Universidade Federal do Rio Grande do Norte
Escola de Ciências e Tecnologia

Jefferson Denilson da Silva Matias Xavier

Desenvolvimento de Tetris em C++

Natal – RN

Outubro – 2017

Jefferson Denilson da Silva Matias Xavier

Desenvolvimento de Tetris em C++
Relatório apresentado à Universidade Federal
Do Rio Grande do Norte, na disciplina de Linguagem de Programação,
como requisito para avaliação na 3ª unidade,
solicitado pelo professor Diego R. C. Silva.

Natal – RN

Outubro – 2017

SUMÁRIO

1 Introdução

O objetivo deste projeto é desenvolver um jogo semelhante ao Tetris. O programa é capaz de introduzir, mover e rotacionar peças em um cenário, detectar colisão com peças estacionárias e com o cenário, eliminar linhas que estejam completamente preenchidas com as peças estacionárias e verificar se é possível introduzir novas peças. Também foram incluídas funcionalidades como pausa, pré-visualização e troca de peças, reinicio, projeção da posição final da peça, pontuação e alteração da dificuldade do jogo.

1.1 Equipe de desenvolvimento

Nome: Jefferson Denilson da Silva Matias Xavier
Matrícula: 20160107264
Turma/subturma: 1A

2 Desenvolvimento

2.1 Processo geral

Esta seção resume o funcionamento do programa. Primeiramente é iniciada uma repetição while usando window.isOpen como condição, dessa forma todos os processos serão executados enquanto a janela estiver aberta. Em seguida é definida a peça atual e a próxima, assim como a posição inicial. Também são atualizadas as variáveis de texto que mostrarão o nível, objetivo e pontuação. Então define-se a variável desce como true, que será usada para manter a próxima repetição enquanto ainda for possível mover a peça, que descerá com o tempo ou quando solicitado pelo jogador. Nessa repetição também serão desenhadas as peças e as informações do programa. Quando a peça não puder mais se mover, ela será armazenada em uma matriz e será verificado se é possível eliminar linhas, redefinir o nível de dificuldade e introduzir novas peças. Caso não seja possível introduzir novas peças, se chega ao fim do jogo, tendo-se as opções de fechar o programa ou começar do início. A seguir será descrito como as funções funcionam.

2.2 Definição do cenário

Inicialmente é necessário definir o cenário onde serão colocadas as peças, para isso foi criada a matriz cena de 24x12. Para construção dos elementos presentes no cenário inicial foi usada a função desenhacena, que executa 3 repetições. A primeira repetição define as bordas laterais, atribuindo o número 8 a primeira e a última coluna da cena. A próxima repetição preenche a última linha da mesma forma, estabelecendo a borda inferior da matriz. Por último, todos os elementos restantes são preenchidos com 0. Notou-se que apesar de a borda não ser utilizada na interface gráfica, ela se mostrou útil para evitar que as peças fossem desenhadas fora do cenário. As três primeiras linhas da matriz também não aparecem na interface gráfica, para que se tenha o efeito de mostrar apenas parte da peça na tela de game over e para que a peça possa ser rotacionada na primeira posição. Dessa forma também, ao se introduzir uma nova peça só uma parte será vista e conforme a peça for se movendo, ela aparecerá por completo.

2.3 Peças

Para a construção das peças foi utilizado o arquivo pecas.h, que contém uma matriz 7x4x4x4, sendo a primeira dimensão o número de peças diferentes, a segunda as 4 rotações de cada peça, e as duas últimas definem uma matriz 4x4 (sendo 4 a maior altura e a maior largura atingida dentre todas as peças), onde cada peça é desenhada em cada rotação. Para que seja escolhido o tipo de peça, inicialmente é definida aleatoriamente uma variável chamada proximo, usando a função rand, em seguida sempre que for selecionada uma nova peça, o tipo dela será atribuído com o valor do próximo e o próximo será redefinido. Dessa forma pode-se definir qual peça virá depois da atual. As peças também serão selecionadas inicialmente na primeira rotação.

A função desenhapeca é responsável por inserir os elementos da peça dentro da matriz cena. Para desenhar são utilizadas duas repetições para percorrer a cena e a matriz da peça, conferindo se um dado elemento é diferente de 0, se for, o elemento da cena é igual ao elemento da peça e é acrescentado um contador (contbloco), pois é conhecido que cada peça tem no máximo 4 blocos, sendo utilizado como condição. A variável que define a linha inicial é usada como índice inicial para o laço externo, da mesma forma que o índice do laço interno é igual à coluna inicial, sendo acrescentada a condição de enquanto o índice for menor que a coluna inicial mais 4, mas no laço externo só é necessário verificar se os 4 blocos foram desenhados. Dessa forma define-se uma área de 4x4 na matriz cena onde a peça será desenhada.

2.4 Colisão

Para realizar um movimento, antes é necessário saber se ele é possível, ou seja, se a posição para onde se pretende mover não está ocupada. A função simularmov tem funcionamento semelhante a desenhapeca, mas apenas verifica se em uma certa área o elemento da peça ocuparia um espaço não livre. Caso ocupe, a função retorna falso, não sendo permitido o movimento, mas caso passe pelas repetições sem retornar falso, é retornado verdadeiro e o movimento é permitido.

Também é necessário verificar se uma rotação é possível em dada posição, mas pode ser preciso realizar um deslocamento para que a peça exista. Esse problema é verificado, por exemplo, ao encostar uma peça L contra a borda ou contra outras peças, pois como o número de colunas ocupadas varia de 2 para 3, o movimento não seria possível com uma simples verificação de colisão. Portanto, foi feita a função simularot, que chama a função simularmov em diferentes posições para a próxima rotação, alterando as variáveis de rotação, linha e coluna inicial, se for possível o movimento. A função começa definindo qual seria a próxima rotação da peça, sendo que se a rotação atual for 3, a próxima será 0, do contrário a rotação seria acrescentada em 1. Para que não efetue deslocamentos desnecessariamente, a primeira seleção confere se a próxima rotação é possível naquele espaço. Se não for, as seleções seguintes simulam a rotação da peça deslocada em diferentes direções. Se a rotação não for possível nas outras seleções, as últimas verificam se ela é a peça I e simulam um deslocamento de duas unidades, necessário em algumas situações dessa peça.

2.5 Peça fantasma

Para facilitar a visualização da posição final da peça, será desenhada uma projeção na posição mais baixa que ela pode existir, chamada peça fantasma. A função posfantasma define qual a linha inicial para essa peça, verificando se a peça pode existir abaixo da posição atual, com a função simularmov, sucessivas vezes. Quando a função simularmov retornar falso, a função posfantasma retornará a última alteração na variável de linha inicial, que será utilizada para que se desenhe a peça fantasma.

2.6 Eventos e Movimentação

Para os movimentos laterais foi usada a detecção de eventos do SFML, dentro de um laço while as seleções identificam qual foi o tipo de evento e executam o código. A primeira seleção serve para que a janela seja fechada clicando no botão de fechar ou pressionando escape. A segunda identifica se o tipo de evento foi uma tecla pressionada e em seguida verifica qual delas foi.

Para a função movimento ser executada primeiro deve-se verificar se o programa não está pausado ou em game over. Caso a tecla pressionada seja a tecla esquerda ou direita, também será chamada a função simularmov, verificando se a posição desejada está livre, permitindo então a alteração da variável de coluna inicial da peça. Com o pressionamento da seta para cima é executada a função simularot, exceto para a peça O, que não rotaciona.

Nessa função também é feita a troca de peças, que consiste em trocar o tipo da peça atual com a que está na variável manter, ao se pressionar o shift direito. A variável manter é iniciada com o número -1, de forma que inicialmente não corresponda a nenhum tipo de peça, portanto ao ser identificado esse valor, manter terá o valor do próximo e o próximo será redefinido. Em seguida será feita a troca da variável da peça mantida com o tipo de peça atual e tendo um valor diferente de -1 em manter, essa troca será feita com o número que ficou armazenado. As variáveis de linha e coluna inicial e variável de rotação também serão redefinidas, para que a peça sempre surja na posição e rotação inicial. Para que o movimento de troca seja executado apenas uma vez a cada nova peça, é criado um contador com o valor 0 (contmanter) ao ser selecionada a peça, que será adicionado quando a troca for executada. Sendo assim, ao ser pressionado o shift direito também será verificado se o contador é igual a 0, significando que a peça ainda não foi trocada durante o processo de movimento.

Para a pausa, deve ser identificada a tecla enter e verificado se o programa não está em game over. Em seguida será alterada a variável pausa, entre os valores verdadeiro e falso, de forma que seu valor seja sempre o contrário do atual. Caso a pausa seja verdadeira, será desenhada

a imagem pausa.png, que contém a mensagem de pausa e as instruções. Para que a pausa seja efetiva, sempre deve ser conferido o valor da pausa antes de executar movimentos e desenhar.

Por último, a seção de eventos de teclado identifica se foi pressionada a barra de espaço, que redefinirá todas as variáveis necessárias para que o programa volte ao estado inicial, tal como as variáveis de continuar, pontuação, peças e tempo de espera, além de chamar a função desenhacena novamente.

Para que a peça desça, são usadas as variáveis de tempo espera e passado. A espera define o tempo máximo antes que a peça desça, que inicialmente é de um segundo, e o passado recebe o tempo que passou desde que a variável cronometro foi reiniciada. Caso o tempo passado seja maior que a espera, o cronometro é reiniciado e a variável desce recebe o valor de simularmov, descendo a peça em uma linha se for verdadeiro ou permitindo que a peça pare caso contrário. A peça também desce caso seja pressionada a seta para baixo, dando o efeito de jogar a peça para baixo mais rápido. Usando-se Keyboard::isKeyPressed se tem uma detecção mais contínua da tecla pressionada (quando comparado a Event::KeyPressed), não sendo adequada para movimentos mais leves como os horizontais e a rotação. Ao ser utilizada a seta para baixo também serão atribuídos pontos por derrubar a peça mais cedo e já que a tecla é identificada múltiplas vezes, a quantidade total de pontos varia com a altura restante para que a peça pare.

2.7 Quebra de linha

Ao se completar uma linha com peças estacionárias é necessário apagar essa linha e trazer as peças acima para baixo. Para isso a função quebralinha recebe o último valor da linha da peça estacionada e verifica se essa e as próximas três linhas da matriz cena estão completas, já que o número máximo de linhas que podem ser eliminadas de uma vez é quatro. Para que não sejam conferidas as bordas da matriz, o número máximo da linha conferida é até 22 e as colunas vão de 1 a 10. Antes de se conferir cada elemento das colunas em uma mesma linha, é criada a variável lcompleta com o valor true e caso algum elemento da linha for igual a 0 (espaço livre), a variável é atribuída com o valor falso, significando que a linha não está preenchida com blocos. Caso essa variável não seja alterada na repetição interna, será chamada a função apagar, que apagará todos os blocos na linha e em seguida será chamada a função baixarlinha, que parte da linha acima da que foi apagada e traz todos os elementos diferentes de 0 para a linha abaixo. Essa função continua até conferir a primeira linha ou até ser encontrada uma linha vazia, já que se ela partiu de cima da linha apagada, não há elementos acima da próxima linha vazia. Após esse processo, é adicionada uma variável que contará quantas linhas foram eliminadas, que será retornada ao final da função quebralinha. Para que se veja melhor a diferença entre antes e depois das linhas quebradas, é usada a função sleep, que pausará a execução por 300 milissegundos.

2.8 Pontuação e Renivelamento

A função renivelar tem o propósito de aumentar a dificuldade conforme o nível do jogo e recalcular os pontos. Após ser executada a quebra de linha, a quantidade de linhas apagadas será subtraída da variável objetivo (quantidade de linhas para que se passe de nível), que começará com o valor 5. Depois, é verificado se o valor do objetivo é igual ou menor que 0, podendo ser alterado o nível em uma unidade. O objetivo é então recalculado sendo o nível vezes 5 mais o valor atual do objetivo, pois dessa forma se o objetivo tiver um valor negativo, as linhas quebradas serão descontadas no próximo objetivo. A dificuldade aumenta diminuindo-se o tempo de espera em que a peça cai, dividindo-se a variável regulador por 1.4 e atribuindo-a como os milissegundos da espera. O regulador só será alterado se for maior que 40, mantendo o tempo de queda constante a partir do nível 11. Por fim, a função renivelar recalcula os pontos somando o valor anterior com o valor das linhas quebradas vezes ele mesmo vezes 100, desse modo mais linhas quebradas de uma vez darão mais pontos.

2.9 Game over

Para que seja verificado se o jogo pode continuar, a variável continuar recebe o valor da função simularmov, passando como parâmetro a próxima peça na rotação inicial e a posição inicial das peças (linha 1, coluna 4). Caso o resultado seja falso, é desenhada a imagem gameover.png e devido as seleções antes de se executar um movimento ou pausa e antes de ser feito o desenho na tela, que conferem o valor de continuar, nada poderá ser executado exceto o fechamento do programa ou o reinicio.

2.10 Interface gráfica

A interface gráfica é constituída por uma janela chamada window, com 300 de largura por 400 de altura, onde serão desenhadas as peças e as informações do jogo.

Antes de desenhar imagens na janela, elas devem ser carregadas em variáveis do tipo Texture, que serão colocadas em variáveis do tipo Sprite. Nas imagens que contém os blocos (tiles20.png) e as peças que serão mostradas como próximo e manter (preview.png), é necessário definir uma seção da imagem de acordo com o que se quer desenhar. No caso das peças, cria-se um retângulo de 20x20 pixels partindo de uma coordenada do canto superior esquerdo da imagem, sendo definida de acordo com o número correspondente a peça. Dessa forma, para se desenhar uma peça estacionária, multiplica-se o número encontrado na cena por 20, pois a coordenada para cada bloco na imagem varia de 20 em 20. Para a peça em movimento multiplica-se o tipo da peça mais 1, já que o tipo de peça na matriz pecas varia de 0 a 6 e o número em que é definida a peça varia de 1 a 7. O desenho de próximo e manter segue o mesmo princípio, mas a coordenada da peça na imagem varia de 80 em 80 e a parte de manter só será selecionada se for diferente de -1. Também foram criados sprites para a moldura, que contém as informações de texto não variáveis, e para as imagens de pausa e game over. Já a peça fantasma é definida por um retângulo de 20x20.

As informações variáveis como nível, objetivo e pontos são mostradas com uma variável do tipo Text, utilizando a fonte do arquivo Cantarell-Bold.otf. Como as variáveis passadas para o texto devem ser strings, é utilizado sprintf de forma a armazenar cada informação numérica em um vetor do tipo char.

Para que os elementos possam ser desenhados na janela, são utilizadas as funções clear, draw e display do SFML. A função clear apaga tudo que está desenhado até o momento, podendo ser usada uma cor para limpar a tela. Em seguida a função draw é utilizada para desenhar a moldura e o texto. As peças de próximo e manter são desenhadas pela função desenharpreview, da forma descrita anteriormente. A função pecaativa desenhará a peça que está em movimento e o fantasma da peça, seguindo um princípio semelhante ao da função desenhapeca, sendo a posição do bloco definida pelos índices das repetições vezes 20. Como o espaço da tela em que serão desenhadas as peças não inclui as bordas nem as três primeiras linhas da matriz cena, é descontada uma unidade da largura e três da altura ao se definir a posição do sprite. Para desenhar as peças que estão paradas é chamada a função cenario, que confere quais elementos da matriz cena são diferentes de 0 e desenha a seção da imagem correspondente. A função para de procurar por elementos quando for encontrada uma linha vazia, já que não seria possível haver mais blocos após esse ponto. Ao fim do processo de desenho, é chamada função display, que termina o frame atual, mostrando todas as peças presentes naquele momento.

3 Cronograma

A tabela a seguir descreve as atividades realizadas durante o desenvolvimento:

Data	Descrição
30/09 – 01/09	Identificação das dificuldades e funcionalidades do projeto, ferramentas a serem utilizadas, opções de projetos. Desenvolvimento do cenário.
07/10 – 08/10	Peças, desenho e movimentos.
09/10 – 11/10	Ideias sobre algoritmos de colisão, quebra de linha e trazer peças abaixo.
12/10	Elaboração do relatório.
13/10	Estudo sobre interface gráfica, colisão e quebra de linha
14/10 – 15/10	Elaboração de interface gráfica e colisão
16/10 – 19/10	Atualização do relatório
21/10 – 27/10	Quebra de linha, game over e alteração de tempo.
28/10 – 29/10	Elaboração de preview, troca de peças e informações na tela.
30/10 – 01/11	Término do relatório

Tabela 1 – Cronograma de desenvolvimento

4 Conclusão

Sobre as dificuldades encontradas, viu-se que este é um projeto relativamente complexo, sendo difícil se basear em códigos prontos, principalmente a parte que envolve conteúdos de fora da disciplina (por exemplo, uso de classes e interface gráfica). Optou-se principalmente por recorrer a tutoriais que enfatizem a lógica do programa e adaptar o que for necessário, ou recorrer a ideias que surgiram durante o processo de desenvolvimento. Também é um projeto que demanda um certo tempo, ficando um pouco difícil desenvolvê-lo durante a semana e conciliar com outras atividades. No entanto, foi interessante pensar em formas de desenvolver um programa partindo da lógica e descobrir novas funções para os programas como inserção de tempo, entrada de teclado e interface gráfica.

5 Referências

LÓPEZ, Javier. **Tetris tutorial in C++ platform independent focused in game logic for beginners**. Disponível em: <http://javilop.com/gamedev/tetris-tutorial-in-c-platform-independent-focused-in-game-logic-for-beginners/>. Acesso em: 22 set. 2017.

HÜPNER, Thiago Henrique. **Tutorial SFML**. Disponível em: <https://www.vivaolinux.com.br/artigo/Tutorial-SFML>. Acesso em: 11 out. 2017.

TUTORIALS for SFML 2.4. Disponível em: <https://www.sfml-dev.org/tutorials/2.4/>. Acesso em: 29 set. 2017.

WILLIAMS, Michael James. **Implementing Tetris: Collision Detection**. Disponível em: <https://gamedevelopment.tutsplus.com/tutorials/implementing-tetris-collision-detection--gamedev-852>. Acesso em: 13 out. 2017.

WILLIAMS, Michael James. **Implementing Tetris: Clearing Lines**. Disponível em: <https://gamedevelopment.tutsplus.com/tutorials/implementing-tetris-clearing-lines--gamedev-1197>. Acesso em: 13 out. 2017.

FAMTRINLI. **Let's make 16 games in C++: TETRIS**. Disponível em: <https://www.youtube.com/watch?v=zH_omFPqMO4&t=4s>. Acesso em: 13 out. 2017.