Knight's Tour

Pedro Alves 88861
Ruben Menino 89185

Resumo **- O presente artigo apresenta informação relativa a uma aplicação 3D interativa que permite ao utilizador observar o funcionamento de um algoritmo.**
O seguinte artigo começa por fazer uma apresentação do algoritmo Knight's Tour. De seguida, apresenta-se a ideia geral do trabalho proposto, como foi realizada a sua implementação e os resultados obtidos.

Abstract **- This paper presents an interactive 3D application that allows the user to observe the operation of an algorithm. The following paper begins by presenting the Knight's tour algorithm. Then, the general ideia of the project, the implementation and the obtained results.**

I. INTRODUÇÃO

O seguinte trabalho foi desenvolvido para a unidade curricular de Computação Visual (4º ano, 1º semestre) do Mestrado Integrado de Computadores e Telemática.

O objetivo da seguinte aplicação é demonstrar o funcionamento do algoritmo bastante conhecido, Knight's Tour, porém numa perspectiva tridimensional. O algoritmo Knight's Tour, é um problema matemático envolvendo o movimento da peça do cavalo em um tabuleiro de xadrez normalmente num tabuleiro 8x8(Figura 1).

Todas as peças de xadrez têm um movimento específico, tendo movimentos diferentes. Para este algoritmo, é necessário somente a peça do cavalo. Inicialmente coloca-se a peça numa das 64 posições do tabuleiro de xadrez. Com o movimento especifico do cavalo, em formato L, o objetivo é a peça passar por todas as 64 casas, passando somente uma vez por cada uma.

Foi-nos proposto resolver este algoritmo recorrendo a uma implementação que recorresse a backtracking, que é um paradigma algorítmico que tenta diferentes soluções até encontrar uma solução correta.

O backtracking funciona de forma incremental e é uma otimização sobre o algoritmo Knight's Tour.

Neste caso especifico, o algoritmo vai funcionar da seguinte forma:

1. Se todos os quadrados forem visitados, a solução é a esperada, e é impressa essa solução.

2. Se nem todas os quadrados forem visitados ocorre o seguinte:

    a. Adiciona-se um dos próximos movimentos ao array de solução e recursivamente verifica-se se esse movimento leva a uma solução. Inicialmente vamos escolher uma das 8 posições possíveis.

    b. Se esse movimento não obter uma solução , remove esse movimento desse mesmo array e tenta com outro movimento possivel.

    c. Se nenhuma dessas alternativas funcionar, então vai retornar falso. Esse movimento vai ser retirado do array, chamando novamente a função inicial, se essa também retornar falso, é porque não encontrou nenhuma possível solução.

Na primeira versão implementada do algoritmo, o tempo de espera para cada caminho encontrado, era no minimo de 2 minutos e para algumas situações era tão lento que parecia que o mesmo tinha parado. Dessa forma foi necessário recorrer a uma otimização do algoritmo, de modo a que o tempo de espera do cálculo desse mesmo caminho, diminuisse consideravelmente.

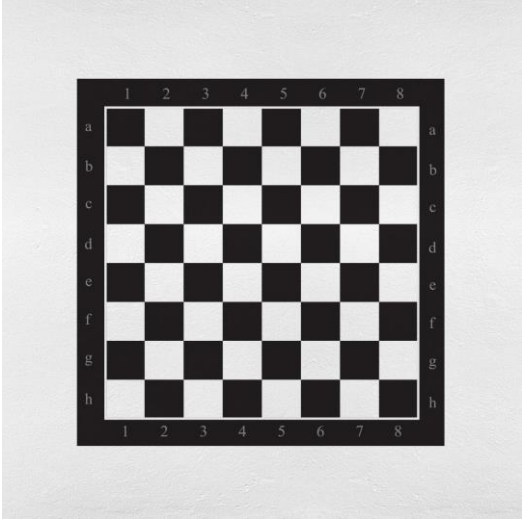

Figura 1 - Modelo base do tabuleiro de xadrez

II. COMPONENTES DO PROJETO

Para a realização deste projeto, foram usados blocos com os quais trabalhamos durante as aulas práticas, e que serviram como ajuda e base para começar a implementação.

Utilizá-mos módulos como o 'maths.js', que permite instanciar e operar com matrizes e vetores para , por exemplo, definer a matriz global de transformação e aplicar transformações elementares, o 'lightSources.js' que permite manipular os focos de luz.

O modulo 'initShaders.js', que permite instaciar os shaders foi também utilizado, mas teve de sofrer alterações para permitir a aplicação de iluminação .

A. Buffers

Para instaciar os elementos necessitamos de 2 buffers por canvas:

No canvas 2D:

- Buffer para instanciar as coordernadas, para qual são passados os vertices.
- Buffer para instanciar as cores dos vértices

No canvas 3D:

- Buffer para instanciar as coordernadas, para qual são passados os vertices.
- Buffer dos vetores normais para a iluminação.

iII. REQUISITOS DA APLICAÇÃO

A – Requisitos gerais da aplicação

As ideias iniciais e que foram implementadas inerentes à realização deste projeto foram as seguintes:

- Entender o algoritmo "The Knight's Tour recursivo com backtracking.
- Visualização de um tabuleiro e de uma peça que simboliza o cavalo.
- Seleção inicial de onde é feita a inicialização do algoritmo.

B – Requisitos para a aplicação 3D

Para a realização da aplicação em 3D, foi necessário as seguintes etapas:

- Visualizar tanto o tabuleiro como a peça em 3D.
- Permitir realizar rotações e translações do tabuleiro e do cavalo como um todo, para permitir uma melhor visualização.
- Permitir escolher o modo de renderização.
- Iluminação presente no tabuleiro e no cavalo.
- Escolher a posição inicial através de input de teclado, para escolher a casa inicial onde quer que se seja iniciado o algoritmo.
- Permitir a translação independente do cavalo.

IV. ELEMENTOS DA APLICAÇÃO

Inicialmente ao executar a aplicação, deparamo-nos com duas janelas, à esquerda a representar o modelo

3D da aplicação e à direita a mostrar o seu funcionamento com uma vista 2D. Para isso foram inicializados dois canvas para permitir essa visualização

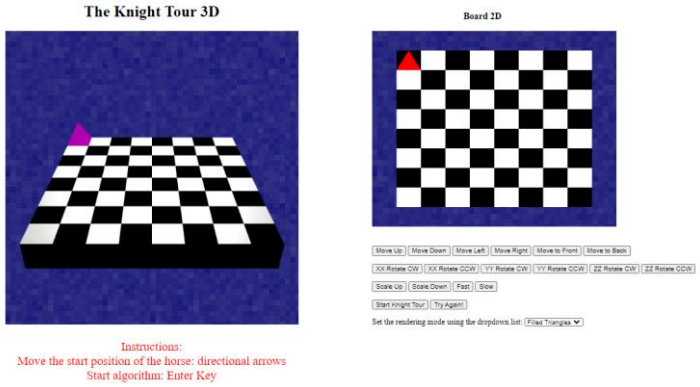

Figure 2 - Imagem inicial

A. Tabuleiro do jogo

O tabuleiro é um dos elementos mais importantes do jogo, sendo que vários cálculos são feitos tendo como base a sua estrutura.

Inicialmente desenhou-se um quadrado com dois triângulos, e definindo cada casa do tabuleiro com uma dimensão de 0.2, recorreu-se a um for loop para criar os restantes 63 quadrados, todos eles criados com 2 triângulos.

B. Peça a simular o cavalo

Antes de desenhar-mos a peça do cavalo em 3D, optámos primeiro por "conectar" o algoritmo Knight's Tour às translações do cavalo e testámos esse desenvolvimento com um simples triângulo.

Posteriormente, depois de termos a certeza que estava funcional, desenhamos uma pirâmide para simular essa peça(3D) no tabuleiro.

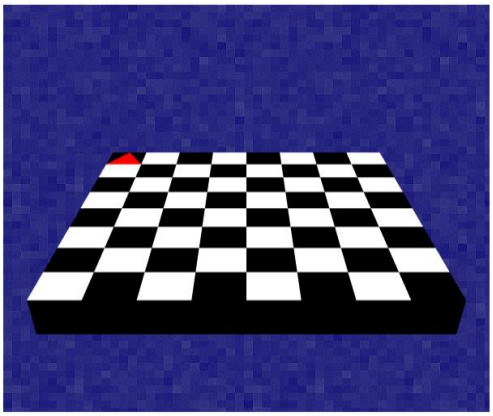

Figure 3 - Fase inicial da peça

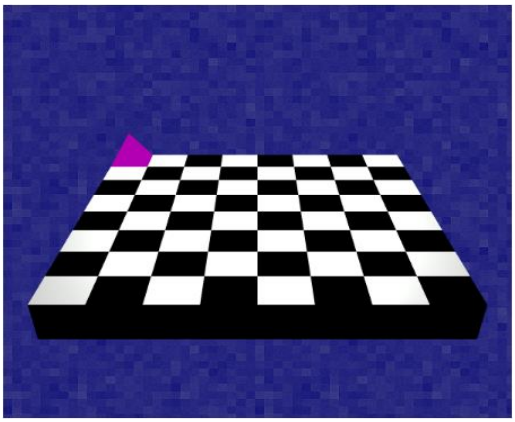

Figure 4 - Fase final da peça

Sabendo como funciona um tabuleiro de xadrez, e as medidas dadas a cada casa do tabuleiro, conseguimos obter coordenadas de uma nova posição do cavalo.

```
for (var k = 0; k < horse2D.length / 3; k++) { //translation horse2D

    var nextX2 = horse2D[3 * k] + 0.2 * (coordX2 - prevCoordX2);
    //console.log(nextX);
    var nextY2 = horse2D[3 * k + 1] - 0.2 * (coordY2 - prevCoordY2);
    //console.log(nextY);
    horse2D[3 * k] = parseFloat(nextX2.toFixed(2));
    horse2D[3 * k + 1] = parseFloat(nextY2.toFixed(2));

}
prevCoordX2 = coordX2;
prevCoordY2 = coordY2;
```

Figure 5 - Translação do cavalo

C. Iluminação

Numa fase inical do projeto optamos por definir cores tanto ao tabuleiro como à peça do cavalo.

Posteriormente decidimos aplicar várias fontes de iluminação para representar o contraste das casas de um tabuleiro de xadrez, e os restantes objetos.

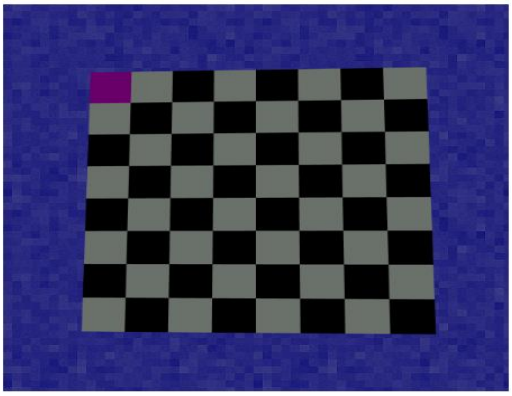

Figure 6 - Sombra causada devido à iluminação

À medida que o tabuleiro é movido em relação aos eixos, pode-se observar a reflexão de luz proveniente de fontes distintas, com diferentes propriedades de intensidade, posição, entre outras.

Para a iluminação do tabuleiro foram criados vetores normais a ele, caso das casas brancas, e noutro sentido para as casas escuras.

As light sources foram posicionadas de acordo com a orientação dos vetores normais, de cada objeto, para haver uma distinção de cores.

Inicialmente, para mostrar ao utilizador o caminho percorrido pelo cavalo, à medida que este se movimenta, foi criado um quadrado com iluminação verde, que ocupava uma casa inteira do tabuleiro, porém após obtermos algum feedback, no final do "Knight's Tour" o tabuleiro ficava totalmente preenchido, logo, decidimos diminuir o tamanho do quadrado, de modo a não acontecer essa situação, mantendo assim a visualização do tabuleiro.

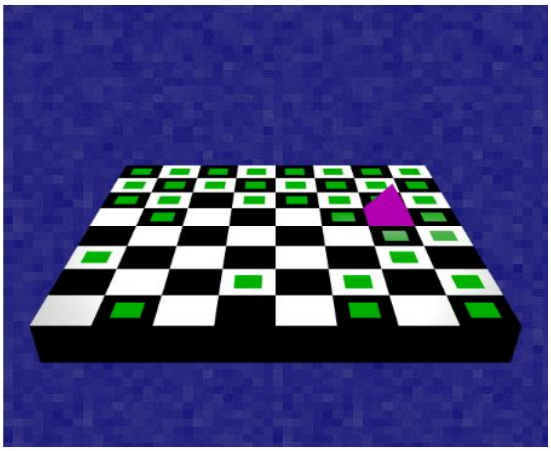

Figure 7 – Visualização do algoritmo

Se rodarmos o tabuleiro, devido às várias fontes de luz presentes no modelo 3D, várias cores irão aparecer tanto no tabuleiro como no cavalo e nos quadrados que representam a evolução do algoritmo.

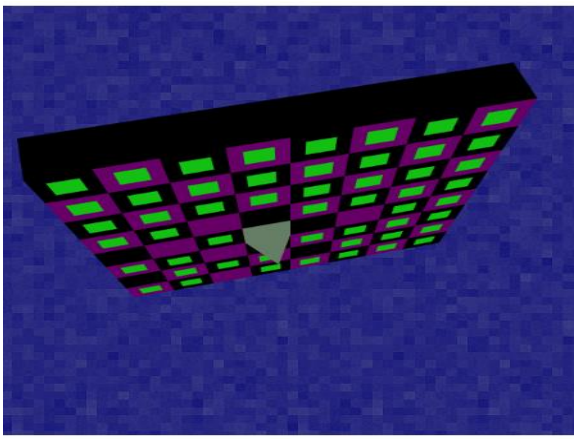

Figure 8 - Fontes de luz em diferentes perspetivas

IV. INFORMAÇÕES EXTRA

No decorrer do projeto, para dar ao utilizador uma visão mais geral do que se pretende com o algoritmo foi feita a tentativa de uma polyline que seguia o movimento do cavalo no tabuleiro.

Como sabemos, uma polyline é uma junção de pontos todos eles interligados entre si, sem loop, ou seja, o primeiro ponto, não se vai conectar ao ultimo.

A partir das posições do cavalo, definimos um ponto no centro de cada casa por onde este passava.

Primeira posição tinha um ponto, que era ligado à segunda casa e assim sucessivamente até o algortimo encontrar o caminho especifico e as 64 casas para percorrer a totalidade do tabuleiro.

Depois de analisar reparamos que, se em vez de traduzirmos os movimentos numa polyline, os representassemos recorrendo ao desenho de quadrados verdes que marcassem as casas passadas pelo cavalo, (falados anteriormente), não seria tão confuso, e traria uma melhor visualização do pretendido ao utilizador.

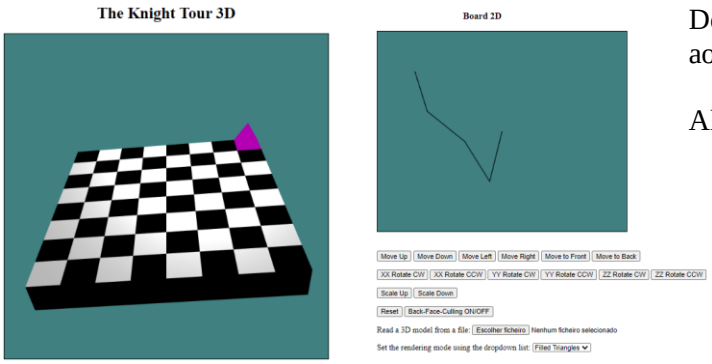

Figure 9 - Tentativa de polyline

Para melhorar a experiência do utilizador à medida que se fazem as translações do cavalo, e quando se inicia o algoritmo, adicioná-mos sons relativos ao cavalo, quando são pressionadas as teclas da aplicação

Relativamente às texturas, tentamos aplicar uma textura ao modelo do cavalo, porém tivemos problemas na integração do tabuleiro e o cavalo em simultâneo no mesmo canvas.

Figure 10 - Textura de um cavalo

IV. INTERFACE

A. Inicío

Optou-se por realizar uma interface simples de compreender e com uma mais valia para a experiência do utilizador.

Ao iniciar a aplicação deparamo-nos com dois canvas, uma para o modelo 3D, e outra para o modelo em duas dimensões, repectivamente. É também apresentado uma breve explicação das instruções necessários para iniciar a aplicação e consequentemente o algoritmo Knight's Tour.

Do lado direito, temos várias opções relativamente ao modelo 3D.

Algumas dessas opções são:

- Rodar o tabuleiro, de modo a observa-lo de vários ângulos.
- Mover o modelo para as algumas direções
- Aumentar/diminuir a escala de visualização do modelo
- Aumentar/diminuir a velocidade do cavalo a resolver o respetivo algoritmo
- Botão try again , para voltar ao modelo inicial, e a possibilidade de escolher novamente outra casa de inicio de algoritmo.
- Modo de renderização do modelo
  - Filled Triangles
  - Wireframe
  - Vertices

Figure 11 - Opções de visualização

B. Controlos da aplicação

Inicialmente deve-se realizar translações ao cavalo através das teclas do teclado (ArrowUp, ArrowDown, ArrowRight, ArrowLeft), e selecionar a casa de onde se quer que se inicie o algoritmo(Enter Key).

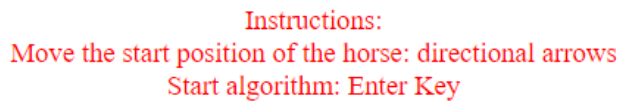

Figure 12 - Intruções de utilização

VI. CONTRIBUIÇÃO DOS AUTORES

O trabalho foi realizado por ambos os membros do grupo, resultando numa participação de 50% para cada um.

V. CONCLUSÃO

Todos os passos realizados ao longo do projeto, desde a revisão do relatório, e à realização deste projeto, foram uma mais valia. O principal objetivo deste trabalho foi atingido, pois foi possivel por em prática os conhecimentos adquirido ao longo do semeste, tanto aproveitar os guiões práticos como as aulas teóricas lecionadas pelos professores.

VI. REFERENCES

[1] Joaquim Madeira, Paulo Dias https://elearning.ua.pt/course/view.php?id=1828

[2] Wikipedia Knigt's Tour https://en.wikipedia.org/wiki/Knight%27s_tour, 2013

[3] Stack Overflow, How to optimize Knight's Tour Problem https://stackoverflow.com/questions/19214109/how-to-optimize-knights-tour-algorithm, 2014.

[4] WebGL Fundamentals, https://webglfundamentals.org/