Computação Gráfica

1º Semestre

2020/2021

Tetris 2D

Docente:

Abel Gomes

Discentes:

André Salgado (41223), Guilherme Lopes (41558), Rafael Vitória (41857)

Conteúdo

Motivação

Neste projeto pretende-se implementar o jogo Tetris em 2D, que consiste em empilhar blocos dentro de um repositório retangular. Quando uma linha se forma, ela desintegra-se e as camadas superiores descem uma linha, o que resulta num acumular de pontos para o jogador. Quando a pilha de peças chega ao topo do repositório, a partida termina. O Tetris foi desenvolvido por desenvolvido por Alexey Pajitnov, Dmitry Pavlovsky e Vadim Gerasimov no ano de 1984.

Tecnologias

Ao desenvolver esta proposta recorremos a diversas tecnologias, maioritariamente utilizadas nas aulas. Utilizámos a linguagem C++, bem como as bibliotecas OpenGL, GLFW, GLEW, GLM e Common. Recorremos também a algumas ferramentas de edição de imagem, o GIMP e o Blender, e a outra de gestão de Projetos, o Trello.

Gestão do projeto

A divisão de tarefas foi feita através do Trello, ferramenta mencionada anteriormente. Começámos por definir a lógica referente ao jogo. O André ficou encarregue de construir os “tetranóminos”, o Guilherme ficou encarregue de construir o tabuleiro e a mecânica de movimento das peças, por fim, o Rafael ficou encarregue de definir o posicionamento das peças. O passo seguinte foi modelar e renderizar os objetos da cena e definir o método de observação. O Rafael modelou as peças, o Guilherme renderizou-as e o André definiu o método de observação. No que toca à texturização do ambiente, o Guilherme texturizou as peças, o Rafael o fundo e o André apresentou as letras. O Guilherme foi o autor dos shaders principais e o André utilizou shaders já fornecidos para o desenho das letras (com uma pequena alteração).

Descrição do código

Main

```
int main(void) {
    startState();
}
```

Começamos por verificar o estado do programa com uma função feita por nós.

```
//Diz se estamos no menu ou no jogo
void startState() {
    while (true) {

        switch (state) {
            case 'J':
                startGame();
                state = 'M';
                break;
            case 'M':
                setup();
                Menu();
                break;
        }
    }

}
```

Caso o estado seja ‘J’, referente a jogo, começamos o jogo e definimos o estado como ‘M’, de menu, para a seguir abrir o menu como podemos ver a seguir no caso ‘M’.

```
void setup() {

    // Initialise GLFW
    glfwInit();

    glfwWindowHint(GLFW_SAMPLES, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // To make
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

    // Open a window
    window = glfwCreateWindow(WINDOW_W, WINDOW_H, "Tretis", NULL, NU

    // Create window context
    glfwMakeContextCurrent(window);

    // Initialize GLEW
    glewExperimental = true; // Needed for core profile
    glewInit();

    // Ensure we can capture the escape key being pressed below
    glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);

    // White background
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

}
```

Começamos por fazer *setup* do esquema, como fazíamos nos trabalhos práticos da cadeira de Programação Gráfica. Criando as janelas e inicializando as bibliotecas.

```
void Menu() {
    initText2D("Holstein.DDS");
    glGenVertexArrays(1, &VertexArrayID);
    glBindVertexArray(VertexArrayID);

    while (showMenu) {
        drawMenu();
        if (listenKeys()) {
            state = 'J';
            cleanGPU();
            return;
        }
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
}
```

Na função Menu() começamos por inicializar o shadder que nos permite utilizar as letras com a font “Holstein” e os vértices que vão ser utilizados. Depois desenhamos o menu com a função drawMenu() que utiliza as funções para desenhar o texto com a cor definida e atualizamos o estado e a gráfica consoante a instrução dada pelo input.

No caso ‘J’ onde começamos o jogo, chamamos a função startGame().

```
void startGame() {
    Game game = Game(window);
    game.Start();
    state = 'M';
    startState();
}
```

Onde inicializamos a classe jogo com a janela que demos setup anteriormente. Após a sua inicialização, começamos o jogo com o seu método Start(). Quando o jogo acabar o estado volta a ser 'M' voltando para o menu.

Game

```
    bool rotateFlag = true;
    bool tick();

    bool listenMoves();

    void render();

    void draw();

    void transferDataToGPUMemory(void);

    void cleanGPU();

    void gameOverWait();
};
```

A classe jogo (game) também nos disponibiliza uma função chamada tick().

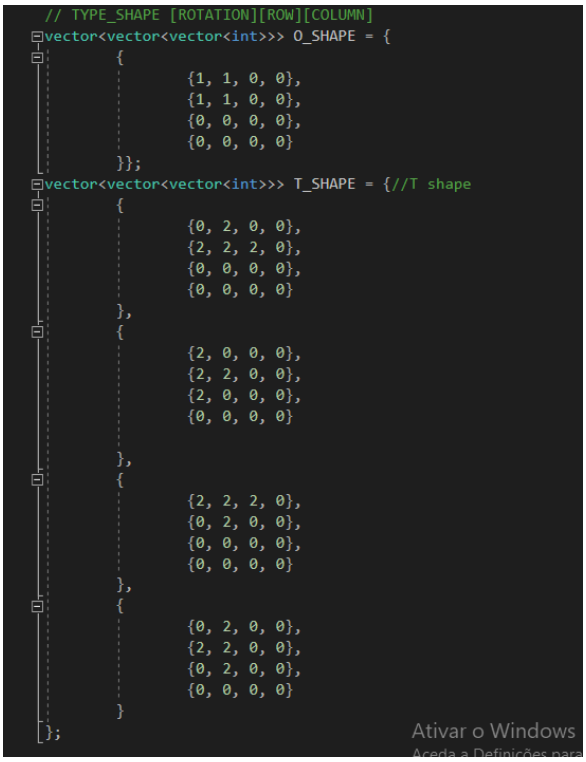
```
bool Game::tick() {
    bool running = true;
    Shape shape = Shape();
    gameBoard.insertShape(&shape);

    while (running) {
        render();


        if (sleptTicks % 4 == 0) {
            running = listenMoves();
        }
        if (sleptTicks <= sleepTicks) {

            sleptTicks = sleptTicks + 1;
            continue;
        }
        sleptTicks = 0;
        if (!gameBoard.move_piece('D')) {
            vector<int> clearedRows = gameBoard.clearFullRows();
            if (pontos != pontos + clearedRows.size()) {
                pontos += clearedRows.size() * clearedRows.size();
                sleepTicks = sleepTicks - sqrt(pontos);
            }

            running = !gameBoard.currentShape->hasNegative();
            if (!running) {
                fflush(stdout);
                printf("FINISHING");
                return true;
            }
            shape = Shape();
            gameBoard.insertShape(&shape);
        }
    }
    return false;
```

Esta função cria uma peça e insere-a no tabuleiro. Com isto depois durante o decorrer de cada tick a peça vai descendo até ser limitada pelo fundo do tabuleiro.

A função listenMoves() recebe o input do teclado e manda a peça mover consoante a direção definida pelo valor de entrada.

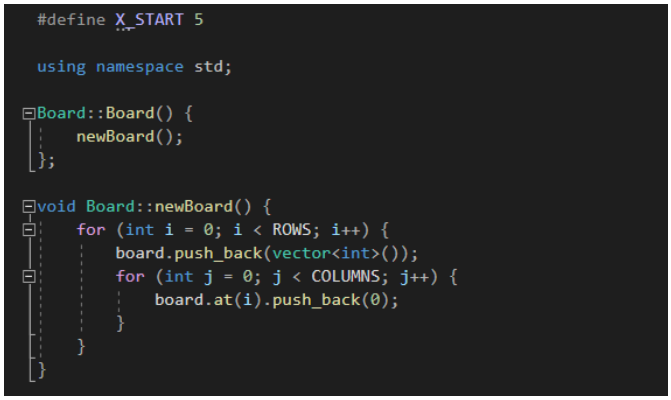
A função render() serve para renderizar as instruções, chamando a função draw() que é onde são definidos os dados desenhados. O draw() tal como foi dito, foi utilizado para desenhar as peças , o tabuleiro e definir a posição da câmara. Estes dados são transferidos para a GPU com a função transferDataToGPUMemory().

Shape



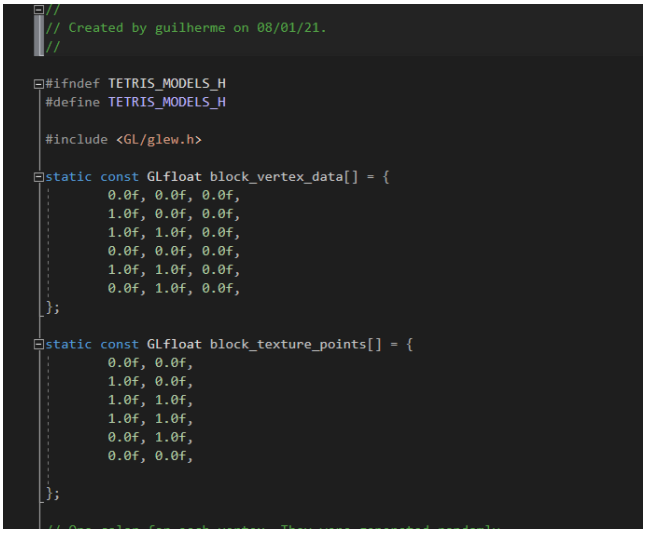
As peças são representadas através de uma matriz 4x4, onde os índices que têm números diferentes de 0, vão representar a posição da peça. Cada número vai representar uma cor, assim sabemos qual cor devemos renderizar para aquela peça. Esta classe também contém vários métodos que fornecem a informação necessária para mexer com as peças.

Board



A classe board, cria uma matriz 20x10 que vai representar o tabuleiro. Contém também vários métodos que facilitam a sua manipulação.

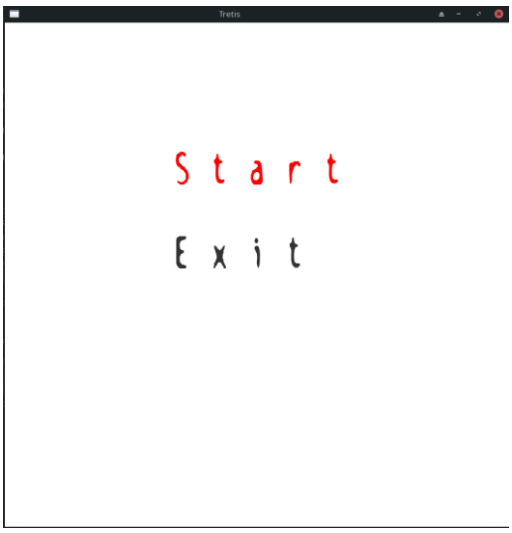
Models



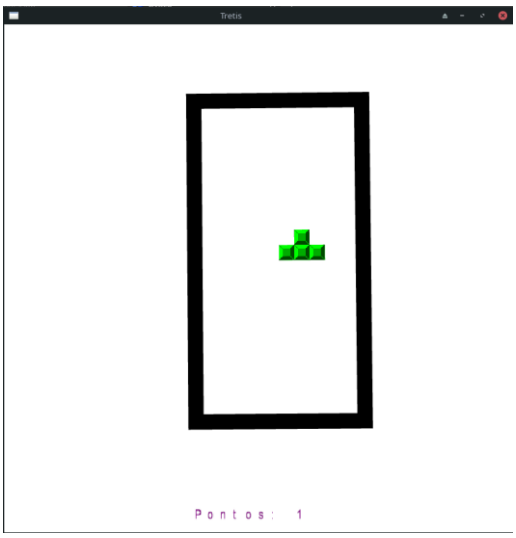
```
//
// Created by guilherme on 08/01/21.
//

#ifndef TETRIS_MODELS_H
#define TETRIS_MODELS_H

#include <GL/glew.h>

static const GLfloat block_vertex_data[] = {
        0.0f, 0.0f, 0.0f,
        1.0f, 0.0f, 0.0f,
        1.0f, 1.0f, 0.0f,
        0.0f, 0.0f, 0.0f,
        1.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
};

static const GLfloat block_texture_points[] = {
        0.0f, 0.0f,
        1.0f, 0.0f,
        1.0f, 1.0f,
        1.0f, 1.0f,
        0.0f, 1.0f,
        0.0f, 0.0f,

};
```

No models.h são definidos os vértices dos triângulos que formam um quadrado das peças. Também são definidas a textura e as cores.

Descrição do funcionamento do Software



Quando a aplicação é iniciada é apresentado um menu para começar o jogo ou para sair dele.



Após iniciar o jogo, são apresentados um tabuleiro de jogo e a nossa pontuação. Vamos ganhado pontos à medida que completamos uma fila de peças. Começam também a cair as peças, dando a hipótese ao jogador de acelerar a queda da peça ou de efetuar uma rotação da mesma.

Quando o jogador perde é apresentada no ecrã uma frase a dizer que perdeu com a sua pontuação final.

Conclusões

Após a realização do projeto, conseguimos alcançar quase todos os objetivos propostos.
Conseguimos modelar as peças originais do Tetris, replicar a sua movimentação, conseguimos interagir com o teclado e ainda texturizar as peças do jogo.
Ainda assim, é possível melhorar bastante o trabalho no futuro. A principal dificuldade encontrada foi a implementação de iluminação nas peças. Não conseguimos implementar esta funcionalidade, mas pretendemos consegui-lo fazer no futuro.
Em suma, podemos dizer que há maior confiança para realizar mais projetos futuros relacionados com a unidade curricular em si, não excluindo a possibilidade de ainda melhorar muito as nossas técnicas referentes a computação gráfica em C++.