

Fonte: Wikipedia

Um jogo simples que fez muito sucesso (e continua fazendo) é o Tetris (<https://en.wikipedia.org/wiki/Tetris>). Esse jogo foi criado nos anos 80 por Alexey Pajitnov e rapidamente se espalhou pelo mundo em computadores, videogames, aparelhos portáteis, etc.

Neste trabalho iremos explorar algumas ideias do jogo Tetris para praticar alocação dinâmica de memória. Mais especificamente, criaremos uma estrutura de dados dinâmica capaz de representar um estado (uma “tela”) do jogo.

Leia este roteiro com bastante atenção antes de começar sua implementação. Antes de submeter a versão final do seu trabalho leia este roteiro novamente e confira se sua implementação seguiu toda a especificação.

Aviso: apesar desta especificação ter ficado grande (devido aos detalhes), sua implementação do trabalho provavelmente ficará pequena! Ele é mais simples do que aparenta.

Aviso 2: comece sua implementação com antecedência! O deadline dos trabalhos é firme.

Observação: como um dos objetivos é praticar o uso de alocação de memórias e ponteiros, seu programa deverá utilizar alocação dinâmica de memória de forma explícita (utilizando os operadores *new* e *delete* -- i.e., você não pode utilizar classes prontas (como MyVec) para armazenar os dados na versão final do seu trabalho).

Implementacao

Você deverá implementar uma classe chamada Tetris. Crie um arquivo chamado Tetris.h (contendo a interface da sua classe) e outro chamado Tetris.cpp (contendo a implementacao).

Sua classe representará um estado da “tela” do jogo Tetris. Cada “pixel” da tela terá um caractere representando ou um espaço não utilizado ou uma letra maiúscula (a letra indicaria que na posição em questão há um pedaço da peça representada pela letra).

Sua classe deverá ter **pelo menos** as seguintes funcoes (você certamente precisará criar **outras funções** necessárias para o bom funcionamento da classe Tetris):

Construtor com um argumento: o argumento do construtor será um inteiro indicando a largura (numero de colunas) do jogo Tetris que será representado.

Método get(int c,int l): recebe dois argumentos: a coluna (primeiro argumento) e a linha (segundo argumento) de um pixel e retorna um caractere que representa o estado de tal pixel no jogo atual.

Método removeColuna(int c): dado o índice c de uma coluna ($0 \leq c < \text{número de colunas}$), remove a coluna do jogo (diminuindo, assim, sua largura).

Método removeLinhasCompletas: remove todos os pixels do jogo que estiverem em linhas completas (linhas que não contém espaço em branco). Ao remover uma linha completa os pixels acima de tal linha são “deslocados para baixo”.

Método getNumColunas: retorna o número de colunas (largura) do jogo Tetris. Esse número deverá ser igual ao valor passado pelo construtor do jogo (a não ser que algumas colunas tenham sido removidas utilizando o método removeColuna).

Método getAltura(int c): retorna a altura da coluna c do jogo. A altura de uma coluna é igual a altura do pixel (não vazio) mais alto da coluna. Uma coluna onde todos os pixels são vazios possui altura 0.

Método getAltura: retorna a altura maxima do jogo atual.







Método adicionaForma(int coluna,int linha,char id, int rotacao): método mais importante da sua classe. Recebe como argumentos: coluna (um inteiro), linha (inteiro), o id de uma peca (um caractere) e uma rotacao (um inteiro que pode valer 0, 90, 180 ou 270).

No jogo de Tetris ha 7 possiveis pecas (cada uma representada por um caractere distinto). Veja a tabela abaixo. Os pixels de uma peca sao representados por caracteres maiusculos.

Essa funcao devera tentar adicionar a peca a tela atual, sendo que o pixel superior esquerdo da peca (note que esse pixel e' vazio, por exemplo, para a peca 'S' sem rotacoes) devera ser adicionado a coluna e linha passados como argumento para a funcao.

Peca I: I I I I	Peca J: JJJJ J	Peca L: LLLL L	Peca O: OO OO
Peca S: SS SS	Peca T: TTT T	Peca Z: ZZ ZZ	

Se a rotacao não for 0, a peca devera ser rotacionada em sentido horario considerando o angulo (em graus) de rotacao. Quando uma peca e' rotacionada ela e' deslocada para a esquerda e para cima para evitar que haja linhas ou colunas com pixels vazios na representacao dessa peca. Veja o exemplo abaixo (no exemplo com rotacao realizada de forma errada a peca deveria ter sido deslocada o mais a esquerda possivel)

Rotacao 0: TTT T	Rotacao 90 (certo):  T  TT  T	Rotacao 90 (errado)  T  TT  T
------------------------	---	---

Esse método deve retornar false se a peca não puder ser adicionada ao jogo (e true caso contrario). Uma peca não pode ser adicionada ao jogo caso pelo menos um de seus pixels (não vazios) “colidir” com um pixel ja no jogo ou quando pelo menos um dos pixels (não vazios) estiver fora da area da tela. Se não for possivel adicionar uma peca o estado da tela atual não devera ser modificado.

Detalhes de implementacao

Sua classe devera possuir um array dinamico de arrays dinamicos de apontadores (o nome desse array **deverá** ser *jogo*) representando a tela do jogo. O conteudo *jogo[c][l]* devera ser o caractere representando o pixel da peca na coluna c (comecando de 0) e linha l (tambem comecando a partir de 0) do jogo. Tal caractere devera ser um espaco em branco caso o pixel esteja vazio ou uma letra maiuscula representando a peca que está no local em questao.

Se uma peca for adicionada a uma coordenada da tela cada uma das colunas afetadas devera crescer de modo a armazenar a peca. Observe que todas as colunas deverao ser “justas” (você não deve armazenar pixels vazios após o ultimo pixel não vazio).

Alem do array “jogo”, você também precisara de um inteiro representando o numero de colunas do jogo e de um array de inteiros alturas. O valor de alturas[c] sera a altura da coluna c no jogo (i.e., alturas[c] e’ o número de linhas no array jogo[c]). Você deverá seguir exatamente essa estrutura (incluindo o nome das variáveis: jogo, alturas e largura).

Exemplos

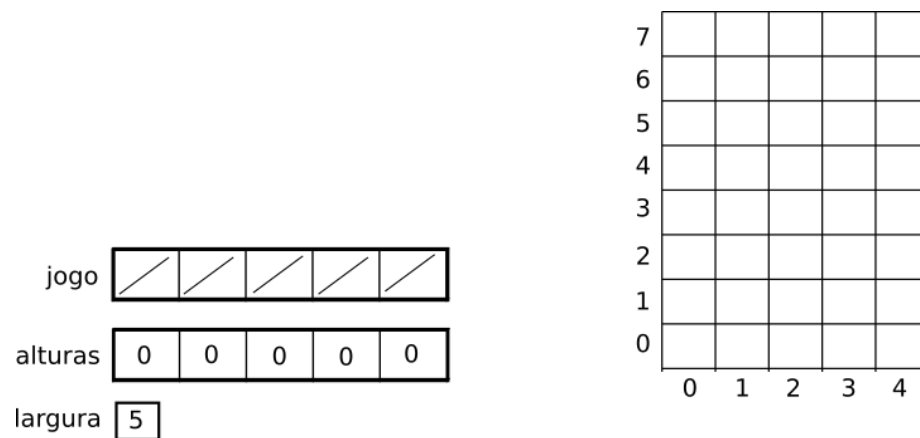
Jogo game(5); //cria um jogo com 5 colunas

cout << game.getAltura() << endl; //deve imprimir 0

cout << game.getAltura(1) << endl; //a altura da coluna 1 e’ 0

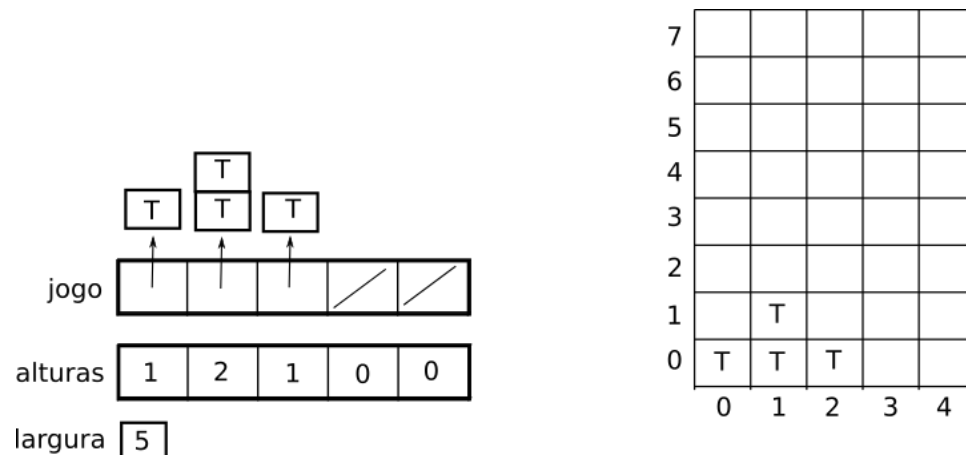
cout << game.get(0,0) << endl; //conforme pode ser visto na figura a direita (abaixo), o pixel 0,0 está vazio (deve retornar o caractere espaco em branco)

Seu objeto deverá criar os arrays conforme ilustrado na figura a esquerda. A figura a direita exibe o estado da tela.



game.adicionaPeca(0,1,'T',180);

O comando acima adicionara a peca “T”, sendo que o canto superior esquerdo dela (após a rotacao de 180 graus) estara nas coordenadas 0 (coluna), 1 (linha). Veja a ilustracao abaixo.



```
game.adicionaPeca(0,1,'I',90);
```

O código acima não deve fazer nada (e retornar *false*)! A peça I não pode ser adicionada visto que ela colidiria com o pixel 'T' na coordenada 1,1.

```
game.adicionaPeca(1,4,'S',0);
```

```
game.adicionaPeca(3,1,'O',0);
```

```
cout << game.get(1,2) << endl; //imprime espaço em branco
```

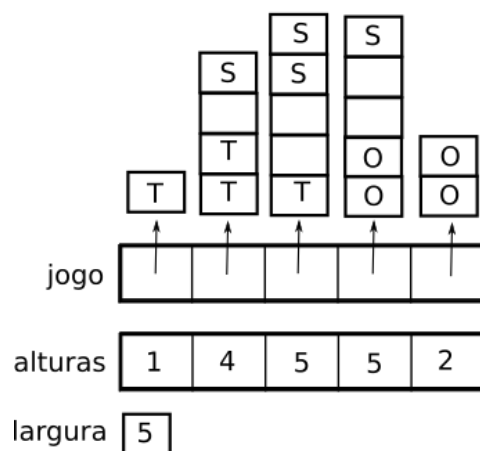
```
cout << game.get(0,0) << endl; //imprime T
```

```
cout << game.get(1,1) << endl; //imprime T
```

```
cout << game.get(4,1) << endl; //imprime O
```

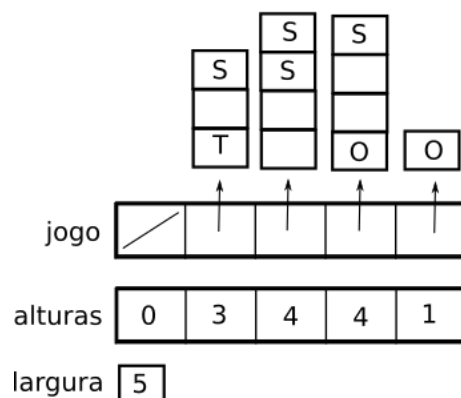
```
cout << game.get(3,4) << endl; //imprime S
```

Veja o novo estado do jogo abaixo (note que nossa classe não “simula uma gravidade”). Seu programa deverá realocar os arrays de algumas colunas (por exemplo, da coluna 2)



7					
6					
5					
4			S	S	
3		S	S		
2					
1		T		O	O
0	T	T	T	O	O
	0	1	2	3	4

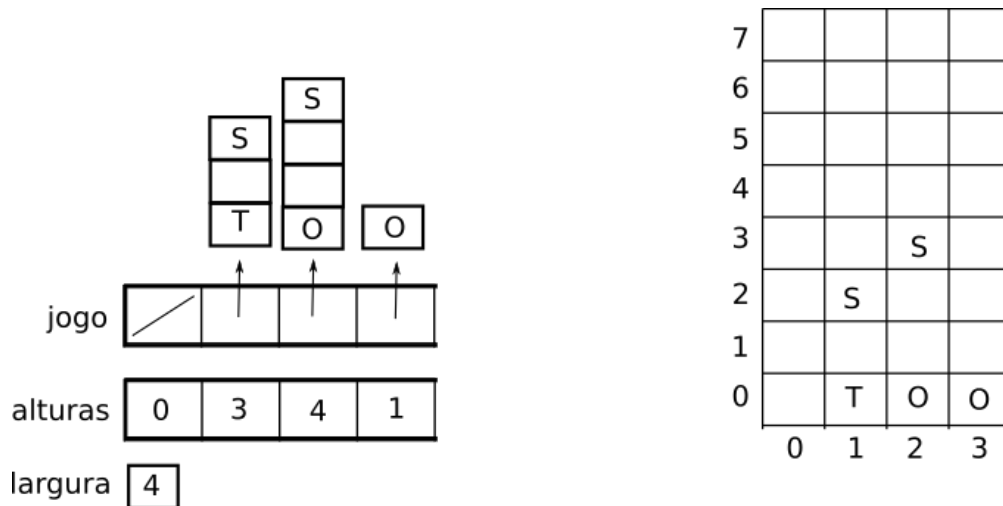
Note que a linha 0 está completa e, assim, se chamarmos o método `removeLinhasCompletas()` o estado do jogo ficara (note que todos arrays são realocados!):



7					
6					
5					
4					
3			S	S	
2		S	S		
1					
0		T		O	O
	0	1	2	3	4

Finalmente, se o código abaixo for chamado:
`game.removeColuna(2);`

O estado do jogo deverá ser (observe que a coluna 2 foi deletada e, com isso, os arrays `jogo` e `alturas` foram realocados dinamicamente com tamanho menor):



Observe que provavelmente há outras formas mais eficientes de se implementar a classe Tetris (por exemplo, manter todas as colunas “justas” exige muita alocação e desalocação de memória!). Porém, siga essa implementação ineficiente (o objetivo é praticar alocação dinâmica de memória!).

Jogo utilizando sua classe

Uma forma de testar sua classe é compilando o jogo Tetris disponível em:

https://drive.google.com/open?id=1YdY05NA7lwbQ2r_ycAJhNHwkvuVeWAmV

Tal implementação foi feita de forma bem simples (e provavelmente ineficiente) e utiliza a biblioteca `ncurses`. Para instalar a biblioteca `ncurses` no Linux Mint/Ubuntu/Debian, tente o seguinte comando no terminal: “`sudo apt-get install libncurses-dev`” (esse comando pode variar com base na sua distribuição).

Para compilar o programa exemplo, digite: “`g++ Tetris.cpp jogoTetris.cpp -lncurses`” (esse comando supõe que a biblioteca `ncurses` esteja instalada e que `Tetris.cpp`, `Tetris.h` e `jogoTetris.cpp` estejam no diretório atual).

Note que esse é apenas um exemplo simples de interface para o jogo Tetris. É possível criar versões bem melhores do jogo (por exemplo, usando uma interface gráfica ou mesmo utilizando uma interface de texto com cores na biblioteca `ncurses`). Como exercício extra (seu trabalho poderá ganhar um bônus de até 20% em pontos extras!), tente criar um jogo com interface (ASCII) melhor de Tetris utilizando sua classe e a biblioteca `ncurses`. Submeta o

arquivo do seu jogo melhorado junto com o trabalho e mencione sua implementacao (incluindo instrucoes de compilacao) no README.

Obs: seu programa sera testado pelo Submittty considerando não apenas a implementacao de jogoTetris.cpp (por exemplo, essa implementacao não testa a remocao de colunas).

Dica

Implemente todo seu trabalho no Linux. Ele sera avaliado utilizando o compilador g++ em um servidor com Linux e, dessa forma, se você implementa-lo em Windows (com outro compilador) ha um risco dele funcionar de forma um pouco diferente no sistema Submittty.

Embora seu trabalho deva seguir a especificacao descrita acima, criar uma versao inicial da classe Tetris que não segue tudo que foi descrito pode ser uma boa forma de praticar a implementacao e se acostumar com os principais conceitos (por exemplo, sua primeira versao da classe Tetris pode tentar utilizar uma matriz `MyVec<MyVec<char> >` para representar o estado do jogo e, com isso, você não precisara se preocupar com alocao dinamica de memoria inicialmente).

Conforme falado anteriormente, a eficiencia da nossa implementacao não sera muito importante neste trabalho (estamos apenas praticando o uso de ponteiros e alocao dinamica de memoria). Porem, boas praticas de engenharia de software serao avaliadas.

Por fim:

---> Desenhe vários diagramas para entender o que está implementando <<----

Testando sua classe

Tente criar casos de teste para avaliar a sua classe! Não a teste apenas com o arquivo jogoTetris.cpp.

Arquivo README

Seu trabalho devera incluir um arquivo README.

Tal arquivo contera:

- Seu nome/matricula
- Informacoes sobre fontes de consulta utilizadas no trabalho

Submissao

Submeta seu trabalho utilizando o sistema Submittty ate a data limite. Seu programa sera avaliado de forma automatica (os resultados precisam estar corretos, o programa não pode ter

erros de memoria, etc), passara por testes automaticos “escondidos” e a qualidade do seu codigo sera avaliada de forma manual.

Você devera enviar os arquivos: README, Tetris.cpp e Tetris.h

Duvidas

Dúvidas sobre este trabalho deverão ser postadas no PVANET Moodle. Se esforce para implementá-lo e não hesite em postar suas dúvidas!

Avaliacao manual

Principais itens que serao avaliados (alem dos avaliados nos testes automaticos):

- Comentarios
- Indentacao
- Nomes adequados para variaveis
- Separacao do codigo em funcoes logicas
- Uso correto de const/referencia
- Uso de variavies globais apenas quando absolutamente necessario e justificavel (uso de variaveis globais, em geral, e' uma ma pratica de programacao)
- Etc

Regras sobre plágio e trabalho em equipe

Leia as regras gerais aqui:

https://docs.google.com/document/d/1qwuZtdioZO-QiDsq6SAm7m-DU6bLrid7_7nEL4g9HOk/edit?usp=sharing