entre carros e inimigos ---- carros e gasolina
this.collisionPlayerCars = this.physics.add.collider(this.player, this.cars, this.gameOver, null, this)
this.collisionPlayerGas = this.physics.add.collider(this.player, this.gasolGroup, this.adicionaGas, null, this)
```

FIGURA 2.12: Colisões

Quando existe colisão entre o jogador e a gasolina é chamada a função **adicionaGas**.

```
adicionaGas(player, gas) {
    gas.destroy()
    this.collectGas.play()

    this.gasol = Math.min(this.gasol + 30, 100)
}
```

FIGURA 2.13: Função adicionaGas

```
else if (this.cursors.up.isDown){
    this.gasol -= 0.13;
}
else{
    this.gasol -= 0.065;
}
```

FIGURA 2.14: Update do combustível

A função Game Over quando existe a colisão entre carros é a seguinte.

```
// Mostrar a mensagem de fim de jogo
const gameOverText = this.add.text(170, 160, text , {
    fontSize: '49px Georgia',
    fill: '#ffa500',
    align: 'center',
    borderColor: '#fff',
    borderWidth: 10,
    padding: {
        left: 10,
        right: 10,
        top: 5,
        bottom: 5
    },
    shadowOffsetX: 5,
    shadowOffsetY: 5,
    shadowColor: '#FFF',
    shadowBlur: 5
});
this.timerEvent.remove(false);
this.timerEvent2.remove(false);
this.timerEvent3.remove(false);
this.scoreEvent.remove(false);
this.GasEvent.remove(false);
//Parar todos os sons
this.allSounds.forEach(sound => {
    sound.stop();
});
this.defaultMusic.stop()
```

FIGURA 2.15: Função GameOver

```
// Parar a movimentação dos objetos
this.player.setVelocityX(0);
this.player.setVelocityY(0);
this.cars.setVelocityY(0);
this.cars.setVelocityX(0);
this.gasolGroup.setVelocityY(0)
this.gasolGroup.setVelocityX(0)
this.timerEvent.remove(false);
this.timerEvent2.remove(false);
this.timerEvent3.remove(false);
this.scoreEvent.remove(false);
this.GasEvent.remove(false);

//Criar botão e mete lo interactive
const botao = this.add.image(550, 690, 'button');
botao.setScale(0.5)
botao.setInteractive();

//Clicar no botão e recomeçar jogo
botao.on('pointerup', function (event) { this.scene.start('load');}, this);
```

FIGURA 2.16: Função GameOver

Capítulo 3

Dificuldades

Inicialmente houveram algumas dificuldades no que toca à organização do código, uma vez que o grupo estava a fazer o código todo num só ficheiro. Mas depois de nos organizarmos melhor, decidimos reformular o código e desenvolver as cenas já mencionadas neste relatório, o processo de reformulação foi relativamente fácil, só a adaptação para as classes é que foi um pouco mais trabalhosa.

De resto, o jogo fez se relativamente bem, foi-se trabalhando todos os dias para que todos os pormenores estivessem alinhados.

Capítulo 4

Conclusões

Concluindo este projeto / jogo, o grupo acredita que correspondeu com o esperado, e com os objetivos delimitados, tendo também cumprido com alguns requisitos que no início eram considerados como extras.

Deste modo foi possível obter um conhecimento mais profundo da framework do Phaser, bem como conhecimentos de HTML, JavaScript e CSS. Sendo JavaScript a linguagem mais utilizada para o desenvolvimento deste jogo.