Universidade da Beira Interior

Departamento de Informática

Maze - 2

Elaborado por:

41237 - Pedro Manuel Pires Lopes
41400 - Luís Miguel Freitas do Espírito Santo

14 de janeiro de 2021

Conteúdo

Capítulo

1

Introdução

1.1 Enquadramento

Este relatório foi feito no contexto da unidade curricular de *Computação Gráfica* da **UBI!** (**UBI!**), com o sentido do desenvolvimento de uma aplicação *3D* com recurso às tecnologias *OpenGL*.

1.2 Motivação

Este trabalho motivou-nos por se tratar de um jogo onde poderíamos incorporar elementos da nossa imaginação para desenvolver um jogo de terror e por também ser uma área de interesse onde poderemos assim ganhar experiência no manuseio de tecnologias que, até à sua abordagem nesta unidade curricular, nos eram estranhas.

1.3 Objectivos

1. Gerar um labirinto aleatório;
2. Converter o labirinto num mapa a três dimensões;
3. Implementar uma câmara *FPS* que emite luz;
4. Construir os *shaders* das paredes;
5. Criar um sistema de colisões;
6. Construir um sistema de texto;

7. Criar texto auxiliar ao jogador;
8. Criar um menu, com base no sistema de texto;
9. Adicionar áudio, *No-Clip* e um mapa;

1.4 Organização do Documento

Para que este documento esteja encadeado de forma a explicar o projecto realizado, encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projecto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objectivos e a respectiva organização do documento.
2. O segundo capítulo – **Introdução as Tecnologias Utilizadas** – descreve os conceitos mais importantes no âmbito deste projecto, bem como as tecnologias utilizadas durante do desenvolvimento do projecto.
3. O terceiro capítulo – **Construção do Labirinto e Sistema de Detecção** – aqui se descrevem os passos detalhados para o manuseio de colisões no *OpenGL* e da implementação do labirinto em *C++*.
4. O quarto capítulo – **Texto, Menu e funcionalidades extra** – Aqui explicamos a adição detalhada de funcionalidades extra tal como o áudio, *No-Clip*, mapa e *spawn* do jogador.
5. O quinto capítulo – **Conclusões e Trabalho Futuro** – Aqui falamos sobre as conclusões que tiramos deste trabalho e o que aprendemos e poderá ser implementado no futuro. Falamos sobretudo do desempenho do nosso projecto em nível de funcionalidades consoante o que nos comprometemos a realizar.

Capítulo

2

Introdução às Tecnologias Utilizadas

2.1 Introdução

Neste capítulo iremos explicitar as tecnologias que estamos a utilizar no nosso projecto de modo a garantir a compilação e execução da nossa aplicação *Maze - 2*.

2.2 *C++*

A linguagem de programação *C++* é uma linguagem sucessora à famosa linguagem C, tendo algumas funcionalidades novas como o suporte a classes. Esta linguagem, por apresentar muitas semelhanças com a linguagem C, tem também muitas das suas vantagens, como a programação de baixo nível possibilitando um recurso manual dos recursos do sistema.

2.3 *OpenGL*

A biblioteca gráfica *OpenGL* permite a renderização de vectores de duas e três dimensões, utilizando a *GPU* para atingir renderização acelerada pelo *hardware*.

2.4 *GLFW*

A biblioteca *GLFW* permite ao utilizador criar janelas gráficas, capazes de reproduzir imagens renderizadas, *input* através do rato, do teclado, ou outros, dessa forma permitindo controlar a janela da forma que esta fora configurada.

2.5 *GLSL*

A linguagem *GLSL*, também conhecida como *OpenGL Shading Language* permite com que o programador tenha acesso directo ao *pipeline* gráfico sem que este tenha de recorrer a linguagens específicas do *hardware* ou a uma linguagem *assembly*.

2.6 *FreeType*

A biblioteca *FreeType* permite ao programador criar texto gráfico de forma fácil recorrendo apenas a uso de *shaders* para desenhar caracteres em duas ou três dimensões.

2.7 *Assimp*

A biblioteca *Assimp* permite carregar objectos com muitos triângulos através de malhas, permitindo ter objectos complexos, de forma eficiente.

2.8 *Glm*

A biblioteca *Glm* permite aplicar operações entre matrizes e vectores permitindo criar de forma fácil matrizes, isto é, permite controlar os objectos de forma fácil para o programador.

2.9 *Stb*

A biblioteca *stb* permite carregar imagens de forma eficiente, esta é uma biblioteca complementar para a construção das malhas que permitem renderizar o objecto.

2.10 *IrrKlang*

A biblioteca *irrKlang* permite carregar e executar ficheiros de áudio em duas ou três dimensões, esta biblioteca é suplementar ao projecto visto que trabalha com áudio quando o projecto é sobre computação gráfica, no entanto, complementa o ambiente do labirinto por introduzir mais um elemento ao jogo.

2.11 Conclusões

Podemos concluir que o nosso trabalho se baseou fortemente nas utilização da linguagem *C++*, embora existam outras linguagens que também podem ser utilizadas para a criação de aplicações bidimensionais e tridimensionais. A nossa utilização maioritária da linguagem *C++* em detrimento de outras candidatas pode explicar-se por ter sido esta a linguagem usada na instrução desta unidade curricular e em termos já uma experiência considerável com a linguagem *C*, experiência essa que se revelou imperativa para a bem-sucedida realização deste trabalho. Houve também um forte uso de algumas bibliotecas que permitiram a produção deste projecto parecer mais fluida e fixante.

Capítulo

3

Construção do Labirinto e do sistema de detecção

3.1 Introdução

Aqui serão explicados: o processo de criação do labirinto; a detecção de uma parede; e a construção das paredes em três dimensões, através da livraria *OpenGL*.

3.2 Inicialização do Labirinto

A criação do labirinto é um processo que se revela interessante na qualidade das decisões por nós tomadas durante a implementação do mesmo. O primeiro passo para a criação de um labirinto é a criação de um *array* de duas dimensões de tamanho igual, este tamanho é ajustável no código apresentado através da manipulação da variável *sized*, este *array* é então enviado para uma função que o inicializa a zeros e de seguida é aleatoriamente escolhida uma posição inicial que vamos usar no próximo passo para começar o preenchimento deste *array* com os dados referentes ao labirinto.

3.2.1 Criação dos Caminhos

Esta próxima etapa consiste em criar os caminhos que vão, em colectivo, constituir o nosso labirinto no seu estado final. Toda esta etapa é abordada na função *CriaCaminho*, esta função vai funcionar de forma recursiva, escrevendo uma posição de cada vez para um caminho de cada vez, para entender de forma completa o seu funcionamento há que primeiro abordar os parâmetros

que recebe, são estes: o *array*; o par de coordenadas cartesianas dos índices do *array* x e y guardados nas variáveis *i* e *j* respectivamente; um inteiro *d* que servirá como a direcção usada pela função em destaque, tendo sempre um de 4 valores simbolizando cada um uma direcção: 0 para a esquerda, 1 para cima, 2 para a direita e 3 para baixo; dois contadores, *contd* e *contb*, os quais representam a quantidade de "turnos"a decorrer sem tentar alterar a direcção ou criar uma ramificação do actual caminho, estas duas variáveis são decrementadas em cada execução desta função; e por fim uma variável booleana que indica se este caminho é o caminho original ou uma ramificação do mesmo.

3.2.2 A questão da direcção e das ramificações

A cada execução desta função a primeira tarefa a realizar é a de verificar tanto o *contd* como o *contb* para poder decidir se vai tentar mudar a direcção do caminho actual, caso em que chamará uma função para testar esta mudança consoante a função aleatória, havendo igual probabilidade para os dois lados adjacentes à direcção actual e uma pequena probabilidade de manter a mesma direcção, ou tentar criar uma ramificação, caso em que, mediante o recurso a números escolhidos aleatoriamente, haverá uma pequena chance de criar uma ramificação para um dos lados adjacentes à direcção actual (caso esse caminho não esteja obstruído) e uma pequena chance de criar uma ramificação nos dois lados, a criação de ramificações nada mais nada menos é que a chamada recursiva desta mesma função, passando como parâmetros os pretendidos para a ramificação nova. Caso algum destes testes verificar o valor verdadeiro (ou ambos) as variáveis correspondentes (*contd* e *contb*) receberão um valor aleatório entre 4 e 6.

3.2.3 O mover dos caminhos

As instruções que se seguem são de verificar se é seguro para o caminho continuar na sua direcção actual e tanto as ramificações como o caminho principal têm uma parte inicial igual em que é chamada uma função que verifica se o caminho pode prosseguir na sua direcção actual e que, se não puder, altera a direcção um máximo de duas vezes, finalizando a execução com um *return* se nenhuma direcção for possível. No entanto, se o caminho de uma ramificação se tratar há uma verificação adicional que às ramificações é exclusiva: a verificação de conexão. Esta verificação verifica se movimentar o caminho na direcção pretendida levaria à junção com outro caminho, se isto se verificar então haverá uma probabilidade igual de seguir em frente ou de tentar criar uma ramificação nesse instante. É de especial interesse o final deste segmento de código onde encontramos um *return* dentro de uma instrução *if*, se retirar-

mos este *return* o labirinto fica sem becos, caso o mantenhamos sempre, sem instrução *if* o labirinto é composto apenas por becos, a actual configuração com o *if* tem como objectivo misturar estas duas realidades num labirinto onde haverão becos e caminhos cruzados. O resto da função tem como única funcionalidade efectivamente mover o caminho na direcção pretendida, alterando o *array* e decrementando os contadores. Caso o caminho se encontre com outro terminará aqui os seus movimentos com um *return*.

3.2.4 Todo

3.2.5 Renderização das paredes

Para que se pudesse transformar a matriz que produziu o labirinto num conjunto de cubos que representassem um mapa, necessitou-se de filtrar desenhos inúteis, isto é, evitar o gasto de recursos a desenhar blocos em caminhos, para tal simplesmente foram guardadas as posições dos elementos que representavam paredes para a produção de um *array* de vectores a três dimensões com parâmetros x,y e z. Também foi implementado um limitador de objectos renderizados para aumentar a *performance*, foi usado o *Field of View* do jogador para renderizar uma circunferência de objectos visíveis ao jogador.

3.2.6 Iluminação das paredes

Para que as paredes não tivessem uma cor estática foi colocado um um conjunto de estruturas no *shaders* das paredes que permitem manipular a quantidade de luz emitida causando assim um efeito de escurecimento que ser-se-ia removendo conforme a aproximação da fonte de luz do jogador, a lanterna. Por sua vez esta não ilumina tudo, mas apenas um segmento circular que representa a circunferência que a lanterna produz.

3.3 Colisão

Ora como isto representa um labirinto, não faria sentido ser possível atravessar paredes, para tal foi preciso implementar um sistema que soubesse quando dois objectos se encontravam, no caso a câmara e as paredes.

3.3.1 Detecção de Colisão

Para que fosse possível detectar uma colisão foi usado o algoritmo *AABB* que com base na posição de factores indica se este ponto pertence ao outro ob-

jecto. No entanto, no nosso caso foi preciso normalizar a posição dos objectos, visto que, o ponto que representa uma parede não representa todo o espaço que esta ocupa, para resolver isto foi feito um *shift* de menos meio valor nas coordenadas x e z de forma a se ter um sistema de pontos centralizados num plano unitário.

3.3.2 Tratamento da Colisão

Após estar a ser detectada a colisão foi necessário implementar um sistema que bloqueia o movimento do jogador para dentro da parede, para tal sempre que o jogador tente ir contra uma parede, este retorna para a posição anterior ao choque, de tal forma a que efectivamente este não consiga entrar em paredes.

3.4 Saída

O nosso labirinto tem uma característica que o diferencia dos labirintos que normalmente encontramos em jogos de labirintos como o nosso, esta característica é a ausência de uma saída inicialmente visível, dando a aparência de um labirinto sem solução. No entanto, a solução existe mas encontra-se visualmente "escondida"do jogador, apenas se tornando visível quando o jogador se aproxima da mesma. Para calcular uma saída filtramos todos os blocos que estão distantes o suficiente das coordenadas actuais do utilizador (usando a distância euclidiana) e depois escolhemos um deles usando mais uma vez a escolha de números aleatoriamente.

3.5 Conclusões

Aqui foi demonstrado como foi cuidada a construção e a interacção do jogador com as paredes do labirinto e com a saída. A lógica principal do jogo foca-se nesta região que por si só é responsável por cuidar de noventa por cento do projecto, mas sem nunca deixar de referir que é também a região onde encontramos grandes dificuldades por ser esta a mais complexa e mais extensa parte do projecto.

Capítulo

4

Texto, Menu e funcionalidades extra

4.1 Introdução

Aqui encontrar-se-á explicado como funciona a parte do menu do texto e das funcionalidade extras como o *No-Clip*, o mapa e o áudio.

4.2 Texto

Para que alguém se possa guiar num labirinto é preciso que tenha informações sobre o mesmo, para isso foi decidido que ter-se-iam: um conjunto de texto informativo com os quadros por segundo; a latência; a posição do jogador no plano; a posição da saída; e o tempo desde o inicio do jogo. Foi criado também através da activação da tecla "H"um conjunto auxiliar de informações ao utilizador como os comandos, e as *flags* activas.

Para se ter texto recorreu-se à biblioteca *FreeType* que disponibiliza funções que pegam em tipos de letras e convertem em glifos, assim reduzindo apenas para o programador o trabalho de este criar o *Shader* que vai representar o carácter, com base nas texturas.

4.3 Menu

Para desenhar o menu decidiu-se algo à base de texto, mas para evitar que fosse afectado o labirinto decidiu-se mover o jogador para uma posição no mundo onde nada seria renderizado. Para representar o menu à base de texto

foram criadas três funcionalidades: uma para sair do jogo; outra para iniciar o jogo e activar o relógio; e outra para desligar a musica.

4.4 Funcionalidades Extra

Aqui será referido o que foi adicionado extra para aumentar a inclusão do jogador no labirinto.

4.4.1 Áudio

Como em qualquer jogo em 2020, é impensável não ter áudio e portanto foi usada a biblioteca *IrrKlang* que disponibiliza uma *engine* de som para o programador gratuita, com capacidade de simular áudio em duas e três dimensões. Aqui foram escolhidos três sons que acontecem de forma independente uns dos outros: um som de ambiente para simular o medo e a natureza da noite num lugar escuro e sombrio; um som de um grito que ocorre a cada dois minutos, auxiliado pela perda de visão do jogador por instantes; e um som de vitória que aparece quando este chega ao fim do labirinto emitindo um texto que muda a cada quadro de cor.

4.4.2 No-Clip

Esta é uma funcionalidade que se encontra presente na maioria dos jogos *FPS*, esta ignora qualquer sistema de colisão e deixa o utilizador voar a altas velocidades, ora, isto é uma funcionalidade útil para analisar tanto a geração do labirinto como os padrões gerados, esta funcionalidade é também útil para testar bugs. O *No-Clip* pode ser activado com a tecla "N"e pode ser usado ao mesmo tempo que o mapa.

4.4.3 Mapa

Aqui foi implementado um sistema diferente do sistema de mapas de *FPS*, ora este mapa tem como objectivo dar um auxílio ao jogador, para que este não perca muito tempo na mesma localização, simplesmente foi retirada toda a velocidade da câmara neste estado (salve a excepção do *No-Clip*), e é colocada na altura y igual a dez onde permite ter alguma visão do jogo, mas não suficiente para revelar o jogo pela totalidade. Esta funcionalidade activa-se através da tecla "M".

4.4.4 *Spawn* do jogador

Para aumentar a fasquia do jogo, foi decidido que o *spawn* do jogador é diferente em cada labirinto, isto permite com que caso aconteça ser gerado o mesmo labirinto, o jogador ainda tem de percorrer um caminho diferente para chegar a um final diferente.

4.5 Conclusões

Aqui introduziram-se ligeiramente algumas das funcionalidades disponíveis no jogo, sendo que as que não foram aqui abordadas têm origem (complementando) as funcionalidades que foram aqui referidas. Faltou serem levantadas algumas das funcionalidades do jogo, que não chegaram a ser implementadas, como é o caso do mini-mapa.

Capítulo

5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

Este trabalho permitiu com que se conhece-se um outro lado da programação que se foca em aspectos diferentes dos que os alunos estão acostumados, também se provou aqui que a manipulação de matrizes é essencial para que se possa trabalhar com a geometria a duas e três dimensões.

5.2 Trabalho Futuro

O que é que ficou por fazer, e porque?
Terminar de implementar as paginas com o objectivo de ter texto que cause medo as pessoas, baseado no jogo de terror *Slender*.

O que é que seria interessante fazer, mas não foi feito por não ser exactamente o objectivo deste trabalho?

Implementar uma *IA* que persegue o jogador para o matar, que aumentar a frequência que esta *IA* aparece com base nas paginas encontra-das.

Em que outros casos ou situações ou cenários – que não foram estudados no contexto deste projecto por não ser seu objectivo – é que o trabalho aqui descrito pode ter aplicações interessantes e porque?

Alguns algoritmos aqui desenvolvidos podem ser aplicados para a geração de outros jogos, por exemplo o algoritmo de geração de labirintos é algo que não fazia parte dos objectivos da cadeira mas que fazem parte do contexto do trabalho.