Crossy Roads Endless Runner in WebGL

Dário Matos, 89288 Pedro Almeida, 89205

Resumo **–Este documento relata o processo criativo da aplicação Crossy Roads, criada no âmbito da cadeira de Computação Visual. Através da tecnologia WebGL, este projeto retrata um videojogo já existente, com o mesmo nome, que visa simular a travessia sem fim de uma personagem principal, a galinha, contornando obstáculos de forma sucessiva. O presente relatório começa por descrever a interação do utilizador com o projeto, seguida do desenvolvimento do jogo, onde apresenta os métodos e algoritmos utilizados de forma a alcançar mecânicas semelhantes ao videojogo, os modelos e a implementação dos mesmos.**

Abstract **– The following report presents the development of an application similar to the Android/iOS videogame Crossy Roads, regarding the Visual Computation course. The app allows one to move as a main character, throughout an infinite map, avoiding obstacles, aiming to reach successively bigger distances. All of the features where developed using the WebGL technology. This document structures the application starting with a small introduction, followed by the user interaction. Lastly, the game development, in order to describe how the video game mechanics were replicated: not only the algorithms in use but also the 3D models that are seen.**

I. Introdução

O projeto foi desenvolvido no âmbito da unidade curricular de Computação Visual. O objetico do projeto é implementar o jogo *Crossy Roads*, explorando a tecnologia *WebGL API*. No jogo original, uma personagem principal (*Chicken*) com uma capacidade de se movimentar em 4 direções, tem como objetivo atravessar estradas e rios, sem colidir com obstáculos, tentando alcançar a maior distância ao ponto de início. Como será explicado mais à frente no documento, foram feitas algumas simplificações ao jogo original. Contudo, foram também implementadas novas funcionalidades.

Ao iniciar o jogo, este encontra-se em *Pausa*. Para dar início ao movimento é necessário carregar na letra *P* (Play/Pause). Isto acontece para ser possível escolher o modo de jogo e a velocidade dos inimigos antes de se começar. Imediatamente abaixo da janela do jogo, é possível verificar o nível e os pontos do jogo a decorrer. É ainda possível carregar um ficheiro externo *OBJ* para mudar a personagem principal e verificar os *FPS* (frames per second).

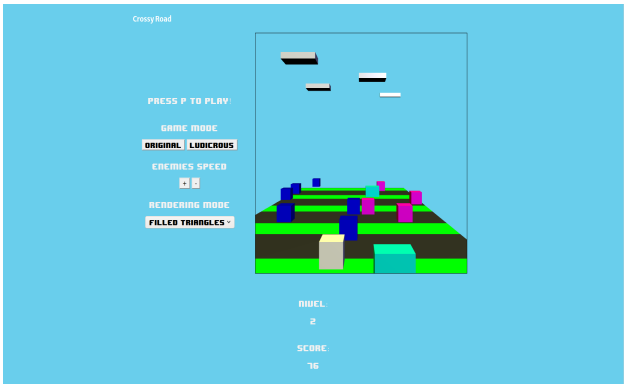


Fig. 1: Página do jogo

II. Interação com o Utilizador

A. Instruções

As instruções do jogo encontram-se abaixo da janela do jogo.

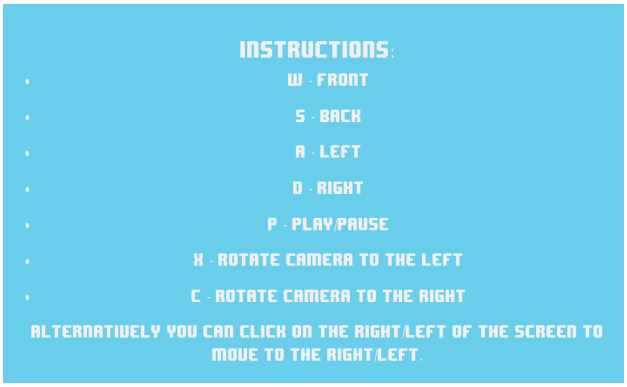


Fig. 2: Instruções

B. Modos de jogo

Foram desenvolvidos dois modos de jogo: *Original* e *Ludicrous*. Em ambos os modos é possível configurar a velocidade dos obstáculos através dos botões '+' e '-' e selecionar o modo de renderização entre Filled Triangles, Wireframe e Vértices. O objetico é o mesmo, chegar o mais longe possível ao ponto de início sem chocar contra nenhum obstáculo, tendo apenas condições de travessia diferentes.

B.1 Original

O modo original é uma réplica do jogo original, ou seja, o jogador controla todos os movimentos da Chicken.

B.2 Ludicrous

Neste modo de jogo, a galinha avança sozinha pelo mapa, sendo que o jogador apenas tem de evitar colisões com os obstáculos. Este modo foi desenvolvido para aumentar o nível de dificuldade, já que retira a possibilidade de parar num lugar seguro do mapa (relva). Neste modo de jogo é possível jogar apenas com o rato, clicando no lado da janela do jogo para o qual se pretende mover a Chicken, como foi explicado na secção Instruções (II-A).

III. Desenvolvimento do Jogo

Neste capítulo serão explicado as etapas do desenvolvimento do projeto.

A. Código base

Ao iniciar o desenvolvimento do jogo, foi decidido reaproveitar o código das aulas práticas. Assim, foi escolhido o exemplo da aula 8 *Example_NEW*. Este exemplo tinha já o ficheiro *sceneModels.js* que permitiu criar e mais tarde manipular todos os elementos do jogo.

B. Personagem principal

O segundo passo no desenvolvimento do jogo foi a criação da personagem principal (Chicken). Neste momento, a Chicken foi representada com um cubo, apenas por motivos de simplificação.

Contudo, foi acrescentado a opção de carregar um ficheiro externo OBJ como forma de representar a Chicken.

Fig. 3: Personagem Principal

C. Movimentos da Chicken

O passo seguinte foi implementar a capacidade de movimentar a Chicken. Inicialmente a Chicken podia mover-se para cima-baixo-direita-esquerda com as teclas, respetivamente, w-s-d-a.

D. Estrada

Seguidamente, foi necessário pensar numa forma de representar a chão onde a Chicken iria andar. A solução encontrada para representar a estrada foi através de um *simpleCubeModel* e tirando partido dos atributos de escala *sx*, *sy* e *sz*, esticá-lo de forma extensiva no eixo ZZ, achatá-lo praticamente todo no eixo YY e esticá-lo ligeiramente no eixo XX.

```
274
275  // chao
276  sceneModels.push( new simpleCubeModel() );
277
278  sceneModels[3].tx = 0.0; sceneModels[3].ty = -0.3; sceneModels[3].tz = -13;
279
280  sceneModels[3].sx = 3;
281  sceneModels[3].sy = 0.001;
282  sceneModels[3].sz = 13;
283  sceneModels[3].kDiff = [0.18,0.18,0.18];
284  sceneModels[3].type = "Road";
285
286
```

Fig. 4: Criação do elemento Estrada

Foi neste momento que foi alterada a posição do utilizador para ser semelhante ao jogo original, para isso, a "câmara" foi deslocada para cima e ligeiramente para a direta, utlizando a matriz de transformação global para o efeito.

A relva foi implementada com o mesmo processo que a estrada. A caracteristica da relva é que é um lugar seguro no mapa, ou seja, o tráfego nunca passa por cima da relva, apenas na estrada.

```
sceneModels.push( new simpleCubeModel() );

sceneModels[4].tx = relvatx; sceneModels[4].ty
     = relvaty; sceneModels[4].tz = -0.5;

sceneModels[4].sx = relvasx;
sceneModels[4].sy = relvasy;
sceneModels[4].sz = relvasz;
sceneModels[4].type = "Grass";
```

E. Movimento do Mapa

Como explicado atrás (III-C), a Chicken possuia inicialmente a capacidade de se movimentar para cima-baixo-direita-esquerda. Contudo, a partir do momento em que foi implementada a estrada, esta capacidade deixa de fazer sentido. Assim, a Chicken tem apenas de andar no sentido frente-trás, ou seja, tem de se deslocar no eixo ZZ e não no eixo YY.

Como característica, foi decidido que a galinha se pode deslocar no eixo dos ZZ, não sendo possível ultrapassar o limite do visível do canvas. Também ao longo do comprimento da estrada, no eixo dos XX, existe um limite para que a personagem não deixe de ser visível. A solução encontrada para uma implementação mais simples dos movimentos em ZZ é a de movimentar o mapa e todos os outros elementos para trás/frente. Assim, é possível dar a sensação de deslocação da Chicken, poupando-se translações de mais objetos.

F. Tráfego

Depois de termos uma personagem principal capaz de se movimentar pelo mapa, era necessário implementar a simulação de tráfego. À semelhança da Chicken, os carros foram representados como um cubo por razões de simplificação de modelação.

A diferença destes elementos para todos os outros até ao momento criados é que estes deslocam-se sozinhos, isto é, são animados, deslocando-se uma certa distância no eixo XX a cada *tick*.

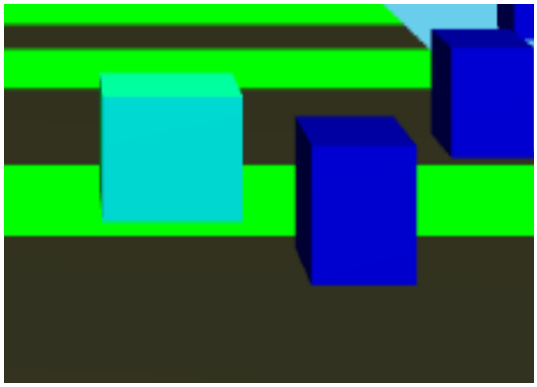


Fig. 5: Objetos 3D de simulação de obstáculos

F.1 Sentido

Foram implementados dois tipos destes obstáculos, distintos apenas no sentido da deslocação ao londo do eixo XX. Os que se deslocavam no sentido esquerda - direita eram gerados obrigatóriamente do lado esquerdo e vice-versa. Esta dinâmica tem como objetivo aumentar a dificuldade do jogo, passando a haver mais obstáculos em simultâneo e movimento dos mesmos, ainda que previsível.

Numa primeira fase, estes elementos apenas atravessavam o mapa uma única vez. Mais tarde, isto foi mudado devido ao facto de o jogador poder apenas parar num lugar seguro (por cima da relva) durante alguns segundos para que todos os obstáculos tivessem sido evitados. A solução encontrada para este problema foi fazer com que estes obstáculos efetuassem a travessia da estrada em sentido oposto a partir do momento em que atingem um extremo do objeto que percorrem.

F.2 Posição

A posição onde estes elementos são gerados foi cautelosamente pensada. Foi necessário calcular os intervalos de posições ao longo do eixo ZZ para que não se encontrassem em zonas seguras (relva) nem imediatamente em frente à personagem principal, aquando do início do jogo.

Em baixo está explícito o código de uma função auxiliar criada para a geração dos obstáculos.

```
function addEnemy1(z_value){
  //obstaculo
  var a = sceneModels.length
  sceneModels.push( new simpleCubeModel() );
  sceneModels[a].tx = getRandomArbitrary(0,3);
       sceneModels[a].ty = 0.0; sceneModels[a
      ].tz = z_value;
  console.log(sceneModels[a].tx)

  sceneModels[a].sx = sceneModels[a].sy =
      sceneModels[a].sz = 0.2;
  sceneModels[a].kDiff = [1,0,1];
  sceneModels[a].type = "Enemy1";
  sceneModels[a].dirXX = "-1";
}
```

A função recebe um argumento *z_value* que é o valor onde o elemento deve ser criado no eixo do ZZ. Esse valor deverá estar num destes intervalos: (-0.9,-1.4, -1.8), (-3.4, -4.3), (-5.9, -6.8), (-8.4, -9.3), (-10.9, -11.8).

Como pode ser confirmado ainda no código da função criada, o valor do eixo XX é aleatório dentro de um certo limite. Isto acontece para que sempre que se joga se tenha uma experiência diferente.

Podemos concluir que a criação dos obstáculos tem uma valor "fixo" no eixo do ZZ, e um valor variável no eixo do XX.

F.3 Deteção de choque

Finalizando a criação do tráfego, foi necessário implementar a funcionalidade de deteção do choque da Chicken com os elementos obstáculo.

Como se pode confirmar no código abaixo, a deteção do choque da Chicken com outros elementos consiste em verificar se a diferença da posição no eixo do XX, YY e ZZ da Chicken com os restantes elementos é menor que um certo intervalo. Estes valores podem ter de ser atualizados ao carregar um ficheiro OBJ para a representação da personagem principal.

```
function colision(){
  for(i = 0; i < sceneModels.length; i++){
    if(sceneModels[i].type == "Enemy1" ||
        sceneModels[i].type == "Enemy2"){
      if ( Math.abs(sceneModels[getChicken()].
          tx - sceneModels[i].tx) <= 0.4 &&
          Math.abs(sceneModels[getChicken()].
          ty - sceneModels[i].ty) <= 0.4 &&
          Math.abs(sceneModels[getChicken()].
          tz - sceneModels[i].tz) <= 0.4){
        return 1;
      }
    }
  }
  return 0;
}
```

Contudo, é importante referir que a "aproximação" dos obstáculos ao utilizador, isto é, o deslocamento no sentido positivo do eixo ZZ, confere imprevisibilidade à deteção de choque, pois altera os valores de deslocamento em XX e YY, caso este exista.

G. Mapa "Infinito"

De forma semelhante ao jogo original, o mapa utilizado é de carácter infinito e imprevisível. Esta infinidade é possível através da adição e remoção dinâmica de elementos ao array de modelos *sceneModels*. Com o intuito de tornar a jogabilidade mais intuitiva e cativante, não poderiam existir quebras no mapa, pelo que a adição de novos elementos é efetuada em posições fora do alcance de visão do jogador. Esta

adição é efetuada sempre que seja detetada a falta de elementos imediatamente a seguir ao limite de visibilidade (componente do eixo ZZ inferior a -14,5).

```
sceneModels.forEach(element => { if(element.tz
     <=-14.5) add =1;});

sceneModels.forEach(element => {if(element.tz
    > 2){ var ind= sceneModels.indexOf(element
    );
      sceneModels.splice(ind,1);
  }
});
  if(add==0){
    extendMap();
  }
```

Inicialmente, apenas era feita a adição de novos elementos. Contudo, e após o teste da aplicação durante alguns minutos de jogabilidade, era notável uma sobrecarga da máquina que executava o projeto, devido à quantidade excessiva de objetos 3D, ainda que não visíveis. Desta forma, foi implementado um mecanismo de remoção de elementos, tendo como alvo todos os corpos já ultrapassados pela Chicken, ou seja, com uma componente do eixo ZZ superior a 2.

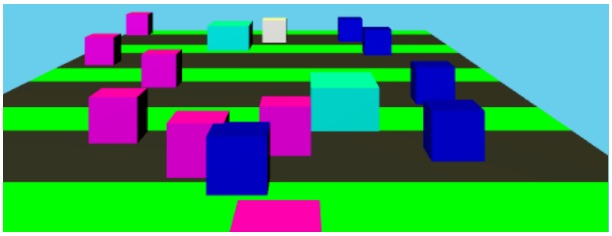


Fig. 6: Exemplo de objetos nos limites inferiores e superiores da visibilidade de ZZ, Chicken e Obstáculo Rosa, respetivamente.

H. Novos elementos

H.1 Arbustos

Neste momento tinha sido implementada um versão simplificada mas funcional do jogo. De modo a adicionar novos elementos, foi decidido criar arbustos. Este obstáculo tem a particularidade de bloquear o caminho da Chicken. Uma outra caracteristica é que estes elementos estão sempre por cima da relva. Foram, novamente, representados como um cubo como forma de simplificação na modelação.

Segue o exemplo de criação de um destes elementos.

```
sceneModels.push( new simpleCubeModel() );

sceneModels[10].tx = 1; sceneModels[9].ty =
    0.0; sceneModels[9].tz = -7.5;

sceneModels[10].sx = 0.3;
sceneModels[10].sy = 0.2;
sceneModels[10].sz = 0.2;
sceneModels[10].kDiff = [0,1,1];
sceneModels[10].type = "Block";
```

Com a criação destes elementos surgiu uma outra tarefa: detetar quando a Chicken tenta passar por um arbusto e bloquear essa ação.

```
function moveChicken(amount){

  if(!checkBlocked()[0]){
    for(i = 1; i < sceneModels.length; i++){
      if(sceneModels[i].type == "Cloud")
          sceneModels[i].tz += 0.002
      else sceneModels[i].tz += amount;
    }
    if(sceneModels[getChicken()].tz >= -10)
        sceneModels[getChicken()].tz -= amount
        ;
  }
}
```

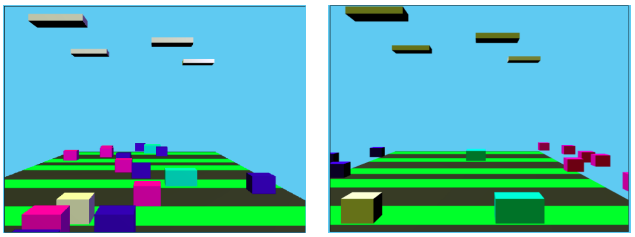
No excerto de códido imediatamente acima é exemplificada a utilização da função para verificar se a Chicken está bloqueada ou não (retorna 1 caso esteja bloqueada, 0 caso contrário). Como se pode confirmar, apenas se a Chicken não se tiveblorqueada é que se efetua o seu deslocamento. Ainda no mesmo excerto de código, pode ser confirmado que também é verificado que a Chicken é bloqueada caso a sua posição no eixo do ZZ alcance um certo valor. Isto é feito para impedir que o jogador avance demasiado no mapa e perca ou cause comportamentos inesperados por parte do jogo.

I. Nuvens

De forma a verificar o mais possível as parecenças com o jogo oficial, foram adicionados objetos representativos de núvens. Estes objetos assemelham-se aos previamente adicionados, possuindo apenas cor diferente e uma posição mais elevada no eixo dos YY, de forma a simular um céu no ambiente 3D.

J. Iluminação

Na iluminação, foram implementadas duas fontes de luz. Uma mais escura e numa posição mais baixa e uma mais clara e alta. Esta última tem a função de simular a luz do sol. Uma vez que as fontes de luz rodam, faz com que os elementos fiquem mais claros/escuros ao longo do tempo. Este efeito juntamente com as nuvens resulta na sensação de sombra proveniente da existência de nuvens, resultando numa experiência mais realista e imersiva.



(a) CLaro (b) Escuro

Fig. 7: Diferença na iluminação

Para a rotação das luzes foi aproveitado o código das aulas.

K. Rotação da Câmara

Finalmente, a última funcionalidade implementada foi a rotação da câmara. Esta funcionalidade não está

presente no jogo original mas foi decidido que seria uma funcionalidade interessante.

```
// GLOBAL TRANSFORMATION FOR THE WHOLE SCENE
mvMatrix = mult( mvMatrix, translationMatrix(
    globalTx, globalTy, globalTz) ) ;
vMatrix = mult( mvMatrix, rotationYYMatrix(
    globalAngleYY) );
```

Como referido na secção Instruções (II-A), o utilizador pode rodar a camara com as teclas x e c. De modo a adicionar dificuldade ao jogo, a partir do nível 10, a camara começa a rodar automaticamente.

L. Interface

A interface exterior ao canvas foi desenvolvida a par com a implementação de novas funcionalidades da aplicação. Inicialmente, foi colocada cor e tipos de letra correspondentes aos utilizados no jogo original *Crossy Roads*. Com o desenvolvimento do projeto e a capacidade de efetuar diversos movimentos através de mais que um método (teclado e rato), foi sentida a necessidade de indicar instruções ao utilizador. Foram também implementados alertas de fim do jogo, em situações de choque ou ultrapassagem dos limites válidos do mapa, de forma a fornecer feedback ao jogador, proporcionando melhores condições de utilização da aplicação. Infelizmente, é necessário excluir vários pop-ups de fim de jogo quando esta situação se dá, devido às diferentes velocidades de execução do código de aparecimento destes avisos e do *refresh* da página.

IV. Trabalho Futuro

A haver continuação do projeto, seria interessante modelar com mais detalhe os elementos do jogo, visto que neste momento são representados à base de cubos. Seria também mais cativante se estes elementos possuíssem texturas, representando objetos conhecidos do quotidiano, e se fosse possível obter novos ângulos de visualização do mapa, girando-o utilizando o rato. Finalmente, no âmbito de correção de bugs, um desenvolvimento posterior permitira a melhor integração da aplicação em vários browsers, construindo uma interface mais fixa, e um aperfeiçoamento do mecanismo de deteção de choques com os obstáculos.

V. Conclusões

Com este trabalho concluímos que a WebGL API é uma das ferramentas ideais para a modelação, visualização e interação de objetos em aplicações web. Determinamos também que o projeto demonstra muitas das capacidades desta API e vai de encontro aos objetivos propostos pelos professores, culminando numa plataforma de lazer e aprendizagem para os elementos do grupo.

VI. Contribuição

Ambos os elementos contribuíram de uma forma equivalente para o desenvolvimento do projeto. Assim, a contribuição de cada elemento corresponde a 50%.

VII. Demo

Link: https://dariod9.github.io/CrossyCV/

References

Exemplos das Aulas Práticas
http://sweet.ua.pt/jmadeira/WebGL/
Stackoverflow - Consultas relacionadas com HTML e desenvolvimento da interface
https://stackoverflow.com/