ASTRODJER

Rui Oliveira 89216
Rui Santos 89293

Resumo **- O presente artigo apresenta o desenvolvimento de uma aplicação 3D intuitiva que permite jogar o jogo Astro Dodger utilizando a API JavaScript WebGL.**

O artigo começa por estabelecer as principais regras do jogo e, de seguida, são apresentados os principais requisitos da aplicação 3D considerados ao longo do desenvolvimento do projeto e como foi feita a sua implementação. Por último, descrevem-se os resultados obtidos.

Abstract **- This article presents the development of an intuitive 3D application that allows you to play the Astro Dodger game using WebGL JavaScript API.**

This article begins by establishing the main rules of the game and, afterwards, the main requirements considered for the 3D application throughout the project development are described and after that, their implementation. Finally, the obtained results are described.

I. INTRODUCTION

Astrodger é um jogo 3D onde o utilizador tem o papel de controlar uma nave que se encontra no espaço e o objetivo é chegar o mais longe possível desviando-se de vários asteroides que aparecem no seu caminho.

A. Espaço

O espaço do jogo está dividido em 4 linhas e 4 colunas formando assim 16 corredores onde se encontram os objetos. Cada corredor tem uma abscissa (X) e uma ordenada (Y) constantes mas a sua profundidade (Z) é variável.

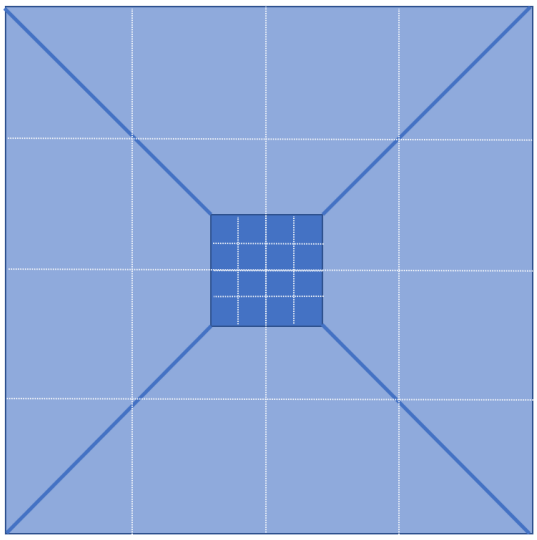

Figura 1 – Organização do espaço

B. Nave

O único modelo 3D que o utilizador tem controlo é a nave. A nave encontra-se inicialmente numa das 16 posições possíveis que as 4 colunas sobre 4 linhas formam, no entanto esta tem a sua profundidade (Z) com um valor fixo (0) e o utilizador só pode fazer variar a abscissa (Z) e a ordenada (Y) da mesma.

C. Asteroides

Os asteroides apenas se movimentam num único corredor aproximando-se mais da profundidade (Z) da nave ao longo do tempo, ou seja, as suas abcissas (X) e ordenadas (Y) são constantes e a sua profundidade (Z) é variável e gradual.

A frequência com que os asteroides surgem aumenta com o decorrer do jogo bem como a velocidade dos mesmos.

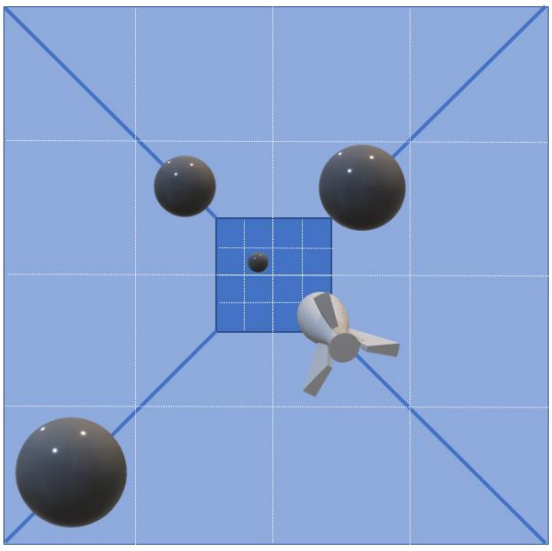

Figura 2 – Organização dos objetos

D. *Colisões*

Sempre que um asteroide colidir com a nave o jogo termina.

E. *Níveis*

À medida do tempo, considerando que o utilizador está a conseguir desviar-se dos asteroides, a dificuldade irá aumentar progressivamente (número de asteroides e velocidade)

II. REQUISITOS

A. *Requisitos do Jogo*

Os requisitos de jogo apresentados na Introdução foram cumpridos e implementados, em específico:

- Visualização do espaço e nave
- Geração aleatória dos asteroides
- Seleção da posição da nave
- Validação da posição da nave:
  Não pode ocupar nenhuma posição para além das 16 (4x4) posições permitidas.
  Não pode ocupar uma posição já ocupada por um asteroide
- Manutenção de um sistema de níveis baseado no tempo de jogo decorrido
- Sistema de menus (*Start, Pause, Resume, Restart*)
- Pequena explicação/exposição dos controlos

B. *Requisitos da Aplicação 3D*

O objetivo do projeto é o desenvolvimento de uma aplicação 3D interativa para jogar o jogo *Astro Dodger*. Isto implica a consideração de um conjunto de requisitos adicionais:

- Todos os objetos devem ter 3 dimensões
- Deverá haver variação nas 3 coordenadas dos - vários objetos, ou seja, movimento nos 3 eixos
- Deverão existir rotações aplicadas aos objetos
- Deverá existir iluminação
- Cada objeto deve refletir a luz de forma distinta
- Deverão existir pelo menos 3 objetos distintos

III. IMPLEMENTAÇÃO

A. *Modularização*

index.html	Página HTML onde o JavaScript é executado e através do qual é realizada toda a interação entre o sistema e o utilizador
astrodger.js	Execução da aplicação (*canvas*, *shaders*, leitura das teclas e rato, menus de controlo, gerador de asteroides, iluminação, texturas, desenho dos modelos, rotações...)
initShaders.js	Funções para criação e *linking* dos *shaders*
lightSources.js	Definição da iluminação do *canvas* - definição dos diversos pontos de luz e as suas propriedades (cor, posição, rotação, ...)
maths.js	Funções matemáticas auxiliares
models.js	Funções para processar modelos de malha de triângulo
sceneModels.js	Definição dos vários objetos bem como os seus atributos e a respetiva inicialização
webgl-utils.js	Este arquivo contém funções que todo programa *webgl* precisará.
background.jpg	Imagem de fundo do jogo
ice.jpg	Imagem com textura de gelo para os asteroides em forma de tetraedro

Tabela 1 - Modularização

B. *Implementação*

1. Movimentação

O espaço de jogo está dividido em 16 corredores com abscissas (X) e ordenadas (Y) fixas. Tanto as naves como os asteroides têm de estar num destes 16 corredores, ou seja, as suas coordenadas X e Y têm de ser iguais às coordenadas do respetivo corredor.

Assim, existem 4 valores possíveis para as coordenadas X e Y: -0.75, -0.25, 0.25, 0.75. A decisão de colocar um número par de valores possíveis para estas coordenadas foi resultado de um estudo prévio onde se concluiu que no caso dos mesmo serem ímpares, os objetos ficariam bastante grandes durante uns instantes e impediriam o utilizador de ver o jogo.

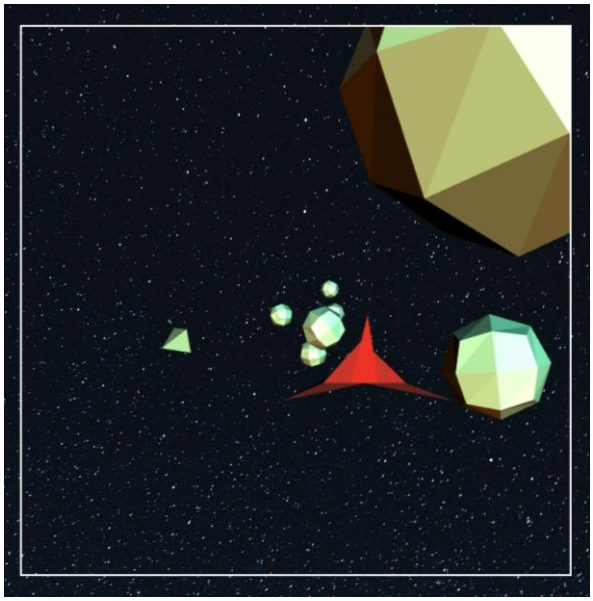

Figura 3 – Campo de visão

Relativamente à profundidade (Z), é constante no caso da nave (assume sempre o valor 0) e varia gradualmente no caso dos asteroides. Para garantir este movimento no eixo ZZ, é calculada a nova coordenada Z para cada asteroide ativo:

```
sceneModels[i+1].tz += asteriodeSpeed * elapsed / 100;
```

Cada objeto é definido no ficheiro "sceneModels.js" e tem um atributo *isActive* que assume o valor *true* no caso deste objeto estar presente no canvas e, no caso contrário, assume o valor *false*.

2. Spawn de Asteróides

Para a inserção de cada asteroide na área de jogo foi utilizado um algoritmo desenvolvido por nós que consiste num contador *spawnCount* , que assume o valor inicial 0, e é incrementado a cada *tick*. Quando este contador assume um valor igual à variável *spawnTop*, cujo valor inicial é 50, o seu valor passa a ser 0, é chamada a função *getFreeIndex*() que devolve o índice de um asteroide livre, e é alterado o atributo *isActive* do asteroide.

Neste processo são também gerados outros valores aleatoriamente como as coordenadas X e Y do asteroide a ser inserido, que têm de assumir um dos valores: -0.75, -0.25, 0.25, 0.75.
São também gerados os valores relativos à rotação do asteroide nos eixos XX e YY sendo o sentido e a velocidade aleatórios. Não se optou por colocar a rotação em mais do que 2 eixos visto que, através de testes, se concluiu que a rotação nos 3 eixos dava um aspeto pouco real.

```
if (spawnCount >= spawnTop) {
    spawnCount = 0;
    var j = getFreeIndex();
    if (getFreeIndex() != -1) {
      sceneModels[j].isActive = true;
      sceneModels[j].tx = (Math.floor(Math.random() * 4) * 0.5) -0.75;
      sceneModels[j].ty = (Math.floor(Math.random() * 4) * 0.5) -0.75;
      sceneModels[j].tz = -13;
      var x = (Math.floor(Math.random() * 2));
      if (x == 0) x = -1;
      var y = (Math.floor(Math.random() * 2));
      if (y == 0) y = -1;
      sceneModels[j].rotXXDir = x;
      sceneModels[j].rotYYDir = y;
      sceneModels[j].rotXXSpeed = (Math.floor(Math.random() * 6) + 1) / 5;
      sceneModels[j].rotYYSpeed = (Math.floor(Math.random() * 6) + 1) / 5;
    }
  }
```

3. Controlo

O controlo da nave é feito através de duas formas:

Teclado
Através das teclas (ASDW) é possível deslocar a nave para a respetiva posição válida. No caso da nave se encontrar num dos limites do espaço de jogo, quando o utilizador clica na tecla com o sentido oposto ao centro, a nave não se move.

Rato
Através do rato, o utilizador poderá clicar numa qualquer posição do canvas e a nave irá mover-se para a posição válida mais próxima do clique.

4. Modelos 3D

Existem 3 modelos 3D distintos no Astrodger.

A. Nave
A nave é uma peça fundamental do jogo visto que é o único modelo que o utilizador tem controlo.
Esta foi pensada e mais tarde desenhada por nós mesmos.

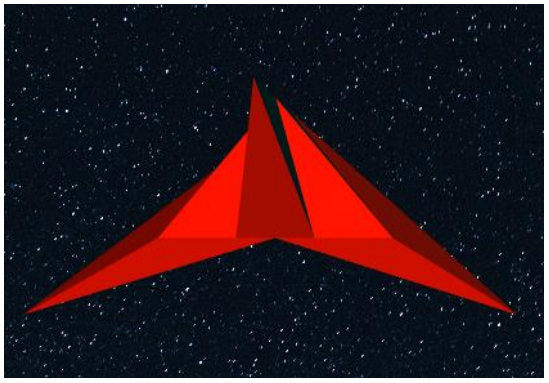

Figura 4 – Nave

B. Asteroides

Existem dois tipos de asteroides:
Esferas com número reduzido de triângulos para dar ideia de rugosidades tal como asteroides reais.

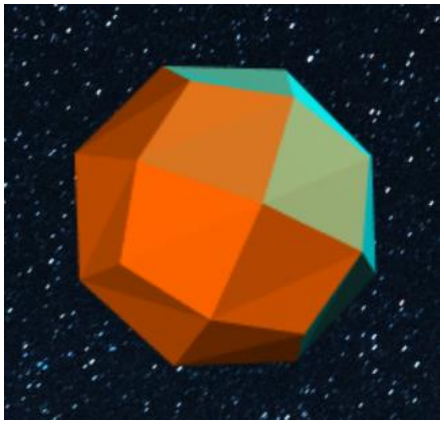

Figura 5 – Asteroide Esfera

Os **tetraedros** são modelos simples de asteroides para ser possível aplicar a textura de gelo.

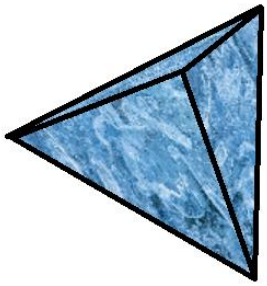

Figura 6 – Asteroide Tetraedro (Traço negro adicionado para melhor compreensão da imagem)

Todos os modelos estão devidamente descritos no ficheiro *sceneModels.js*
Neste encontram-se definidos os vértices, as normais, os vértices das texturas e respetivos índices. Podemos ver nas suas instanciações as suas coordenadas, escalas, rotações, cor e se está ativo (esta aplica-se consoante a profundidade z do asteroide). No caso da esfera está também declarado o nível de subdivisões visto que resulta originalmente de um cubo.
Assim o *array sceneModels[]* tem 30 elementos no máximo: uma nave e vinte e nove asteroides (3 tetraedros e 26 esferas). O modelo *sceneModels[0]* representa a nave e os restantes (*sceneModels[1]...sceneModels[29]*) os asteroides.

5. Colisões

Sempre que a nave colide com um asteroide é registada uma colisão.
Não só o embate frontal será registado como uma colisão, como também os embates laterais levarão ao término do jogo.
Considerando o que o ponto (X, Y, Z) representa o centro de um modelo, sempre que o x e y da nave forem iguais ao de um asteroide e o z menor que a distância mínima que separa os dois modelos, é registado uma colisão e é ativada a *flag gameover*.

6. Dificuldade

O jogo torna-se mais difícil à medida que o score e o nível aumentam e consiste em incrementos na frequência de *spawn* de asteroides e nas suas velocidades.
Para aumentar a frequência de *spawn* dos asteroides é diminuído o valor da variável *spawnTop* fazendo com que o *spawnCount* atinja este valor mais rapidamente e que um novo asteroide seja ativado e inserido no *canvas*.
A velocidade é aumentada através da variável *asteroideSpeed* que tem o valor inicial de 0.8 e é utilizada no cálculo das próximas posições dos asteroides. Com o decorrer do jogo, esta variável aumenta e provoca uma maior velocidade dos asteroides.
Este processo é feito através de uma condição que verifica se o resto da divisão inteira do atual score com 500 é igual a 0 e se o valor da variável *spawnTop* é superior a 5. No caso desta condição se verificar, é subtraído 5 ao valor da variável *spawnTop* e somado 0.03 ao valor da variável *asteroideSpeed*.

```
if ((score % 500 == 0) && (spawnTop > 5)) {
    spawnTop = spawnTop-5;
    asteriodeSpeed += 0.03;
 }
```

7. Apresentação da página Web

Em termos de apresentação a página Web está dividida em três colunas.

Na da esquerda estão os menus. Aqui poderá encontrar-se os botões:

A. *Start* (inicia o jogo)
B. *Pause* (pausa o jogo)
C. *Resume* (retorna o jogo no momento em que anteriormente se tinha pausado)
D. *Restart* (retorna o jogo ao estado inicial restabelecendo a iluminação, número de asteroides e consequentemente o nível/dificuldade)

Os menus têm um papel fundamental na aplicação pois servem de controlo para definir o que é possível fazer em determinado momento.

Na coluna central encontra-se o nome do jogo e o *canvas* que é o jogo em si. Aqui o utilizador poderá controlar a nave (setas e rato) e poder visualizar os asteroides a mover-se.
Sempre que o jogador perder (colisão com asteroide) irá surgir uma informação "GAME OVER!" por baixo do quadrado do *canvas*.
Na coluna da direita é possível visualizar informações relativas à dificuldade (*Score, Highest Score e Level*) e um pequeno painel de instruções para os controlos.

Para tornar a experiência de utilização o mais imersiva possível, optamos por colocar um fundo do espaço com estrelas.

8. Iluminação

A iluminação dos objetos do jogo é feita através de 2 pontos de luz definidos no ficheiro *lightSources.js.* O objetivo seria fazer com que estes pontos de luz se assemelhassem a estrelas e portanto assumem as cores laranja e azul marinho.

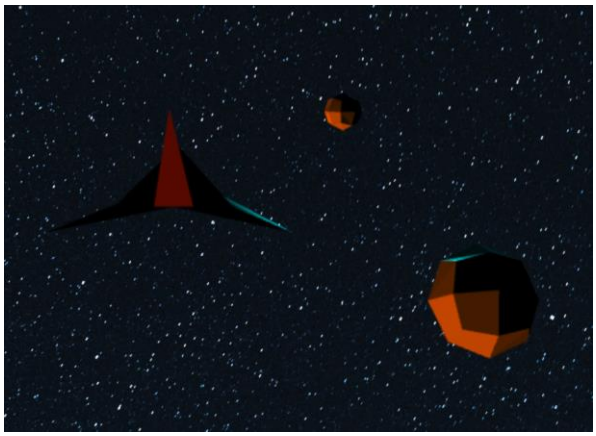

Figura 7 – Iluminação inicial

Para dar o efeito da nave se estar a mover, foi adicionado um movimento no eixo ZZ em ambos os pontos de luz.

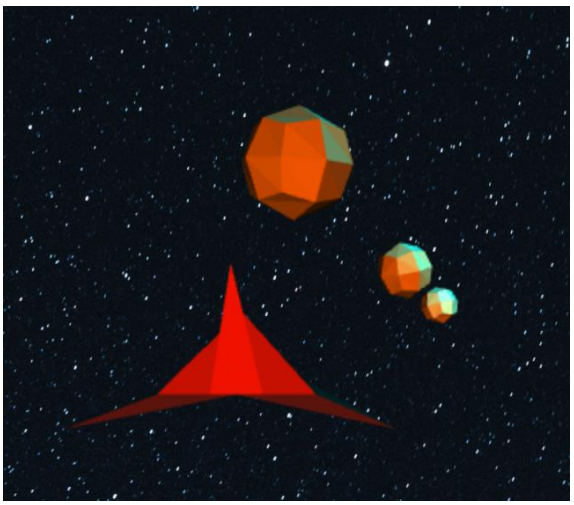

Figura 8 – Iluminação final

9. Texturas

Foi aplicada uma textura de gelo nos asteroides em forma de tetraedro.
Assim, definiu-se os vértices das texturas e vértices dos índices a todos os modelos de tetraedros e inseriu-se uma imagem (gelo.jpg) igual para todas as faces.
Apesar de termos testado noutro ficheiro e estar a funcionar, no final acabámos por não conseguir aplicar para o nosso programa principal. Ficámos sem perceber o porquê de não funcionar.
Fizemos as configurações no *shader-vs* e *shader-fs*, alterámos o *initShaders.js* conforme o previsto, carregámos também os *buffers* com as texturas relativas aos vértices de texturas e respetivos índices e criamos a textura do gelo.

Infelizmente não encontrámos a potencial falha que está a causar a erro no carregamento das texturas nos tetraedros, no entanto o código todo desenvolvido fica nos ficheiros.

IV. RESULTADOS

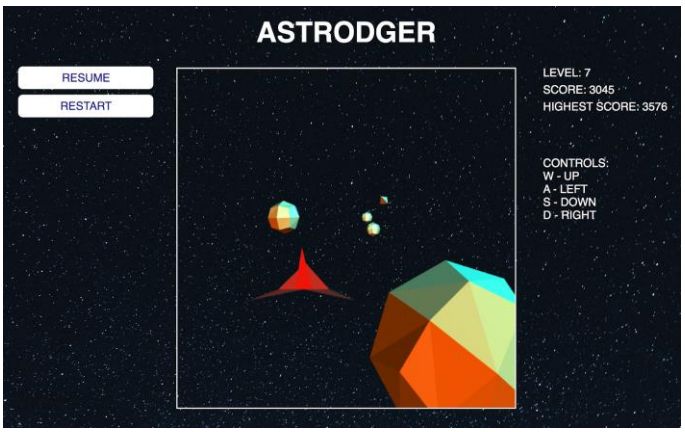

Figura 9 – Dificuldade inicial

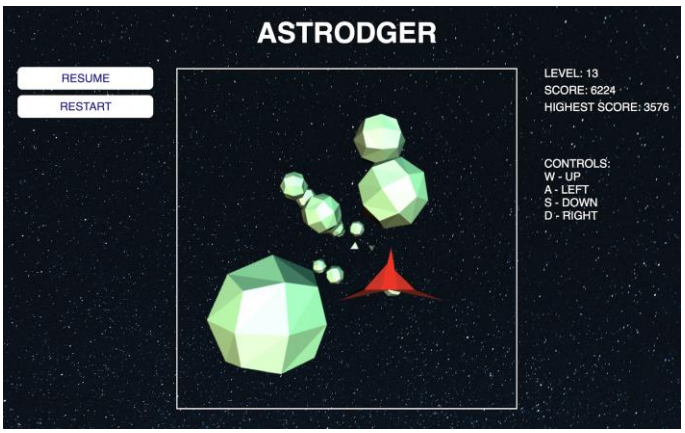

Figura 10 – Dificuldade inicial

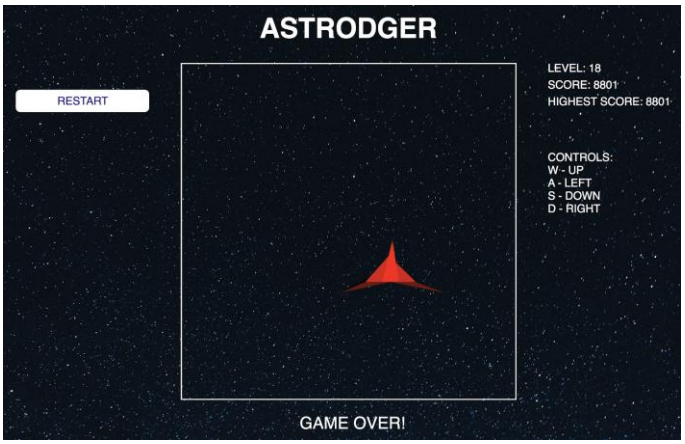

Figura 11 – Game Over

V. REFERÊNCIAS

1. Material fornecido no âmbito da unidade curricular de Computação Visual lecionada pelos professore Joaquim Madeira e Paulo Dias