Impossible Game 3D

Diogo Daniel Soares Ferreira Nº 76504
Luís Davide Jesus Leira Nº 76514
Turma prática P2 – Computação Visual

Resumo – **O presente relatório apresenta a descrição detalhada relativamente a um trabalho realizado no âmbito da unidade curricular de Computação Visual. O resultado é um jogo de plataformas apelidado de "Impossible Game 3D", tendo como base a programação em WebGL.**

O relatório começa por introduzir os objetivos do trabalho em questão e a apresentação da ideia do jogo e a sua inspiração. De seguida, são apresentados os módulos que constituem o projeto e aquilo que fazem. As funcionalidades são depois descritas com o objetivo de familiarizar o leitor com o jogo implementado. Depois descrevem-se detalhadamente a implementação das funcionalidades do jogo. Finalmente, são apresentadas observações e limitações técnicas verificadas.

I. INTRODUÇÃO

Neste relatório iremos explicar as principais funcionalidades do projeto *Impossible Game 3D*. O projeto foi desenvolvido no âmbito da cadeira de Computação Visual do 4º ano do Mestrado Integrado em Engenharia de Computadores e Telemática da Universidade de Aveiro.

A ideia inicial para este projeto teve como base alguns jogos já existentes. O jogo *Impossible Game*[1], lançado em 2009, foi o primeiro jogo deste género a ser lançado. Consiste num labirinto com cubos, onde o utilizador necessita de se desviar deles, e saltar por cima deles, para chegar ao fim do labirinto. O jogo *Geometry Dash*[3], apenas disponível para *smartphones*, acrescenta funcionalidades ao jogo, como a alteração da gravidade nos labirintos, línguas de fogo, foguetões e outras novidades. Finalmente, o jogo *Cube Runner (Figura 1)*, que funciona em 3D e apresenta mais novidades nos labirintos e efeitos especiais.

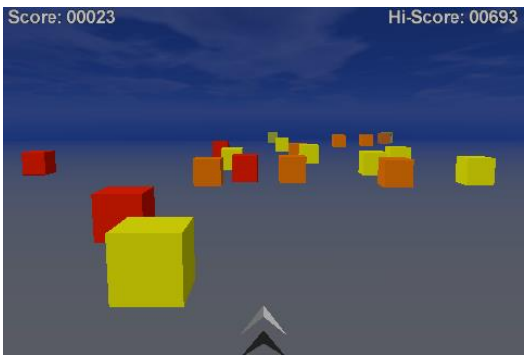

Figura 1 - Imagem retirada do jogo Cube Runner, uma das inspirações para o Impossible Game 3D.

O objetivo para este projeto seria simular um jogo similar aos apresentados anteriormente. O utilizador poderia controlar um cubo e escolher um labirinto para jogar, tal como criar novos labirintos. Cada labirinto seria composto por um conjunto de cubos designados por obstáculos, os quais o cubo do utilizador não poderia embater de frente nem de lado, mas poderia manter-se no topo. Para além destes obstáculos fixos, existiriam mais obstáculos a cair do céu, distribuídos aleatoriamente, não podendo colidir de qualquer forma contra eles. A pontuação seria com base nas moedas que dão pontos que o utilizador teria que apanhar e evitar a todo o custo apanhar moedas que retiram pontuação.

II. DESCRIÇÃO BREVE DO CONTEÚDO DOS FICHEIROS

O projeto apresenta um conjunto de ficheiros necessários à execução do jogo. Nesta secção apresentamos uma breve descrição de cada ficheiro:

- **Impossible-game.html** – Este ficheiro contém o código *html* necessário para apresentar o *viewport* e os botões num browser. Também contém o *fragment* e *vertex shader*, que serão enviados para a placa gráfica do computador.
- **impossible-game.js** – Este ficheiro contém a lógica de alto nível do jogo. É aqui que são definidas as variáveis globais a serem usadas por outros ficheiros, e também são definidas as principais funções de alto nível do jogo (alterações na cena, geração de obstáculos, moedas e zonas de aceleração, criação de labirintos, aplicação de transformações globais, …).
- **setUpBuffers.js** - Neste ficheiro são criados e alterados os *buffers* para os modelos utilizados.
- **MouseKeyEvents.js –** Neste ficheiro são tratadas as interações do utilizador com o jogo através do teclado, assim como o movimento do rato para alteração da visão do cubo no *viewport*.
- **game_physics.js** – Este ficheiro contém os algoritmos necessários para alterar a posição do cubo e detetar saltos e colisões com outros objetos.
- **drawModels.js –** Este ficheiro contém as funções necessárias para desenhar os modelos na cena, com a respetiva iluminação, textura e transformações ao modelo.
- **geoFigures.js** – Neste ficheiro estão armazenadas as características de cada modelo: os vértices, as normais aos vértices, o índice de vértices, as coordenadas das texturas utilizadas e as características do material utilizado (ou a cor, quando o material não é iluminado).

- **maths.js, models.js, webgl-utils.js e initshaders.js –** Estes ficheiros foram fornecidos aquando das aulas práticas e contêm funções auxiliares para (i) cálculos com matrizes, (ii) criação de malhas triangulares (iii) interoperabilidade entre browsers e (iv) inicialização e passagem de argumentos aos *shaders*, respetivamente.
- **lightSources.js** – Este ficheiro, também fornecido nas aulas práticas, define os focos de luz a serem utilizados na cena.
- **maze1, maze2, maze3, maze4, maze5** – Estes ficheiros contêm a localização dos cubos que formam o labirinto (mais informações sobre o formato dos ficheiros na sub-secção IV-F. Adicionar novos labirintos).

III. FUNCIONALIDADES

Nesta secção iremos explorar o modo de funcionamento do jogo, bem como detalhar todas as funcionalidades relevantes.

A. *Início do jogo*

Quando a página *web* é aberta com o jogo, o utilizador tem acesso, do lado esquerdo, a um conjunto de botões que pode utilizar para alterar inicialmente a sua execução. No topo, existe um botão para carregar um novo labirinto. O labirinto necessita de estar no formato especificado (ver sub-secção F) para ser válido. Se nenhum labirinto for carregado, o jogo não iniciará.

Em baixo desse botão também existem botões para alterar a velocidade do jogo. Inicialmente a velocidade do jogo encontra-se em *Medium*. Também é possível alterar a velocidade do jogo utilizando os botões do teclado "l", "m" e "h", para *Low*, *Medium* e *High* respetivamente. Se o jogo estiver a decorrer, a velocidade só será alterada no início do jogo seguinte. Existem dois botões para controlar o início e o final do jogo. Também poderão ser utilizadas as teclas "s" e "r", que correspondem a *Start game* e *Restart game*, respetivamente. Por último, é apresentada a pontuação atual e a melhor pontuação obtida até ao momento

B. *Funcionamento do jogo*

No *viewport* apresentado é possível alterar a posição do observador em relação ao cubo. Quando o utilizador carrega com o botão esquerdo do rato no *viewport*, a posição do observador é alterada, sendo possível alterar a vista do cubo (é possível visualizar o cubo do lado esquerdo, direito, de frente e de topo). Com o *scroll* do rato é ainda possível alterar a coordenada Z do utilizador em relação ao cubo, alterando assim a sua distância ao cubo.

Após o labirinto ser carregado, irão surgir obstáculos fixos, sendo estes um conjunto de cubos com um aspeto de blocos de tijolo que formam o labirinto. O utilizador poderá movimentar o seu cubo para os lados (esquerda ou direita) e saltar, tudo através das setas. O seu cubo não deverá ir contra esses cubos, sob o risco de perder o jogo. No entanto, poderá saltar por cima deles ou saltar para cima deles sem prejuízo próprio. Surgirão também, aleatoriamente, obstáculos (cubos com um aspeto de blocos de pedra) que caem progressivamente, tendo o utilizador que se desviar deles para não perder o jogo. O labirinto não terá fim, pois será repetido pelo algoritmo. A velocidade do jogo também aumenta progressivamente à medida do tempo com base no tamanho do labirinto, causando maior dificuldade em ultrapassar os obstáculos.

Irão surgir dois tipos diferentes de moedas. As moedas que refletem a cor amarela são moedas que o cubo do utilizador deve apanhar para aumentar a sua pontuação. Cada moeda vale 10 pontos. No entanto, existem pontos extra, dependendo da velocidade inicial do cubo (0 pontos extra para velocidade inicial baixa, 2 pontos extra para velocidade inicial média, 4 pontos extra para velocidade inicial alta). As moedas que refletem a cor roxa são moedas que o cubo do utilizador deve evitar apanhar. Essas moedas subtraem 10 pontos à pontuação atual, subtraindo ainda um extra de 2 pontos para velocidade inicial média e 4 pontos para velocidade inicial alta.

Ao longo do percurso do labirinto, irão também aparecer quadrados de cor azul no piso. Esses quadrados são zonas de aceleração, que fazem com que o jogo seja acelerado momentaneamente, isto é, enquanto o cubo do utilizador estiver por cima dos mesmos.

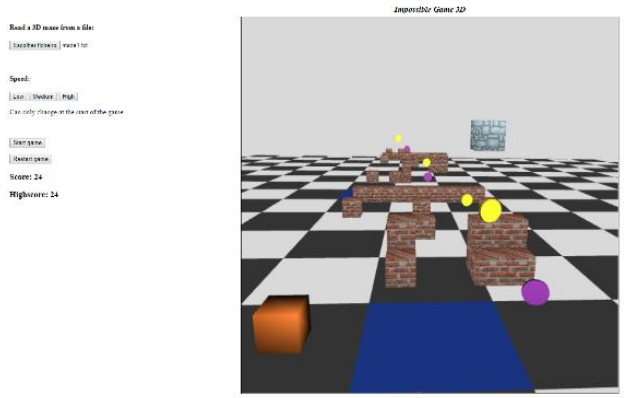

Figura 1 - Imagem de um jogo de Impossible Game 3D. O cubo que o utilizador controla é o cubo cor-de-laranja. A zona azul é a zona de aceleração. Existe moedas benéficas e venenosas no labirinto, bem como um obstáculo a cair no labirinto.

IV. DETALHES DE IMPLEMENTAÇÃO

Nesta secção apresentamos alguns detalhes importantes da implementação efetuada.

A. *Iluminação*

A iluminação do jogo é calculada no *Vertex Shader*, para otimização do jogo. Assim sendo, é passado ao *Vertex Shader* uma lista com todas as fontes de iluminação presentes para cada fragmento. É possível sinalizar ao *Fragment Shader* se o fragmento em questão tem uma cor fixa ou se é iluminado por alguma fonte de luz (através da variável *colortype*). No *Fragment Shader* é possível passar como argumento um coeficiente de textura (valor entre 0 e 1, para a variável *vTextureCoef*), para definir a razão entre a iluminação nesse vértice e a aplicação de uma textura. São aplicadas duas fontes de iluminação ao modelo, uma de cada lado do labirinto (coordenadas (-1, 1, 2.5) e (1,1,2.5)).

B. *Repetição de labirintos e geração de moedas e zonas de aceleração*

Quando é adicionado um labirinto, o algoritmo verifica se o seu tamanho preenche a cena atual (até ao plano *far*). Se isso não se verificar, o código automaticamente repete o labirinto até preencher toda a cena visível para o utilizador. À medida que o cubo do utilizador avança e a cena global recua (na verdade, o cubo recua no eixo ZZ e o labirinto aumenta a sua o seu valor no eixo ZZ, mas a perceção do utilizador é a contrária), o algoritmo verifica se o final do labirinto preenche todo o campo de visão do utilizador (até ao plano *far*). Se isso não acontecer, o labirinto é repetido, e são gerados novos elementos no labirinto, colocados pseudo-aleatoriamente (moedas, obstáculos a cair do céu, zonas de aceleração no piso).

C. *Modelos utilizados na cena*

A cena desenhada no *viewport* apresenta originalmente apenas dois modelos diferentes (cubo e círculo), que são posteriormente replicados com diferentes características, transações, rotações e outros atributos. Os modelos estão representados utilizando a lista de vértices e uma lista com os índices dos vértices conectados. Existe também uma lista de normais aos vértices para facilitar o cálculo da iluminação, que utiliza o método de *gouraud*.

Os cubos do labirinto e os cubos que caem no o labirinto apresentam duas texturas diferentes, ambas alojadas e carregadas a partir do website *imgur*[4], com uma conta criada pelos criadores do projeto. O objetivo é permitir ao browser *Google Chrome* executar o jogo com as texturas (o jogo está otimizado para o browser *Google Chrome*, sendo que noutro browser poderá ser impossível jogar, para mais informações ler a secção V. Observações e limitações técnicas) sem ser necessário executar nenhum servidor local.

D. *Física*

A física do jogo é o que faz com que este se torne o mais realista e jogável possível. É onde são calculados os movimentos de todos os modelos e todas as condições que definem o modo como este se processa.

Após a inserção de um labirinto, o movimento do cubo principal é calculado constantemente, sendo realizada uma translação no eixo ZZ com base na velocidade do jogo. Se o cubo se encontrar numa zona de aceleração, a velocidade do jogo aumenta enquanto o cubo se encontra a passar por tal zona. Este, para além de poder movimentar-se para os lados, pode também saltar. O salto é feito tendo em conta uma variável que se designa de gravidade, simulando assim um salto mais realista, sofrendo também uma rotação de 90º durante o mesmo.

É também através deste módulo que se verifica a colisão entre o cubo do utilizador e os restantes modelos do jogo (cubos que servem de obstáculos e as moedas que atribuem pontuação). Para cada cubo com coordenada perto da coordenada Z do cubo principal, é verificado se existem cubos na mesma posição que o cubo principal (verificar se coordenadas x, y e z dos centros dos cubos são iguais ou diferem menos do que a distância de uma aresta) e, caso existam, faz-se a verificação se o mesmo colidiu contra algum deles (de frente, de lado ou de baixo) ou se saltou para cima dele (caso a velocidade em Y do cubo seja negativa e o cubo esteja com o seu centro acima do outro cubo). No caso de colisão, o jogo é terminado. No caso de salto para cima de outro cubo, a coordenada Y do outro cubo é mantida em cima do outro cubo até existir um salto (velocidade em Y ficará positivo) ou o cubo do utilizador deixe de estar em cima do cubo (velocidade em Y ficará negativa). Depois verifica-se se o cubo colidiu com alguma moeda, ou seja, se a apanhou, de modo semelhante à verificação da colisão entre cubos. Desta forma, o utilizador ganhará ou perderá pontos consoante as moedas que apanhou (ou manter a pontuação no caso de não ter apanhado nenhuma nesse movimento). Verifica-se também se o cubo principal colidiu com algum cubo que se encontra em queda (designado por obstáculo voador) de modo semelhante às verificações anteriores com os obstáculos fixos e moedas.

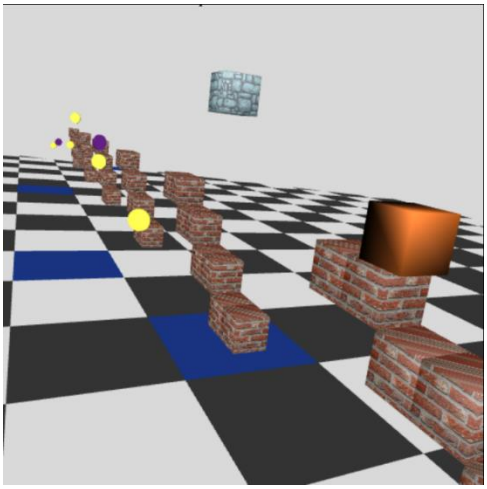

Figura 2 – Imagem de um jogo de Impossible Game 3D. O cubo do utilizador está atualmente em cima de outros cubos. O módulo de física permite que o cubo suba para cima de outros cubos, mesmo que estes não estejam apoiados diretamente no solo. Também pode ser visualizado a repetição do labirinto ao longo da cena.

E. Otimização de arrays

Durante a implementação do jogo, foi notado inicialmente que quando o jogo se estendia por algum tempo, começavam a surgir alguns atrasos que causavam perturbações na jogabilidade. Após alguma investigação sobre o problema, concluímos que se devia não ao cálculo da iluminação, mas aos cálculos efetuados no módulo de física. Porque existiam demasiados cubos, moedas e zonas de aceleração, o módulo de física testava as colisões com todos os modelos, o que atrasava a execução do jogo.

Para melhorar a questão, os *arrays* gerados pelo algoritmo são ordenados por valor da coordenada Z. Assim sendo, quando a sua coordenada Z ultrapassa a cena vista pelo utilizador, eles são removidos do *array*. Para além disso, no módulo de física, as comparações só são efetuadas para cubos que estão em coordenadas Z perto do cubo do utilizador.

F. Adicionar novos labirintos

A construção de novos labirintos traduz-se pela construção de um ficheiro que contém as coordenadas dos cubos que servem como obstáculos fixos do percurso. O ficheiro deve conter inicialmente o número de cubos que o labirinto contém, seguido das coordenadas x, y e z do centro dos cubos, separadas por espaços. É importante que se tome nota de alguns pormenores importantes na geração dos cubos. Para o cubo se situar no labirinto, a coordenada x necessita de ter um valor entre -1 e 1. O valor da coordenada Y para um cubo se situar no solo é -0.5. Qualquer coordenada Y acima do previsto causará que o cubo se encontre acima no nível do cubo do utilizador. O utilizador inicialmente encontra-se na coordenada Z com o valor 2.5 em relação ao labirinto. Isso significa que qualquer valor para a coordenada Z superior a 2.5 fará com que o cube fique fora do campo de visão do utilizador. Finalmente, o cubo tem arestas de tamanho 0.5.

Relativamente aos cubos que caem, tal como as moedas e as zonas de aceleração, são gerados pseudo-aleatoriamente com base no labirinto inserido. Como tal, assumem posições diferentes em cada jogo do utilizador, não sendo possível definir previamente a sua localização.

Sempre que o final do labirinto não ultrapassa o plano *far*, o algoritmo repete o labirinto até a premissa não se verificar. Portanto, se o labirinto for demasiado pequeno, o utilizador perceberá que está a jogar sempre no mesmo labirinto. No entanto, a dificuldade aumentará, pois a velocidade irá aumentar com o progresso do jogo.

G. Rotações e translações da cena através de interação do utilizador com o rato

Quando o utilizador carrega com o botão esquerdo numa zona do *viewport*, a cena sofre transformações conforme a posição do rato. Se a posição do rato for do lado esquerdo do *viewport*, o utilizador poderá visualizar o labirinto segundo uma rotação de 90º para a direita. Quando o utilizador desloca o rato para o lado direito, a rotação inversa é efetuada, sendo que no máximo é feita uma rotação de 90º para a esquerda.

Acontece um processo similar para as coordenadas Y do rato. Se o utilizador pressionar o botão esquerdo do rato e deslocar o rato até ao topo do *viewport*, é possível ver o labirinto "de cima", ou seja, foi aplicada uma rotação de 90º para baixo. O inverso também pode ser visto quando o utilizador desloca o rato para a parte inferior do *viewport*, onde não é efetuada nenhuma rotação ao labirinto.

A par com as rotações, também são efetuadas translações nos três eixos XX, YY e ZZ. O objetivo é formar um "círculo" à volta do cubo do utilizador, onde podem ser aplicadas rotações à cena, mas o utilizador continua a ver o seu cubo e o labirinto à frente. No entanto, também é possível alterar de modo a ver apenas o labirinto à frente do cubo ou toda a cena de uma posição mais afastada. Para tal, pode ser efetuado um *scroll* com o rato, a toda a cena irá deslocar-se no eixo ZZ, de acordo com o sentido do *scroll* efetuado.

V. OBSERVAÇÕES E LIMITAÇÕES TÉCNICAS

Nesta secção iremos expor algumas observações sobre a implementação e jogabilidade do jogo, bem como limitações técnicas.

O jogo está implementado para ser jogável no browser *Google Chrome*. Noutros browsers como *Firefox* ou *Edge* é possível que o jogo fique extremamente difícil, ou mesmo impossível de proporcionar uma experiência agradável ao utilizador. Tal é devido ao atraso na execução do código *Javascript* de outros browsers. Especificamente, o salto do cubo para cima de outro cubo é calculado através da sua coordenada Y em relação à coordenada homónima de outro cubo. Como noutros browsers existe um atraso maior

considerável, na maior parte das vezes o algoritmo de cálculo da física dos cubos apenas consegue calcular o embate do cubo quando ele já se encontra com a tcoordenada Y para lá do centro do outro cubo, o que causa uma colisão e a consequente perda do jogo.

As imagens utilizadas nas texturas estão armazenadas no website *Imgur*[4]. Tal deve-se à política restrita de controlo de objetos locais do *Google Chrome*, que não permite que seja carregado um objeto local para o jogo. Neste caso, o *Imgur*[4] é um website onde as imagens são de domínio público, e que permite pedidos *Cross-Request*, não sendo necessário preparar nenhum servidor local para carregar as imagens para o browser. A desvantagem é que o jogo não irá carregar as texturas se o utilizador não se encontrar *online*.

No *Google Chrome* existe um aviso (*Warning*) que é enviado para a consola, que refere a inexistência de texturas em alguns modelos. Noutros browsers esse aviso não aparece, e não influencia em nada a jogabilidade nem a segurança do jogo, pois os modelos sem texturas são-no propositadamente, sendo que possuem iluminação ou uma cor fixa.

Para a construção do modelo das moedas, inicialmente foi pensado em subdividir o cubo em malhas triangulares e expandir essas malhas, de acordo com normais ao centro do cubo. No entanto, nesse caso, as normais eram calculadas em relação a cada malha, e não aos vértices, como é efetuado pelo método de *gouraud*. Portanto, a opção foi utilizar um *software* de modelação para construir as moedas, de acordo com a sua forma cilíndrica.

Como foi utilizado um *índex* para definir as faces de cada modelo, os *buffers* internos de *WebGL* necessitam de possuir o tamanho certo, para serem enviados para os *shaders* (buffers de coordenadas dos vértices, das normais aos vértices, das coordenadas das texturas em cada face). No entanto, as moedas não apresentam nenhuma textura, apenas características do material. Como a distinção para utilizar texturas apenas é efetuada dentro dos *shaders*, é necessário criar um *array* com coordenadas "falsas" de texturas que nunca serão utilizadas, apenas para o código não gerar nenhum erro.

VI. CONTRIBUIÇÃO DE CADA MEMBRO DO GRUPO E IDENTIFICAÇÃO DO TRABALHO EFETUADO

A contribuição de cada membro do grupo, em percentagem, foi a seguinte:

- Diogo Daniel Soares Ferreira – 55%
- Luís Davide Jesus Leira – 45%

Os módulos MouseKeyEvents.js e SetUpBuffers.js foram escritos por Diogo Ferreira. A maior parte do módulo physics.js também foi escrita por Diogo Ferreira, com contribuições de Luís Leira. O ficheiro geoFigures.js foi maioritariamente escrito por Luís Leira, com contribuições de Diogo Ferreira. O resto dos ficheiros foram escritos por ambos os alunos, com participação repartida igualmente.

VII. CÓDIGO EXTERNO UTILIZADO

A construção deste jogo teve por base conhecimento adquirido nas aulas, tendo sido utilizado código fornecido através dos guiões práticos das aulas da unidade curricular em que este trabalho se insere.

Os ficheiros maths.js, models.js e webgl-utils (descritos em II. Descrição breve do conteúdo dos ficheiros) são que foram integralmente fornecidos nas aulas práticas.

Os ficheiros lightSources.js, Impossible-game.html, impossible-game.js e setUpBuffers.js contêm código parcialmente utilizado nas aulas, embora o conteúdo dos ficheiros esteja profundamente diferente do fornecido.

No ficheiro MouseKeyEvents.js existem duas pequenas funções retiradas de um URL externo[5] (funções getRelativeMousePosition e getNoPaddingNoBorderCanvasRelativeMousePosition) para obter a posição do rato na cena.

O restante código do projeto pertence aos autores.

VIII. CONCLUSÕES

A realização deste trabalho permitiu integrar diversos conhecimentos aplicados nas aulas, tal como a necessidade de exploração dos mesmos e integração de diversas componentes que permitem a interação e um nível bastante razoável em termos de jogabilidade do Impossible Game 3D.

A nível de conteúdos presentes no projeto, quase todos os que foram lecionados estão contidos no jogo: Desenho de modelos 3D, animação de modelos 3D, transformações globais e locais, projeção perspetiva, método de *gouraud* para iluminação calculada na GPU, varias fontes de iluminação, interação com o utilizador através do rato e do teclado, estrutura de dados que representa os vértices apenas uma vez e texturas aplicadas a alguns modelos. Não foram aplicadas transparências porque se concluiu que apenas prejudicariam o utilizador no jogo.

Achamos que atingimos o objetivo a que nos propusemos no início, que foi o de construir um jogo agradável para os utilizadores, com diversas funcionalidades que permitem que cada jogo efetuado seja interessante e diferente do anterior, utilizando as ferramentas aprendidas nas aulas de Computação Visual.

Durante o decorrer do projeto, houve algumas dificuldades que nos foram apresentadas que complicaram o desenvolvimento do projeto. A necessidade de desenhar dois modelos diferentes na cena foi um desafio, principalmente porque um dos modelos (a moeda) continha faces circulares que necessitavam de ser iluminadas da forma correta. Para resolver o problema de forma eficiente, foram criadas duas funções: *SetUpCube* e *SetUpCoin*, que alteram o conteúdo dos *buffers* para desenhar os cubos e as moedas, respetivamente. Para além disso, utilizámos software externo (*Blender*) para criar o modelo das moedas.

Outra dificuldade sentida foi a criação do módulo de física de raiz, responsável por calcular os saltos e detetar colisões entre o cubo e os obstáculos. Devido a limitações no

browser, vimo-nos obrigados a transitar do *Firefox* para o *Google Chrome*, devido à rapidez de execução do código.

Como trabalho futuro, seria interessante acrescentar mais funcionalidades ao jogo, como acrescentar "buracos" no solo do labirinto, onde o cubo não poderia pisar, ou a inversão da gravidade em alguns locais, onde o labirinto ficaria ao contrário (o piso ficaria na parte de cima, e o cubo saltaria para baixo), ou até a possibilidade do cubo do utilizador disparar para destruir alguns objetos, como cubos voadores. Também seria interessante adicionar um conjunto grande e diferente de labirintos, e organizá-los por níveis de dificuldade, para permitir ao utilizador ter um objetivo maior, para além de obter o número máximo de pontos num labirinto.

REFERENCES

[1] Cube Runner - http://store.steampowered.com/app/567280/Cube_Runner/

[2] Impossible Game - http://impossiblegame.org/

[3] Geometry Dash - http://store.steampowered.com/app/322170/Geometry_Dash/

[4] Imgur - https://imgur.com/

[5] Stackoverflow URL - https://stackoverflow.com/questions/42309715/how-to-correctly-pass-mouse-coordinates-to-webgl