

## 2º Projeto de Introdução à Computação Gráfica

### ***NEON RACE RAVES***

**ABSTRACT** - This project consisted of the modelling and animation of a first-person retro game with *neon-vibing* colours, in which the objective is to earn points and to survive as long as possible, using mostly EdgesGeometry and LineSegments, an UnrealBloomPass for the glow effect and a PointerLockControls for the first-person camera.

#### I. INTRODUÇÃO

Este projeto foi desenvolvido aquando da unidade curricular de Introdução à Computação Gráfica da Licenciatura em Engenharia Informática da Universidade de Aveiro. O objetivo pretendido seria elaborar um videojogo, consolidando os conceitos utilizados para o primeiro projeto, bem como o uso de texturas, recorrendo à biblioteca JavaScript Three.js.

#### II. CONTEXTO

Este projeto foi inspirado na moda *neon* dos anos 80, usadas recorrentemente no videojogo *GTA Vice City* (2002), como é o caso da discoteca noturna *Malibu Club*:



Fig.1 - Entrada do *Malibu Club*



Fig.2 - Interior do *Malibu Club*

Um dos tipos de inimigos presentes no jogo, os monstros cor-de-laranja, foram baseados numa criatura presente no anime *Kokkoku* (2018):



Fig.3 - Criatura presente no anime *Kokkoku* (2018)

#### III. CONCEITO GERAL

*Neon Race Raves* é um jogo *retro* em primeira pessoa no qual o objetivo é apanhar o maior número de *checkpoints* possíveis (representados por uma seta amarela), ganhando pontos e não se deixando morrer para três tipos de inimigos/obstáculos: chão inexistente, bolas a moverem-se constantemente e monstros que perseguem o jogador. Outra forma de ganhar pontos (ainda que em menor valor) ocorre quando um monstro cai num dos buracos existentes no jogo.

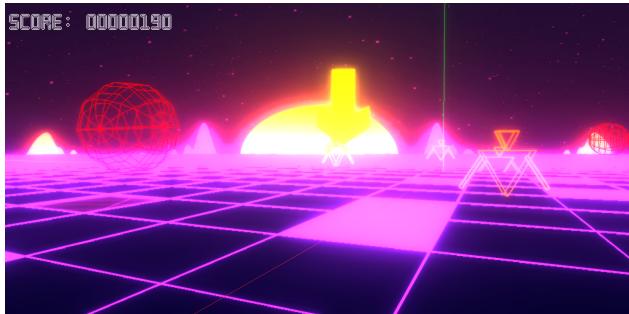


Fig.4 - Layout de *Neon Race Raves* (2021)

#### IV. CONTROLOS E INTERAÇÃO

Os controlos do jogo são bastante intuitivos, visto cingirem-se a W/A/S/D para a movimentação do jogador, o recurso ao cursor para movimentação da câmara e a tecla *Shift* para mover-se mais rapidamente. Estes controlos estão descritos num Menu de Controlos, que aparece ao clicar-se duas vezes em *Esc*:



Tab.1 - Controlos criados

#### V. IMPLEMENTAÇÃO

- Câmara

Foi usado um **PointerLockControls** para simular o movimento e câmara em primeira pessoa.

- Cenário

Para o fundo da cena, foi usada uma **VideoTexture** que repete em loop uma imagem GIF que fora convertida em vídeo.

O chão do jogo é formado por vários planos agregados - recorrendo a **PlaneGeometry** e **EdgesGeometry**; o interior dos planos é a escolha aleatória entre duas cores -

violeta, a cor usada para os limites dos segmentos dos planos, ou um azul-escuro.

Para o efeito de *glow*, foi usado o **UnrealBloomPass** (e outras dependências associadas).

NOTA: Neste projeto, e devido ao seu tema-*neon* característico, foram utilizados **BasicMaterials** e **LineSegments**, pois usando PhongMaterial o efeito de *glow* tornava-se exagerado.

- Checkpoint

O checkpoint consiste numa seta de cor amarela, elaborada através de uma **ExtrudeGeometry**, presente na mesma posição que uma **PointLight**; este foco de luz está a ser usado para determinar mais facilmente a *hitbox* necessária para saber se o jogador chegou ou não ao checkpoint.



Fig.5 - Exemplo de checkpoint

- Inimigos

Existem três tipos de inimigos presentes no jogo:

- ❖ Chão inexistente - já referido anteriormente na secção III, um plano que se tornou invisível; vê-se o fundo animado da cena por entre o buraco apresentado;

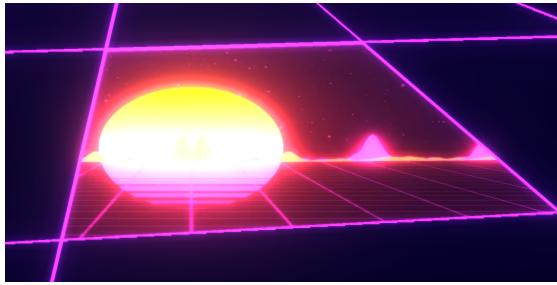


Fig.5 - Exemplo de chão inexistente

- ❖ Esferas vermelhas - elaboradas através de **SphereGeometry** e de **EdgesGeometry**;

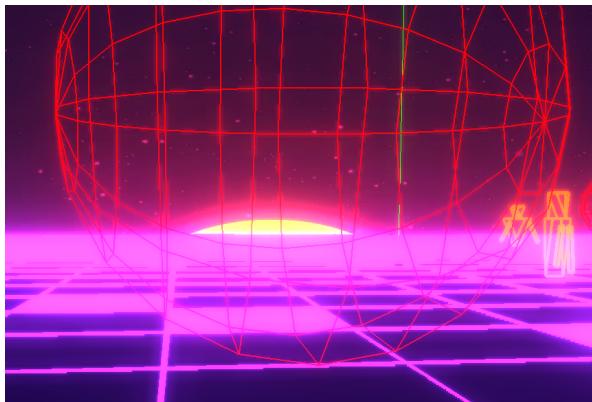


Fig.6 - Exemplo de esfera

- ❖ Monstros - de cor laranja, feitos a partir de **ExtrudeGeometry**, **BoxGeometry** e de **EdgesGeometry**; note-se a sua silhueta, maior no topo do tronco e a diminuir em direção à base, semelhante ao que ocorre na Fig.3 da secção II.

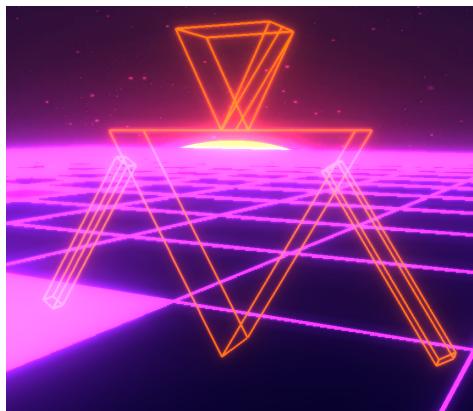


Fig.7 - Exemplo de monstro

## VI. SOM

Neste jogo foram utilizados vários sons: uma música eletrónica *retro* de fundo para ambientalizar o jogador e vários sons associados à jogabilidade - quando o jogador apanha um checkpoint, quando cai num buraco, ou quando é apanhado tanto por uma bola como por um monstro.

Se algum dos áudios não estiver a funcionar corretamente (ou o fundo aparecer a negro ou não estar animado), deve, antes de abrir o **Google Chrome** no **Ubuntu**, fechar todas as janelas do navegador e no terminal executar **google-chrome --autoplay-policy=no-user-gesture-required**.

## VII. ANIMAÇÃO

- Score

No canto superior esquerdo existe um painel que mostra o score atual do jogador, que sempre que o jogo é iniciado começa a zero e vai sendo atualizado à medida que vai coletando checkpoints, ou monstros morrem. Este score foi feito através de CSS, sobrepondo o “parágrafo” à cena do ThreeJS, através de *position: absolute* e de *z-index: 99*.



Fig.8 - Painel do Score atual

- Chão inexistente/jogador cair num buraco

O jogo verifica constantemente sob que plano é que o jogador se encontra; se acontecer que este esteja sob um plano ao qual o atributo **visible** seja **false**, o jogador começa a cair (diminuindo a sua posição-Y sucessivamente), até perder todos os seus pontos e retomar à sua posição inicial predefinida.

- Esfera

A cada esfera, aquando da sua criação e inserção na cena, está associada uma direção aleatória inicial na qual ela irá movimentar-se: **x**, **-x**, **z** e **-z**. Se uma esfera chegar a um dos limites do cenário (se chegar a uma das pontas do aglomerado de planos), esta muda o sentido do seu movimento, mantendo a sua direção. Cada esfera tem igualmente uma rotação associada para simular uma esfera real a rodar ao longo de um plano.

Quando o jogador é atingido por uma bola, este é redirecionado para a sua posição inicial e perde todos os seus pontos.

- Monstro

Todos os monstros são criados previamente aquando do início do jogo e colocados numa posição aleatória, mas apenas se tornam um perigo para o jogador (e são “jogáveis”) quando se tornam visíveis, isto é, quando passado um tempo aleatório o seu atributo **visible** passa a ser igual a **true**; a partir daí, e para cada monstro, está-se constantemente a determinar o vetor unitário que indica a direção entre este e a posição onde o jogador se encontra. A posição de cada monstro é atualizada somando à sua corrente posição este mesmo vetor. Assim, todos os monstros perseguem o utilizador, que ao ser tocado por um deles, retoma à sua posição original e perde todos os seus pontos.

Como já foi referido anteriormente, na secção III, os monstros podem ser “derrotados” se caírem dentro de um buraco - a sua posição-Y vai diminuindo até chegar a um dado limite que, após ser atingido, este volta para a sua posição-Y inicial e para posições-X e Z aleatórias, voltando novamente a ter um **visible=false**.

#### IX. TRABALHO FUTURO

- Adicionar novos items possíveis de serem “coletados”, como poderes especiais (*shield*, inexistência de buracos por um período de tempo, esferas e monstros deixam de se mover por um tempo, etc.)
- Adicionar novos tipos de inimigos, como objetos a cair
- Entre outros.