E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Computação Gráfica

CÓDIGO: 21020

DOCENTE: Pedro Pestana e António Araújo

A preencher pelo estudante

NOME: Mário Pedro Capela Rodrigues Carvalho

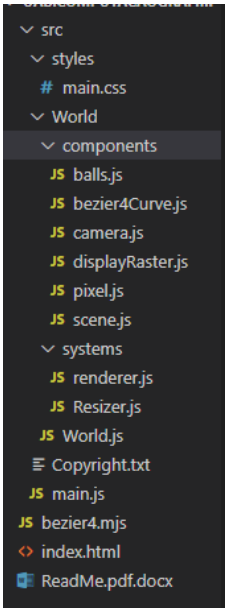
N.º DE ESTUDANTE: 2000563

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 8 de janeiro de 2023

TRABALHO:

Este relatório tem como objetivo de explicar o trabalho realizado em paralelo, até ter sido alcançado o objetivo final do e-fólio em questão. Este, demonstra assim, o que fora pensado em cada etapa, e como se foi ponderando cada parte da sua construção, para ser realizado com sucesso. Assim, antes de se iniciar qualquer parte de código, ou reproduzir as condições necessárias, teve-se em atenção em corrigir o problema de fluidez não detetado anteriormente, com uma nova implementação do "orbitControls", e de seguida, manteve-se a arquitetura do e-fólio anterior, eliminando apenas os ficheiros desnecessários, e adaptando/ restruturando as partes necessárias para responder ao pedido no enunciado: organização das pastas, nomes, recursos aos "imports" necessários – CDN, three.js, ...



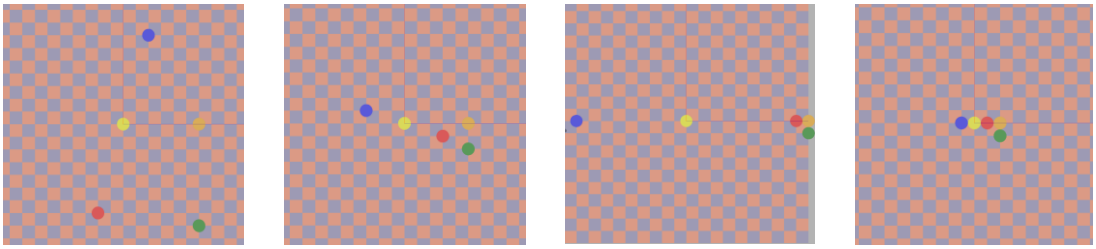
Toda a restante estrutura foi sendo realizada por partes nos respetivos ficheiros, com importação de uma estrutura idêntica e bem dividida segundo a documentação de "threejs.org", com os ficheiros pedidos (index.html, bezier4.mjs, ReadMe.pdf) na sua base, e os restantes, subdivididos dentro de "./src", onde constam três grupos : "./styles" para o ficheiro correspondente a ".css", "./World", que tem "./components" e "systems", e "main.js", que importa o ficheiro "World.js". Esta subdivisão, tal como indicada na documentação de "three.js", permite de forma mais segura, limitar o acesso às propriedades de cada objeto a partir de uma fonte externa, e simultaneamente, facilitar a sua importação para outro projeto de forma idêntica, sem que esta sofra modificações ou altere o projeto seguinte.
Deste modo, em "./systems", correm os ficheiros "renderer.js", e "Resizer.js". Em "./components", correm os ficheiros "camera.js", "scene.js", "./pixel.js", "./displayRaster.js", "/balls.js" e "/bezier4Curve.js".

Além de se ter atribuído particular atenção à Parte I, para se garantir que o cálculo da função de bezier quártica, permitiria calcular com certeza a curva, pelos pontos *Ci*, realizaram-se vários cálculos posteriormente aquando da sua implementação da função de "bezier4.mjs", no respetivo ficheiro. Uma vez que a Parte I – Implementação do algoritmo da curva de bézier quártica, foi concluída, iniciou-se a implementação da Parte II – Interface Gráfico. Para melhor compreensão de qual o ficheiro que estaria a ser usado, e tornar mais visual a ordem em que cada ficheiro se processava, foi introduzido um "console.log(nome-do-ficheiro-aberto)" indicativo com o nome de cada ficheiro.

```
Check oppening file: camera.js                                  camera.js:12
Check oppening file: scene.js                                   scene.js:11
Check oppening file: grid.js                                    pixel.js:12
Check oppening file: balls.js                                   balls.js:11
Pontos gerados aleatóriamente para x 5 e y: 6                   balls.js:25
Check oppening file: bezier4.mjs                                bezier4.mjs:11
Check oppening file: displayRastrer.js                          displayRaster.js:12
Check oppening file: renderer.js                                renderer.js:11
Check oppening file: Resizer.js                                 Resizer.js:11
Check oppening file: World.js                                   World.js:14
Check oppening file: main.js                                    main.js:12
```

De forma a continuar com a estrutura aconselhada por ThreeJs.org, e dado que seria necessário manter o tabuleiro do e-fólio precedente, foi acrescentado à pasta "\components", o ficheiro "balls.js". Consiste em conceber os parâmetros de cada bola, em termos de nome, cor e posição, para que, só na sua importação para o ficheiro "displayRaster.js", é que estas sejam criadas, segundo o seu objeto. Segundo o enunciado, interpretou-se que as bolas teriam de estar num primeiro momento, no plano, mas que, a cada início a posição das suas coordenadas (x,y), seriam aleatórias dentro do tabuleiro. Para facilitar a sua implementação, criou-se neste momento, uma coordenada *random* para x e outra para y, de forma a obter números exatos (int). Como se tratam de cinco bolas diferentes, e existem apenas 4 combinações de coordenadas diferentes possíveis (para dois números, em positivo e/ou negativo), recorreu-se (por tentativas e erros) a subtrações e /ou somas com um ou os dois números, o próprio numero obtido, e/ou

alteração de sinais, para que se conseguisse colocar 4 das 5 bolas em posições diferentes entre si, e que tendencialmente não ficassem no ponto 0 nem no mesmo ponto, desde que, ficassem sempre dentro do tabuleiro. A esta etapa dedicou-se algum tempo, pois, o fenómeno aleatório não propriamente, matemático.



Recorreu-se a “camara.js”, para obrigar que, ao se iniciar a página, se observe como posição central do tabuleiro, a posição (0,0,0) onde se cruzam os 3 eixos, a incidir no ponto de cruzamento dos quatro pixels centrais, com recurso a “camera.lookAt()”. Como neste caso os pontos centrais entre eixos e coordenadas centrais, não se encontravam alinhados no mesmo ponto do tabuleiro (e que este, também teria de estar centrado), recorreu-se para isso, à mudança de pontos iniciais de cada pixel do tabuleiro, para um deslocamento de 0.5 em x e em y. Tendo para isso, que se manter o ficheiro “pixel.js” como este se encontrava para a construção do tabuleiro.

A restante implementação, foi realizada diretamente no “displayRaster.js”, que permitiu acrescentar e manipular a restante construção: bolas, eventos para a interação com o utilizador, e os recursos de visualização (linhas perpendiculares, tubo, ..).

```
/* Funções que permitem integrar na scene todos os materiais, objetos, posições e interações */
/* Para impor os eixos xx, yy e zz, no tabuleiro, com as cores */
function createAxis() {···
}

/* Para criar o tabuleiro a partir de cada pixel, defenindo a dimensão do tabuleiro, do pixel, a posição nos eixos e com isso, a sua cor */
function createBoard() {···
}


/* Função que cria uma bola à vez*/
function createSphere (ballC){···
}

/* Função que cria as bolas todas, e faz com que estas sejam inseridas no tabuleiro numa posição inicial
aletóriaa, com as cores respetivas, a partir do array correspondente */
function createBalls() {···
}

/* Função que recebe um número e seleciona a bola que corresponde a esse numero */
function ballSelected(i) {···
}

/* Função que grava as coordenadas da bola selecionada e tira a seleção de bola */
function ballDeselected() {···
}

/* Atualiza as coordenadas da bola selecionada no ecrã (para atualizar com o movimento do rato, ou quando se pressiona w ou s) */
function updateCoordenates() {···
}

/* Para desenhar a a curve Bezier no tabuleiro segundo a posição das bolas */
function createBezierCurve() {···
}

/*======================EVENTS======================*/
}

/* Para detetar movimentação do rato */
function onMouseMove(event) {···
}


/* Para detetar seleção de teclas */
function onDocumentKeyDown(event) {···
}


/* Para criar o tabuleiro no displayRaster segundo os requesitos estipulados */
function createDisplayRaster(_scene, _camera, _renderer, _controls) {···
}

export { createDisplayRaster };
```

Começou-se por adicionar um eixo zz aos restantes existentes. Para este último, assumiu-se que ficaria com uma cor diferente das restantes. Por mais que tenha semelhanças com a função original de “axesHelper()” preferiu-se adicionar mais uma construção, em vez de utilizar algo próprio da biblioteca de “Threejs”. Desta forma, conseguiu-se até readaptar o tamanho destas linhas à nova dimensão do tabuleiro. Que, também foi adaptado de forma a ter o novo tamanho pedido, com as respetivas cores estabelecidas no e-fólio A. Nesta fase detetou-se que o eixo central se encontrava deslocado da posição com coordenadas (0,0,0), e por isso, preferiu-se adaptar a posição global no espaço de cada pixel. E assim, alterar em “/pixel.js” a posição inicial de x e y, com um ligeiro desvio de + 0,5.
Criaram-se novas funções responsáveis pelo surgimento da estrutura de cada bola (através da função “createSphere()”) em Threejs, que permitisse dar uma forma e aspeto, consoante as condições pré-determinadas em “/balls.js”, e, de seguida, aplicar a sua construção (individual, via “createBalls()”) para tornar possível o seu surgimento no tabuleiro, depois de ser adicionado o array respetivo (balls) à funcionalidade recurso “scene.add()”.

Para que fosse possível ocorrer uma seleção de forma individual de cada bola (via events), criou-se uma função que, permite que se uma bola for selecionada, o número da bola seja reconhecido, esta se torne opaca (para mostrar que foi selecionada). Aqui, surgiu o problema entre o valor do index (1, 2, ..5) em que a bola é guardada na array responsável por enviar a bola para o tabuleiro, e a verdadeira identificação (C0, C1, .. C4). Por uma questão de facilidade, em vez de se fazer a sua atualização ao nível do event respetivo, preferiu-se corrigir o valor depois desta função ser chamada, e alterar o seu valor no interior. Pois, esta função é a responsável pela seleção da bola e não o event. Como se pedia para apresentar no tabuleiro a posição de cada bola, isto é, cada coordenada quando a bola fosse selecionada, procedeu-se à criação de uma dentro desta função também, para que esta função fosse a principal responsável pela seleção de cada bola, e portanto, a sua demonstração ao utilizador.

Do mesmo modo que uma bola pode ser selecionada, esta também pode ser desselecionada via a utilização da tecla “space” (com método implementado no event respetivo). Aqui, procedeu-se à implementação de métodos inversos aos criados na função anterior, começando por permitir que a bolas adquiram esta nova posição, e voltam a ficar transparentes. Obrigando ao surgimento de uma linha indicativa da movimentação de uma bola, quando a esta fosse aplicada um deslocamento o longo do eixo dos zz (seja para cima ou baixo do tabuleiro). Sendo esta linha, meramente indicativa de um deslocamento face à perpendicular do tabuleiro, foi-lhe contruída de modo que a maneira de aparecer fosse apenas dependente da fixação da sua posição, e com uma cor alheia, neste caso, branca para se distinguir das restantes. Foi aqui também que se optou por fazer desaparecer as informações sobre as coordenadas das bolas no ecrã, recorrendo à remoção da criada na função anterior.

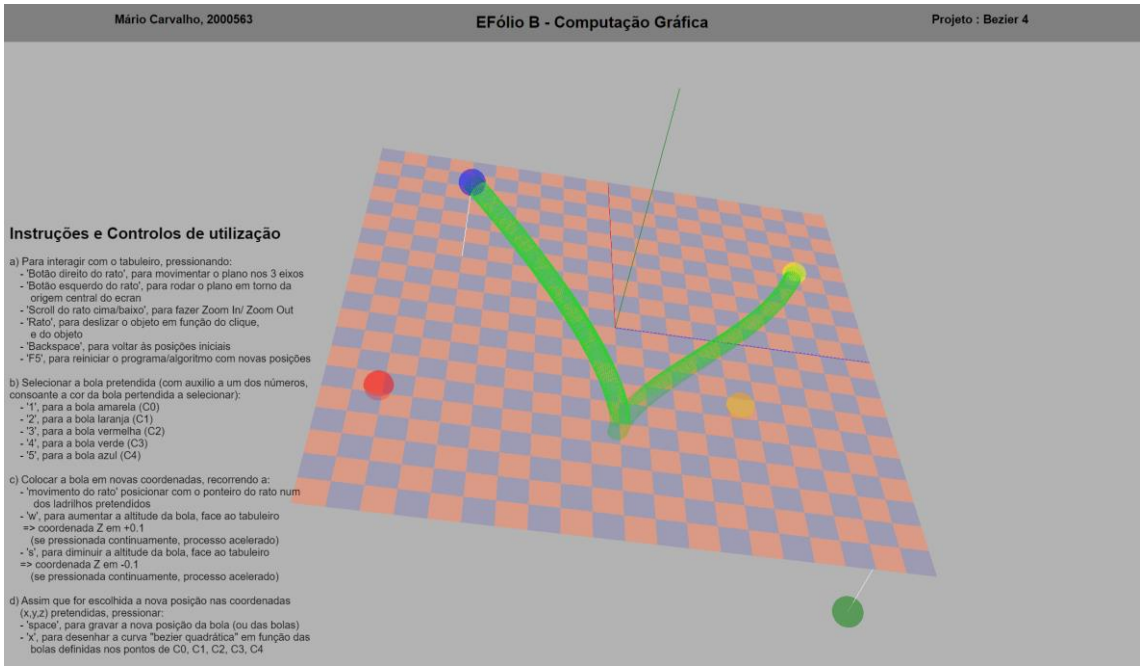
Sendo que, entre a criação desta e o seu desaparecimento, os valores teriam de ser atualizados de modo continuo, enquanto uma bola estivesse selecionada. Para isso, criou-se a função “updateCoordenates”, que é responsável por atualizar os valores destas posições (x, y, z) de modo constante, desde que exista .

Como pedido inicialmente, assim que as bolas estivessem criadas e colocadas no tabuleiro, a curva de bézier deveria surgir no plano (quando pressionada a tecla “x”), com uma forma determinada pela posição de cada uma destas bolas. Portanto, para facilitar a implementação da curva, criou-se um ficheiro em “/componentes”, que de forma independente utiliza-se os resultados da parte I, originarias da função “bezier4mjs”, para proceder à sua construção.

Para que a interação fosse possível, tal como descrita, foi necessário manter a função createEvents(), e onWindowResize(), de forma a permitir a atualização da dimensão do tabuleiro e a leitura das teclas por parte do utilizador. Aqui, deparou-se a dificuldade de fazer match entre as posições da seta do rato, sobre o tabuleiro, combinado com as bolas. Para contornar estre problema, procedeu-se à implementação de um método que intersetasse estes três objetos.

A última função de events, é relativa a cada tecla que o utilizador possa utilizar para interagir com o programa, tal como a descrição apresentada na própria página. Como, nesta zona se constatou a necessidade de atualizar constantemente a função scene e o orbit controls de maneira constante, procedeu-se à criação de uma função especifica para essa atualização, e

assim, garantir que esta seria feita, com menos linhas e melhor compreensão na sua estrutura global.



A “index.html” foi adicionado um explicativo instrutivo e sobre os controlos para a sua utilização, como também uma zona onde se fixariam as coordenadas das bolas selecionadas. Este ponto, teve a sua dificuldade, mas, após consulta sobre o modo de implementar algo dinâmico entre a posição do rato e front-end, com recurso a “document.creatElement(‘div’)”, a sua implementação ficou acessível, trantando-se apenas de criar uma nova manipulação entre os diferentes elementos, para quando existe bola selecionada e para quando não existisse bola selecionada.

Para efeitos de clareza sobre o modo de interagir com a interface, foi acrescentado um texto, sobre o modo de funcionamento do projeto e interação com o tabuleiro e bolas.
Para fazer funcionar o projeto, utilizou-se como servidor o aconselhado no fórum da disciplina (‘Web Server for Chrome’).
O e-fólio encontra-se em modo partilha no github, a partir do dia seguinte à data limite de entrega, em repositório do estudante, para facilitar acesso ao trabalho realizado mesmo em caso de dificuldades de abertura deste documento partilhado em modo “Carvalho2000563.zip”.