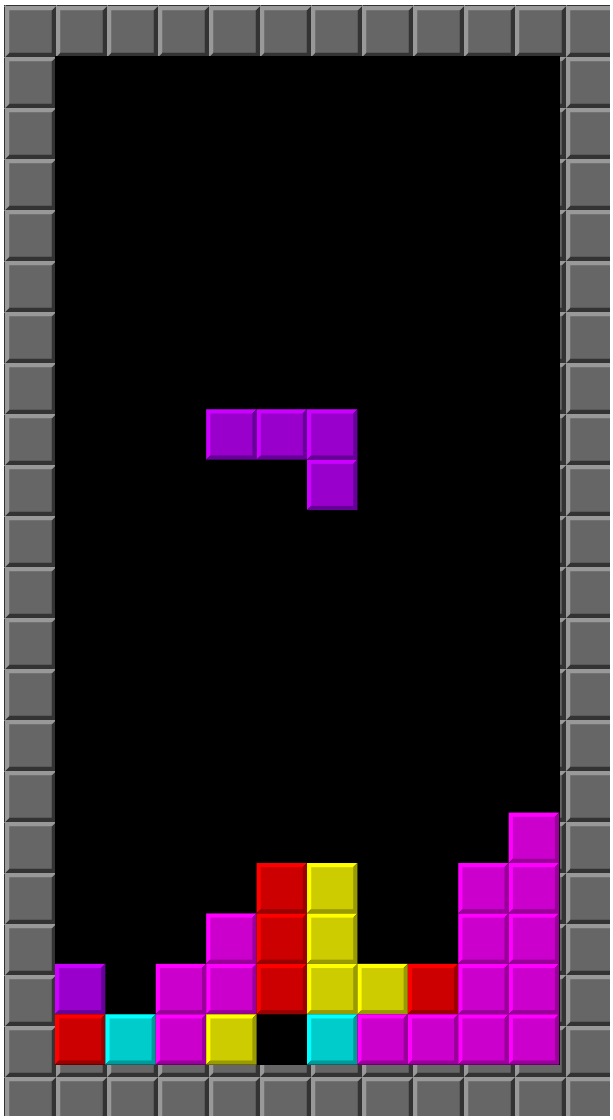
BCM0505-15 - Processamento da Informação
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Profa. Carla Negri Lintzmayer

Projeto Pentris

1 Introdução

Neste projeto você desenvolverá uma versão do famoso jogo **Tetris**. Desenvolvido em 1984, um de seus criadores era um aluno de 16 anos[1].

O jogo funciona da seguinte forma. Um *tetraminó* é uma peça geométrica composta de quatro quadrados cada. Uma sequência aleatória de tetraminós desce do topo da tela e o jogador pode manipulá-los movendo para os lados ou rotacionando-os, na intenção de completar linhas horizontais completas por pedaços de tetraminós. Quando uma linha se completa, ela se desintegra, todas as linhas superiores descem e o jogador ganha pontos.



As peças denominadas de A a G na Figura 1 são os 7 tetraminós "originais". Apenas o tetraminó G tem capacidade de completar quatro linhas simultaneamente, e isso é conhecido como "tetris".

[1] Quer dizer, você também é capaz.

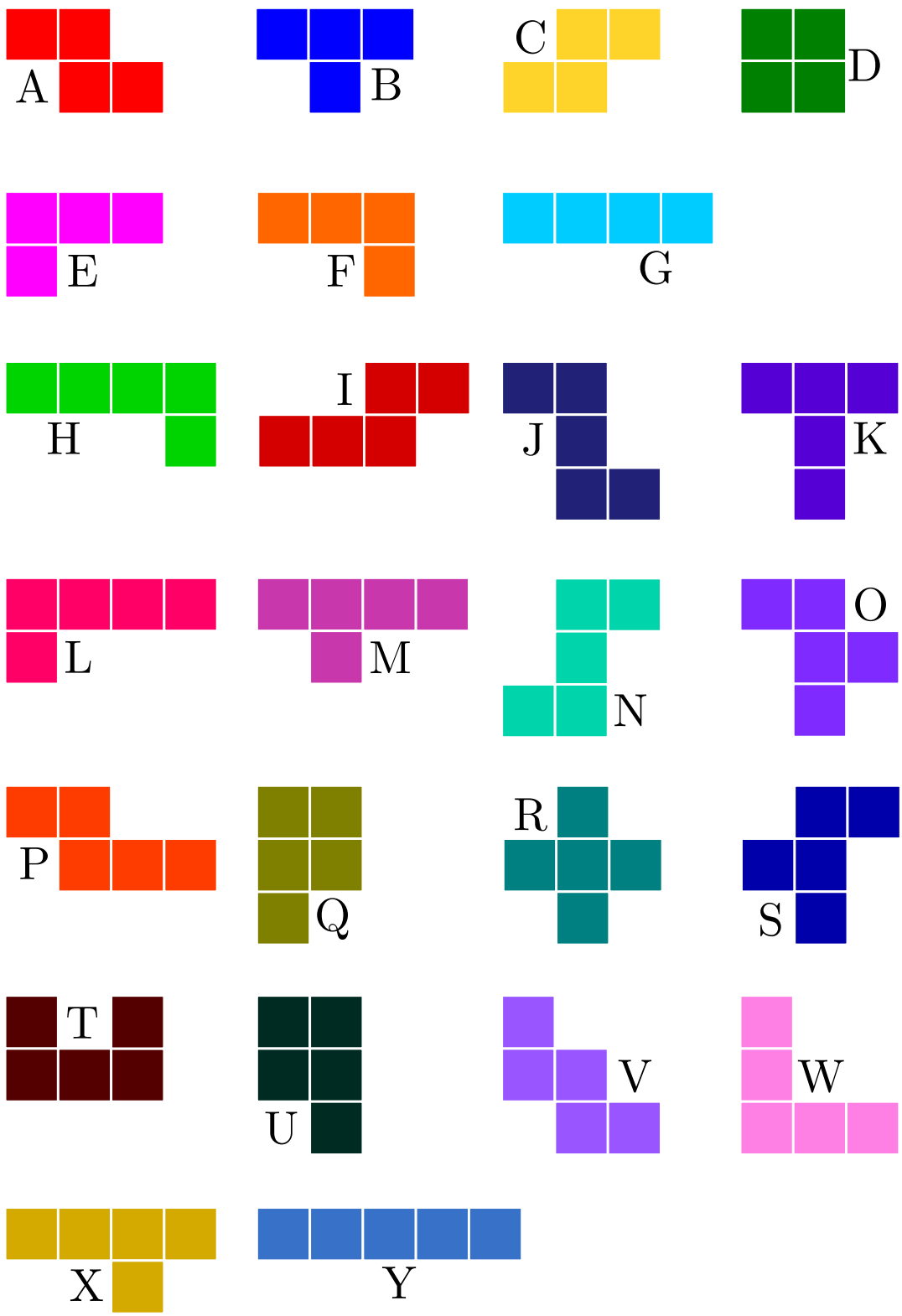


Figura 1: Peças disponíveis.

2 Projeto Pentris

Você desenvolverá uma versão do jogo onde as peças podem ser tetraminós ou pentaminós (peças com cinco quadrados cada): daí o nome Pentris. Mesmo assim, seu jogo terá 7 peças. Você deve escolher 6 peças dentre as disponíveis na Figura 1, sendo que no máximo uma dessas pode ser invariante à rotação (isto é, no máximo uma dentre as peças D e R pode ser escolhida). A sétima peça deverá ser **criada** por você, podendo ser um tetraminó ou um pentaminó, e **não pode** ser invariante à rotação. Daqui para frente, essa peça criada por você será chamada de **bomba**.

Você terá acesso a um arquivo "Pentris.java" com parte do jogo já implementada: a interface gráfica e a leitura do teclado. Seu trabalho é desenvolver as funções específicas que serão chamadas durante o jogo. Você não precisa compreender detalhes do código fornecido para isso[2].

A tela de jogo será representada pela matriz `Color[][] tela`, de dimensão `LARGURA` por `ALTURA`. Assim, `tela[x][y]` indica o quadrado que está na coluna `x` e linha `y`. Os valores das variáveis do tipo `Color` podem ser os pré-definidos `Color.BLACK`, `Color.GRAY`, `Color.CYAN`, `Color.BLUE`, `Color.ORANGE`, `Color.YELLOW`, `Color.GREEN`, `Color.PINK`, `Color.RED`, `Color.MAGENTA` ou então alguma outra cor RGB que você queira definir[3]. Inicialmente, a tela consiste de quadrados pretos internamente e uma borda cinza. Conforme as peças forem se fixando na parte inferior da tela, as cores dos quadrados internos deverão ser modificadas. A cor de um quadrado na borda nunca muda.

O formato das peças será indicado na matriz `Point[][][] pecas`. Cada peça é formada por 4 ou 5 pontos, representando os 4 ou 5 quadrados que a compõem, e ela pode estar em uma dentre suas 4 rotações possíveis. Na primeira dimensão dessa matriz temos as peças em si, isto é, é possível acessar `pecas[0]`, `pecas[1]`, ..., `pecas[6]`. Na segunda dimensão temos a rotação, isto é, podemos acessar `pecas[i][0]`, `pecas[i][1]`, `pecas[i][2]` ou `pecas[i][3]` para todo $0 \leq i \leq 6$, sendo que `pecas[i][1]` é o formato da peça após a peça no formato `pecas[i][0]` ser rotacionada em sentido horário uma vez, e assim por diante. E na terceira dimensão temos os pontos de cada peça, ou seja, entre 4 e 5 posições, uma para cada ponto da peça. Se temos `Point p`, isto é, `p` é uma variável do tipo `Point`, então podemos acessar `p.x` e `p.y` (todo ponto possui duas coordenadas).

O arquivo atualmente possui a matriz `pecas` completa, no entanto com a peça A repetida nas 7 posições. Veja a Figura 2 para entender a representação da peça A. Note que essa peça em específico possui apenas dois estados diferentes de rotação. Mesmo assim, devemos preencher as 4 entradas na matriz. Você deverá modificar o conteúdo atual dessa matriz, implementando as suas 7 peças. Coloque a peça bomba na última posição dessa matriz, de modo que você possa verificar facilmente se a peça atual é uma bomba ou não, quando for necessário.

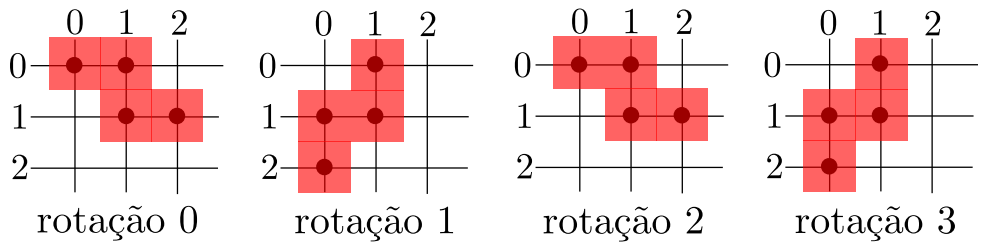


Figura 2: Representação do armazenamento da peça A.

[2]No entanto, ele não é complicado e você conseguiria facilmente se quiser.

[3]Não será necessário definir outa cor, mas se você tiver curiosidade e quiser aprender mais, pode ver a documentação aqui.

A cor de cada peça está definida no vetor `Color[] cor_peca`. Assim, em `cor_peca[i]` temos a cor da peça que está em `pecas[i]`.

O programa conta ainda com as variáveis:

- `int peca_atual`, que tem um valor entre 0 e 6 e indica o índice da peça que está caindo;
- `int rotacao`, que tem um valor entre 0 e 3 e indica qual a rotação da peça que está caindo;
- `Point centro_peca_atual`, que indica as coordenadas da peça que está caindo, isto é, a peça atual está na linha `centro_peca_atual.x` e coluna `centro_peca_atual.y`;
- `int[] proximas_pecas`, que armazena as próximas peças que irão cair;
- `long score`, que armazena a pontuação do jogo.

> Note que cada ponto `p` da peça atual (que pode ser um dos pontos em `pecas[peca_atual][rotacao]`) está na posição `tela[p.x + x][p.y + y]` da tela.

O programa também já conta com as seguintes funções implementadas:

- `void gera_tela()`, que inicializa a tela com as cores preta interiormente e cinza na borda;
- `void nova_peca()`, que gera uma nova peça no topo da tela;
- `boolean existe_colisao(int x, int y, int rotacao)`, que retorna `true` caso a peça atual colida se seu centro for colocado na posição (`x`,`y`) e sua rotação for dada por `rotacao`, e retorna `false` caso contrário. Note que ela não considera o centro atual nem a rotação atual da peça que está caindo;
- `void paintComponent(Graphics g)`, que desenha a tela toda do jogo atual;
- `void main(String[] args)`, que cria a interface gráfica e fica esperando que uma tecla seja pressionada e chama a devida função em cada caso. Adicionalmente, ela automaticamente faz a peça atual cair uma linha constantemente.

Segue a descrição de cada função que você deve implementar. Sugestão: siga essa mesma ordem na sua implementação.

- `void desce()`: se for possível, faz a peça atual cair uma única linha para baixo ou, se não for possível, fixa a peça na tela (atualiza as cores da matriz `tela` com as cores da peça atual, nas devidas posições).
- `void fixa_na_tela()`: faz com que a peça atual se torne parte da tela, trocando a cor dos quadrados da `tela`. Se a peça atual for a bomba, deve corretamente extinguir as peças ao redor (veja Seção 3)[4]. Ao fixar uma peça, ela pode gerar linhas completas. Depois desses tratamentos, uma nova peça deve ser gerada. Essa função é chamada quando não é possível mais descer uma peça.

[4]Dica: primeiro implemente todas as funções sem considerar que existe uma peça bomba para só então considerá-la.

- `void rotaciona()`: se possível (isto é, não houver colisão), rotaciona a peça atual no sentido horário. Termina com uma chamada à função `repaint()`, para que a tela seja redesenhada. Essa função é chamada quando a tecla "seta para cima" é pressionada.

- `void move(int i)`: se possível, move a peça atual uma posição para a direita se $i = 1$ ou para a esquerda se $i = -1$. Termina com uma chamada à `repaint()`. Essa função é chamada quando a tecla "seta para a esquerda" ou a tecla "seta para a direita" é pressionada.

- `void limpa_linhas_completas()`: verifica se há linhas completas e as remove da tela, fazendo com que as linhas acima delas desçam. Atualizamos o `score` de acordo com o número de linhas que foram removidas. Essa função é chamada quando a peça atual se fixa na tela.

- `void derruba()`: faz com que a peça atual caia imediatamente, não sendo mais possível descê-la, momento em que ela deve se fixar na tela. Essa função é chamada quando a tecla "espaço" é pressionada.

3 Bomba

Quando a peça bomba chega na parte inferior da tela (no momento em que ela é fixa na tela), deve-se excluir a própria peça e pedaços de peças ao redor dela, respeitando o formato da peça e considerando um raio de um quadrado. Formalmente, se (x, y) é a posição de um quadrado da peça, todas as 8 posições vizinhas a essa devem ser excluídas se for possível ($(x-1, y-1)$, $(x-1, y)$, $(x-1, y+1)$, $(x, y-1)$, $(x, y+1)$, $(x+1, y-1)$, $(x+1, y)$, $(x+1, y+1)$), inclusive a própria posição (x, y). Veja a Figura 3.

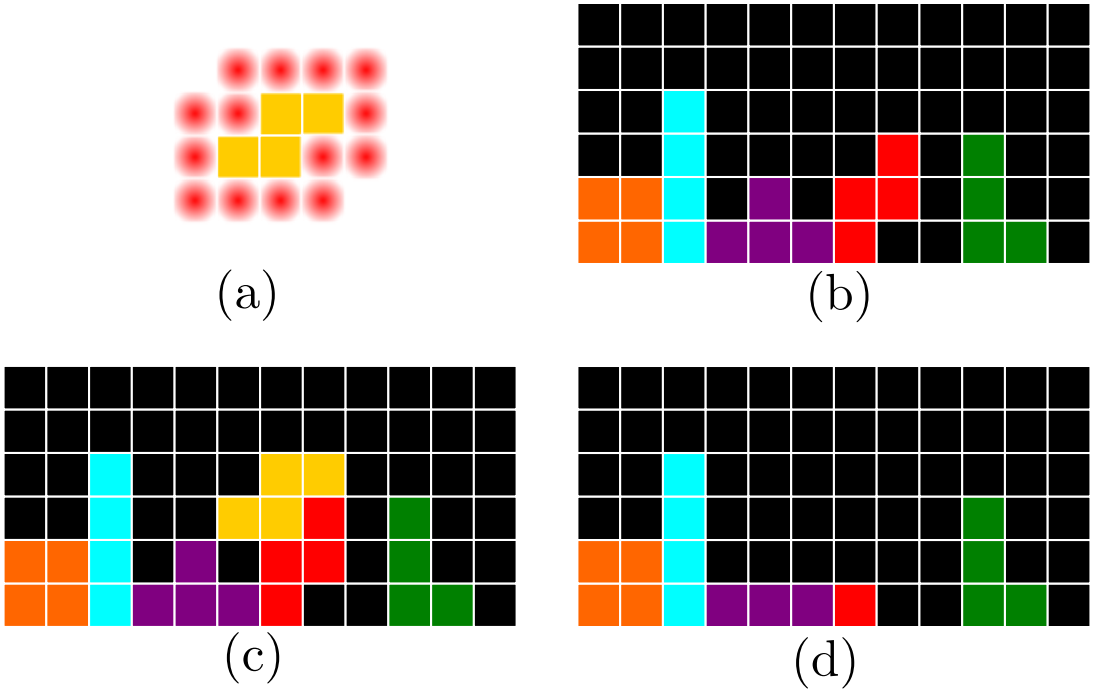


Figura 3: A parte (a) mostra a peça C e as posições ao seu redor que são explodidas. Dado o estado atual do jogo (parte (b)), se a peça C é encaixada (parte (c)), então todas as peças ao redor somem (parte (d)).

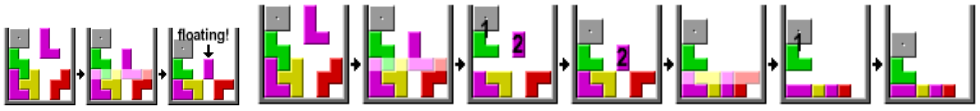
4 Entrega e avaliação

- O projeto deve ser feito **individualmente**, sem exceções.
- Você deverá enviar o arquivo `Pentris.java` contendo as suas implementações até o dia **04/05/2018 às 23:55** via atividade no TIDIA.
  - Soluções entregues fora do prazo valerão 70% da nota se entregues em até 24h.
  - Não serão aceitas soluções com mais de 24h de atraso.
- O cabeçalho do arquivo deve conter seu nome, os nomes das 6 peças escolhidas (referenciando a Figura 1) e um desenho simples da peça bomba desenvolvida.
- Arquivos enviados com erros de compilação (ou que não contenham o código fonte) receberão nota **zero**.
- A avaliação do projeto será feita em duas partes: uma automática (que contará inclusive com **detector automático de plágio**) e outra manual (comentários e indentação serão verificados).
- Fraudes de qualquer tipo não serão toleradas. Caso seja determinado que houve fraude durante a elaboração do trabalho, **todos** os envolvidos serão **reprovados** (conceito F) na disciplina.

5 Bônus

Com a possibilidade de ganhar **até** um ponto extra na média final da disciplina, você pode implementar outras funcionalidades para o seu jogo. Você **deve mencioná-las no seu arquivo**, explicando-as no cabeçalho, pois caso contrário não saberei que você as implementou. A seguir indico algumas, mas você pode ficar à vontade para criar as que julgar mais interessantes:

- Níveis: quando um determinado número de linhas são completadas, o nível do jogo aumenta, o que significa aceleração na queda das peças.
- Gravidade: atualmente as pilhas de blocos são movidas para baixo uma quantidade de linhas igual à quantidade de linhas que foi removida abaixo delas, ao contrário do que a gravidade faria, deixando blocos flutuantes. Você pode implementar uma versão em que as regiões continuam caindo até que toquem a parte inferior da tela ou pedaços já posicionados das peças. Note que isso pode gerar uma reação em cadeia, onde mais linhas podem ser completadas.



- Desfazer ação: se a tecla "U" for pressionada, deve-se desfazer o encaixe da última peça no jogo, bem como as ações que derivaram desse encaixe (remoção de linhas completas e explosão dos quadrados caso a peça tenha sido uma bomba).
- Próxima peça: a próxima peça pode ser mostrada na tela do jogo.
- Game over: atualmente o jogo está executando "para sempre". Quando uma peça atinge o topo da tela, deveria acontecer o fim do jogo.