Universidade da Beira Interior
Departamento de Informática

Projeto de Computação Gráfica

Realizado por:
Vinícius Costa – 40681
Vítor Neto – 41178
Caio Martins – 41375

Docente:
Professor Doutor Abel J.P. Gomes

9 de Janeiro de 2021

Table of Contents

Capítulo 1

Introdução

1.1 Descrição do problema

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.

1.2 Motivação

A principal motivação deste projeto é analisar o problema proposto, e com base nos conhecimentos adquiridos na disciplina de Computação Gráfica durante o semestre, arquitetar e implementar o jogo Tetris 2D.

1.3 Tecnologias Utilizadas

Para implementar o Tetris 2D utilizamos a linguagem de programação C++ e as bibliotecas OpenGL, GLFW, GLEW e GLM e como ferramentas de versionamento de código e organização de projeto optamos por usar Git + GitHub e Trello.

Capítulo 2

Implementação e funcionamento

2.1 Etapas de desenvolvimento

Para o desenvolvimento da solução, consideramos importante estudar alguns aspectos e peculiaridades relativamente ao jogo Tetris, nomeadamente, como é guardada a informação referente ao tabuleiro, como são geradas as peças e suas respectivas rotações e pontuação. Após o estudo do jogo a implementar, analisamos então, como traduziríamos os requisitos acima descritos para uma solução utilizando OpenGL.

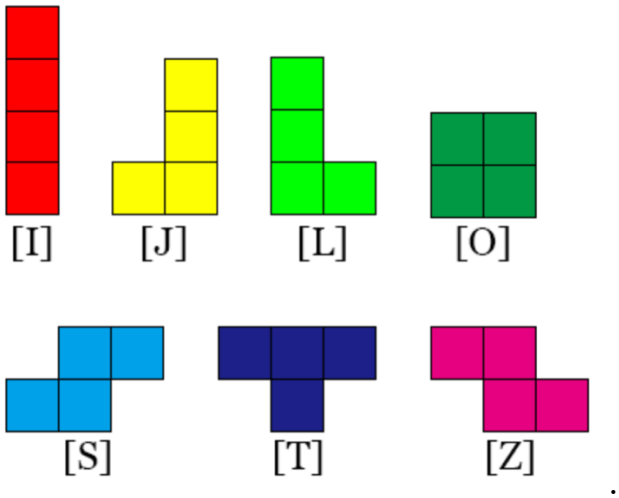
Primeiro começamos por definir um *board* (local onde serão alocadas as peças). Este *board* é representado por uma classe que contém como atributos:

game_board – uma matriz 10x20 inicializada a 0 em todas as suas posições, que armazenará a posição atual de cada peça e o estado do jogo. Cada peça é representada no tabuleiro pelo número de sua *shape.*

***score* –** uma variável inteira que armazena a pontuação atual do jogador e funciona da seguinte maneira: a cada deslocação no eixo Y que uma peça faz, o jogador ganha 10 pontos e caso o mesmo consiga completar uma linha, é recompensado com mais 100 pontos.

Pare além destes dois atributos, a classe Board também possui métodos responsáveis pelo movimento de cada peça no tabuleiro.

Em segundo, começamos a modelar as peças originais do jogo e usamos a seguinte imagem para nos guiar.



Tetrominos originais do jogo Tetris

Para isto, criamos uma classe *Tetromino* em que cada objeto instanciado desta classe, representa uma peça em seu ciclo de vida. Os principais atributos do Tetromino são:

- ***shapes[7][4][4][2]*** – é um array de 4 dimensões onde:

    1. A primeira dimensão representa as peças existentes. (7)
    2. A segunda dimensão representa as rotações existentes para cada peça. (4)
    3. A terceira dimensão representa os quadrados que compõem uma peça. (4)
    4. A quarta dimensão representa os valores X e Y para cada quadrado referido no ponto anterior.

- ***shape*** – inteiro que representa um identificador único para tipo de tetromino.

- ***rot*** – inteiro que representa em que estado de rotação está o tetromino.

- ***prevX*** – localização no eixo X que o tetromino ocupa (utilizado como auxiliar para calcular a nova posição quando um tetromino sofre rotação).

- ***prevY -*** localização no eixo Y que o tetromino ocupa (utilizado como auxiliar para calcular a nova posição quando um tetromino sofre rotação).

- ***finished –*** booleano responsável por sinalizar se a peça já chegou no destino final.

- ***x –*** array que guarda as coordenadas X dos 4 quadrados que constituem uma peça.

- ***y –*** array que guarda as coordenadas Y dos 4 quadrados que constituem uma peça.

Decidimos também, que sempre que for chamado o construtor desta classe, os valores de *shape* e *rot* são gerados aleatoriamente de modo a imitar o comportamento do jogo original.

Tendo o *board* e as peças (*Tetrominos*) implementadas, começamos a analisar como seria feito o movimento das peças e chegamos à conclusão de que as peças se movem nas seguintes situações:

- De x em x tempo a peça se move para baixo.
- Sempre que o jogador usar mover o mouse para esquerda ou direita, a peça deve acompanhar este movimento.
- Sempre que o jogador usar as teclas reservadas para o movimento (seta esquerda, direita e para baixo) a peça deve acompanhar este movimento.
- Sempre que o jogador usar a tecla para mudança de rotação (seta para cima), a peça deve rodar, implicando diferentes posições nos eixos X e Y.
- Sempre que o jogador completar uma linha, esta deve desaparecer e todas as outras linhas devem descer uma posição.

Para isso, criamos diversos métodos responsáveis por cada um dos movimentos acima citados, na classe *Board* que serão abordados com mais detalhe na secção seguinte.

Por fim, definimos os objetos gráficos que apareceriam em tela e criamos seus respectivos *color* e *vertex buffers*:

- *Board* - boardbuffer
- *Grid* – vlinebuffer, hlinebuffer
- *Block* – blockbuffer

E para as cores, optamos por gerar *colorbuffers* de diversas cores, e depois cada um destes ser atribuído a uma peça do jogo.

2.2 Descrição do funcionamento do Software

Primeiramente, começamos por inicializar as variáveis que o jogo depende, nomeadamente, criamos uma instância da classe Board e uma da classe Tetromino que correspondem a o estado inicial do tabuleiro (0 em todas as posições), e a primeira peça do jogo (gerada aleatoriamente no construtor) respectivamente. Para além disso, inicializamos também a variável **m_clock**, responsável pelo tempo em que uma peça demora a cair no eixo Y.

Tendo as principais variáveis inicializadas, desenhamos o board na janela e a partir disto atualizamos o estado do tabuleiro já com a peça, isto é, na variável *board.game_board*, introduzimos o valor *tetromino.shape* (identificador único para cada tipo de peça) nas posições que a peça ocupa.

Tabuleiro quando uma peça "I" ocupa sua primeira linha

Após o tetromino estar dentro da matriz game_board e em ciclos de **m_clock**s, percorremos o tabuleiro e sempre que encontramos um valor diferente de 0, significa que encontramos um bloco de uma peça. Para que o processo de desenhar corretamente no *board* fosse possível, optamos por definir coordenadas para cada posição do *game_board,* por exemplo, a posição game_board[0][0] corresponde ao ponto -25.0f, 50.0f, 0.0f da nossa janela.

Ao encontrarmos um ID de uma peça no tabuleiro, verificamos as cores previamente definidas para cada tipo de tetromino, e passamos o *blockbuffer* juntamente com o respetivo *colorbuffer* para a função *draw()* realizando assim, o desenho da peça completa nas coordenadas correspondentes.

Durante o processo acima citado, o programa tem *listeners* que captam input do usuário para movimento de peça (setas), restart (R), quit (ESC).

Caso o utilizador escolha fazer um movimento, criamos funções específicas para cada movimento, uma vez que esta ação envolve calcular as novas coordenadas da peça, verificar se as novas coordenadas são válidas e por fim limpar as antigas coordenadas. As funções citadas estão dentro da classe Board.

Capítulo 3

Conclusão

3.1 Trabalhos Futuros

Apesar do projeto ter sido entregue num estado funcional este pode-se considerar incompleto, anuncia-se aqui então que características do software e que outras novas funcionalidades podem ser melhoradas/adicionadas; Continuar implementação de textura nos "blocos", pois devido a limitações de tempo não foi possível terminar, mostrar pontuação na *window* ao invés de na linha de comandos, implementar um menu introdutório, ambas estas funcionalidades iriam recorrer a biblioteca *freetype*, mas devido a dificuldades técnicas no *setup* não foi possível, e uma outra funcionalidade final: manter *highscores* ao longo de varias execuções do programa.

3.2 Considerações Finais

Concluímos aqui o relatório do projeto entregue, aproveitamos este momento para deixar umas palavras finais e opiniões pessoais relativamente a este trabalho:
Acho que podemos afirmar que a maior dificuldade deste projeto não foi a sua própria resolução/programação gráfica mas sim a sua resolução logica, isto devido ao acesso e resolução dos vários exercícios propostos ao longo do semestre pelo professor, foi também de maior dificuldade a implementação de funcionalidades/tecnologias não trabalhadas e/ou desenvolvidas nas aulas (*menu*, utilização da biblioteca *freetype*);
Podemos também afirmar que por razoes externas a própria cadeira e características incomuns da própria funcionalidade do semestre levou a uma maior indisponibilidade de tempo que dificultou a resolução atempada e satisfatória do projeto;

3.3 Referências Bibliográficas

Aulas teóricas e laboratórios práticos de CG - http://www.di.ubi.pt/~agomes/cg/

Documentação do learnopengl - https://learnopengl.com/

Stackoverflow - https://stackoverflow.com/