INSTITUTO SUPERIOR TÉCNICO

Introdução aos Algoritmos e Estruturas de Dados 2009/2010

Enunciado do 2^{o} Projecto

Data de entrega: 03 de Maio de 2010 às 23h00

1 Introdução

Neste projecto pretende-se desenvolver um programa cujo objectivo é simular de forma restrita o conhecido jogo `Tetris`. O formato das peças usadas corresponde aos cinco tetraminós na Figura 1. Cada tetraminó tem uma de seis cores: vermelho, amarelo, azul, verde, laranja ou violeta. A Figura 3 mostra os possíveis tetraminós.

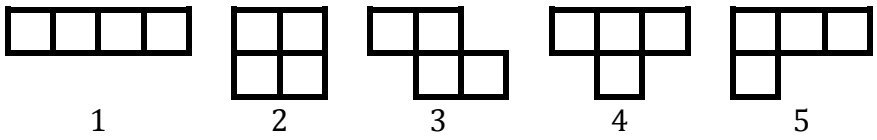

Figura 1: Formato das peças (tetraminós). Os formatos são identificados por inteiros de 1 a 5.

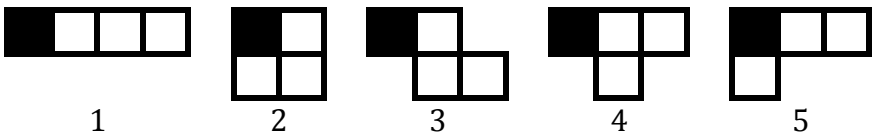

Figura 2: Quadrícula no canto superior esquerdo da peça (a preto) para cada tetraminó.

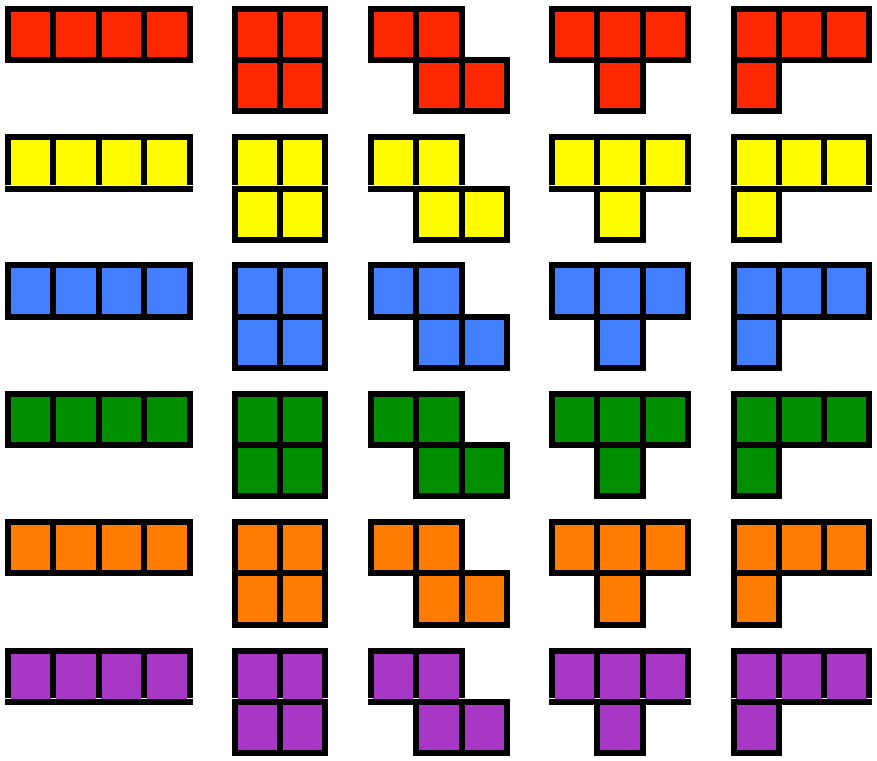

Figura 3: Possíveis tetraminós. As cores são identificadas pelos caracteres R, Y, B, G, O e V.

Na versão do jogo correspondente ao 2º projecto assumem-se as seguintes regras:

- Os tetraminós caem no espaço do jogo sem sofrer deslocações horizontais até pararem.
- Para cada tetraminó na Figura 1 é dada a coluna onde este cai no espaço de jogo, isto é a coluna da quadrícula no canto superior esquerdo da peça (ver Figura 2).
- Apenas podem ser jogados tetraminós com os formatos indicados na Figura 3, isto é, as peças não sofrem rotações ou reflexões.
- A largura do espaço do jogo é fixa e igual a 10 colunas (ver Figura 4).
- A altura do espaço do jogo é igual ao número de linhas com peças (ver Figura 4).
- Assume-se que a coluna onde cai o tetraminó permite colocar a peça no espaço de jogo, isto é os limites laterais do espaço do jogo não são ultrapassados. O limite superior do jogo aumenta como indicado acima garantindo que o limite superior não é ultrapassado e todas as peças são colocadas dentro do espaço do jogo (ver Figura 4).
- Quando uma linha fica completa com peças de mais que uma cor (“linha multicor”) é eliminada do espaço de jogo (ver Figura 5).

- Quando uma linha fica completa com peças de uma só cor (“linha monocor”) é eliminada juntamente com as duas linhas adjacentes (ver Figura 6).
- A eliminação de uma linha multicor dá ao jogador 1 ponto.
- A eliminação de uma linha monocor dá ao jogador 3 pontos.

A Figura 4 mostra um exemplo de final de jogo com 26 jogadas indicadas na Tabela 1.

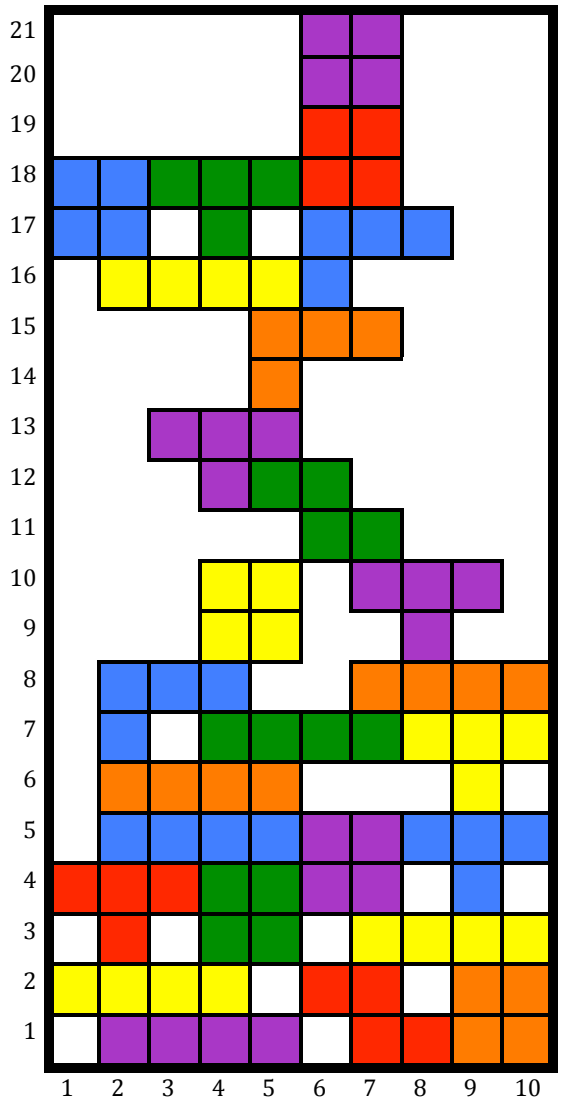

Figura 4: Exemplo de final de jogo.

Tabela 1: Sequência de jogadas efectuadas no jogo da Figura 4.

Jogada	Tetraminó	Coluna	Cor
1	1	2	Violeta (V)
2	3	6	Vermelho (R)
3	1	1	Amarelo (Y)
4	4	1	Vermelho (R)
5	2	9	Laranja (O)
6	1	7	Amarelo (Y)
7	2	6	Violeta (V)
8	2	4	Verde (G)
9	1	2	Azul (B)
10	1	2	Laranja (O)
11	4	8	Azul (B)
12	4	8	Amarelo (Y)
13	1	4	Verde (G)
14	5	2	Azul (B)
15	2	4	Amarelo (Y)
16	1	7	Laranja (O)
17	4	7	Violeta (V)
18	3	5	Verde (G)
19	4	3	Violeta (V)
20	5	5	Laranja (O)
21	1	2	Amarelo (Y)
22	2	1	Azul (B)
23	5	6	Azul (B)
24	4	3	Verde (G)
25	2	6	Vermelho (R)
26	2	6	Violeta (V)

A Figura 5 mostra um exemplo de jogada em que uma linha multicor fica completa e é eliminada. O jogador ganha 1 ponto. A Figura 6 mostra um exemplo de sequência de jogadas em que uma linha monocor fica completa e é eliminada juntamente com as linhas adjacentes. O jogador ganha 3 pontos. A Figura 7 mostra um exemplo de sequência de jogadas em que duas linha monocores ficam completas em simultâneo e são eliminadas, juntamente com as linhas adjacentes. O jogador ganha 6 pontos ficando com pontuação final de 10 pontos. As jogadas efectuadas nos exemplos das Figuras 5, 6 e 7 são efectuadas no seguimento das jogadas efectuadas no exemplo da Figura 4 (Tabela 1) e encontram-se na Tabela 2.

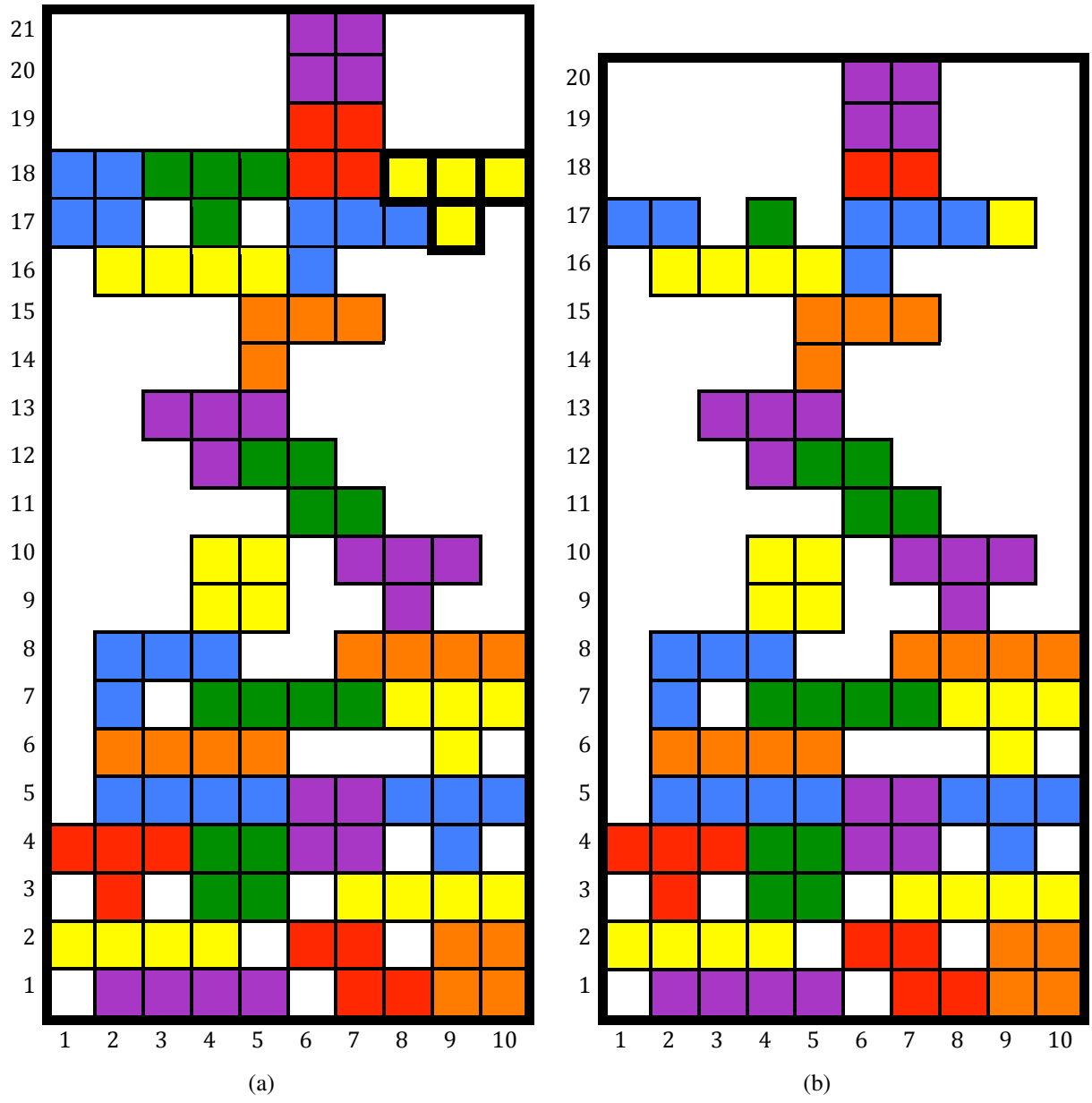

Figura 5: Exemplo de jogada com **eliminação de linha multicor**. (a) A peça com o formato 4 e cor amarela cai na coluna 8 na jogada 27. A linha 18 fica completa e é multicor. (b) A linha 18 é eliminada.

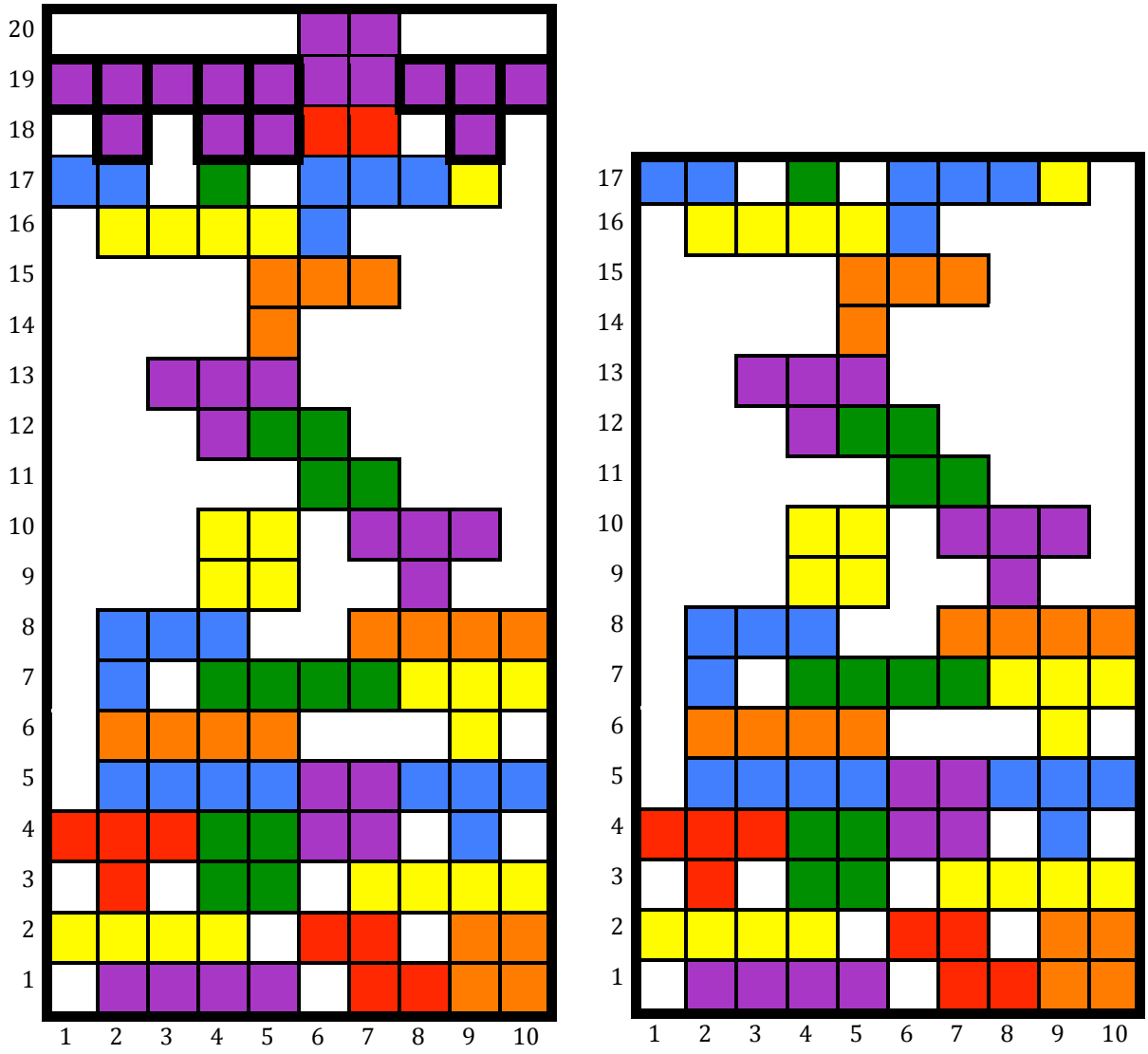

Figura 6: Exemplo de sequência de jogadas com **eliminação de linha monocor**. (a) As peças de cor violeta com formatos 4, 2 e 4 caiem nas colunas 1, 4 e 8, nas jogadas 28, 29 e 30, respectivamente. A linha 19 fica completa e é monocor. (b) A linha 19 é eliminada juntamente com as duas linhas adjacentes: linhas 18 e 20.

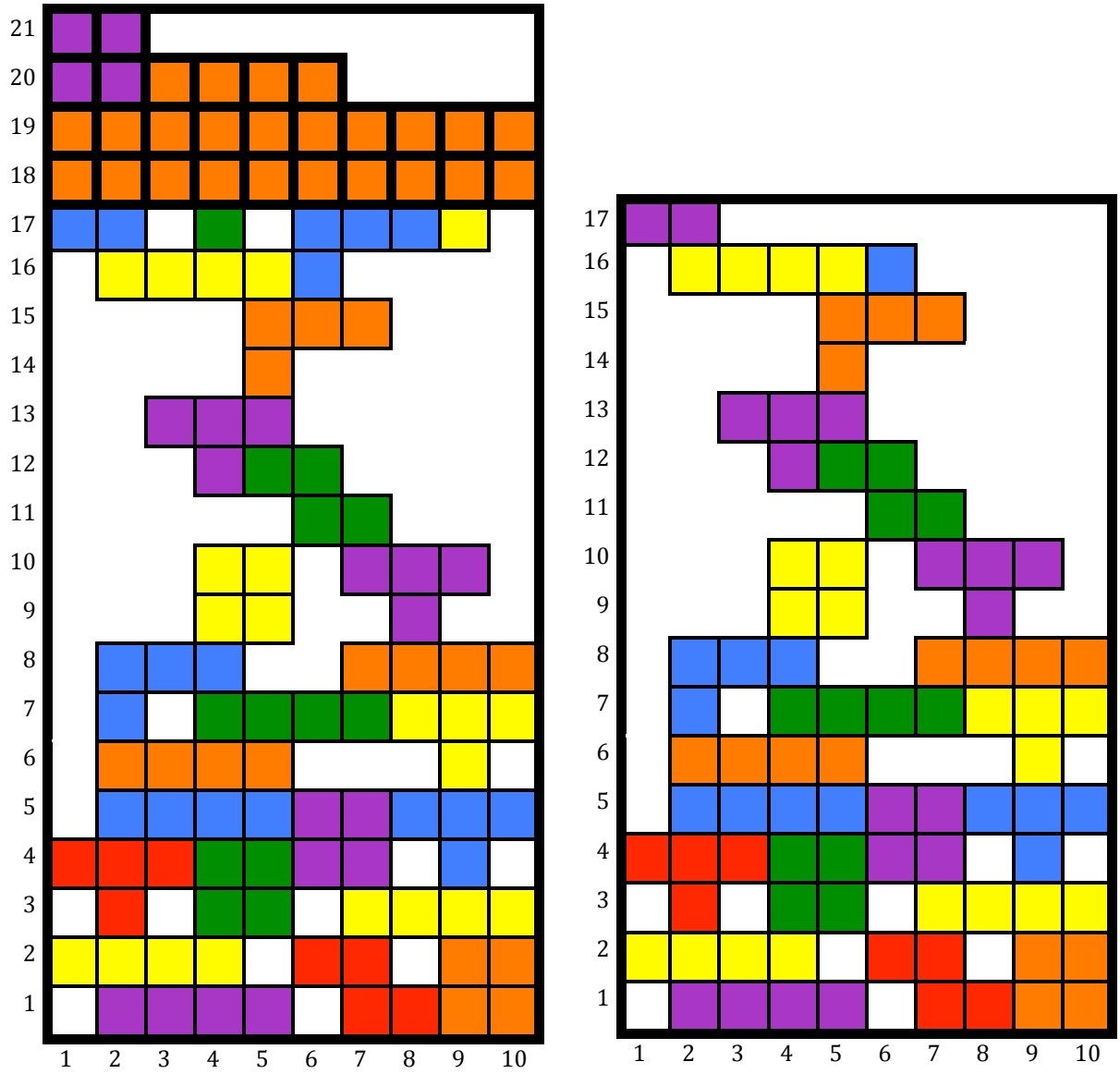

Figura 7: Exemplo de sequência de jogadas com **eliminação de duas linhas monocores**. (a) As peças de cor laranja com formatos 2, 1, 2, 2 e 2 caiem nas coluna 1, 3, 7, 3 e 5, nas jogadas 31, 32, 33, 34 e 35, respectivamente. A peça de cor violeta com formato 2 cai na coluna 1 na jogada 37. A peça de cor laranja com formato 2 cai na coluna 9 na jogada 37. As linhas 18 e 19 ficam completas em simultâneo e são monocores. (b) As linhas 18 e 19 são eliminadas juntamente com as duas linhas adjacentes: linhas 17 e 20.

Tabela 2: Sequência de jogadas efectuadas nos exemplos das Figuras 5, 6 e 7 no seguimento das jogadas efectuadas no exemplo da Figura 4 listadas na Tabela 1.

Jogada	Tetraminó	Coluna	Cor
27	4	8	Amarelo (Y)
28	4	1	Violeta (V)
39	2	4	Violeta (V)
30	4	8	Violeta (V)
31	2	1	Laranja (O)
32	1	3	Laranja (O)
33	2	7	Laranja (O)
34	2	3	Laranja (O)
35	2	5	Laranja (O)
36	2	1	Violeta (V)
37	2	9	Laranja (O)

2 Especificação do Programa

O programa recebe como *input* uma sequência de tetraminós, aos quais está associada uma coluna que permite determinar a localização final da peça, isto é todas as quadrículas (posições) ocupadas pela peça após a sua queda no espaço do jogo. O valor da coluna indica a coluna onde cai a quadrícula no canto superior esquerdo da peça (ver Figura 2) e não muda durante a queda da peça no espaço do jogo.

Por exemplo, se uma peça tem associada a coluna 8, tal como acontece com o tetraminó amarelo com formato 4 que está a negrito no jogo da Figura 5, então a peça cai ao longo das colunas 8, 9 e 10 e é colocada na linha 17 (coluna 9) e na linha 18 (colunas 8, 9 e 10). A localização da peça após a queda no espaço do jogo deve ser calculada de acordo com o formato da peça, a coluna onde a peça cai, e as posições livres e já ocupadas no espaço do jogo.

O valor da coluna apenas garante que a peça pode ser colocada sem ultrapassar os limites laterais do espaço do jogo. **É da responsabilidade do programador garantir que:**

1. O limite inferior do espaço de jogo não é ultrapassado.
2. O limite superior do jogo aumenta de forma a garantir a colocação da peça.
3. Todas as peças são colocadas dentro do espaço do jogo e não se sobrepõem.

3 Dados de Entrada

O programa deverá ler os dados de entrada a partir do *standard input*. O formato dos dados de entrada será o seguinte:

- uma linha contendo um inteiro N ($N \geq 1$) que denota o número total de peças do jogo;
- uma sequência de N linhas onde cada linha denota uma peça definida da seguinte forma:
  - um inteiro F com valor entre 1 e 5 que define o formato da peça (ver código de cada peça na Figura 1);
  - um espaço em branco;
  - um inteiro C com valor entre 1 e 10 que denota a coluna onde a quadrícula no canto superior esquerdo cai no espaço de jogo (ver quadrícula no canto superior esquerdo para cada formato de peça na Figura 2);
  - um espaço em branco;
  - uma letra maiúscula M que indica a cor da peça: R, Y, B, G, O ou V.

Assuma que os dados de entrada para o programa não contêm erros sintácticos, isto é, obedecem sempre ao formato descrito nesta secção.

4 Dados de Saída

Após processar as peças como indicado no input, o programa deverá escrever no *standard output* um conjunto de dados sobre a situação final de jogo. A informação a escrever será a seguinte e por esta ordem:

- uma linha com um inteiro $LMax$ onde $LMax$ denota a linha de maior índice com peças. ($LMax$ será zero no caso em que todas as linhas são eliminadas durante o jogo);
- uma linha com um inteiro P onde P denota a pontuação obtida pelo jogador ao longo do jogo: 1 e 3 pontos, respectivamente, por cada linha multicor e monocor eliminadas;
- uma linha em branco.
- uma sequência de $LMax$ linhas que denota a situação final do jogo. Começando na linha de maior índice com peças e de forma decrescente até à linha de índice 1, temos o seguinte:
  - cada linha é composta por um caracter 'l', 10 caracteres que correspondem ao estado de ocupação de cada coluna dessa linha, um caracter 'l', um espaço em branco, e o número da linha:

1. quando uma posição (linha,coluna) do jogo estiver ocupada, o caracter a mostrar denota a cor da peça que ocupa esse espaço (R, Y, B, G, O, V);
2. quando uma posição (linha,coluna) do jogo estiver livre, o caracter a mostrar deverá ser o espaço em branco.

- uma linha correspondente ao limite inferior do jogo: um espaço em branco e 10 caracteres '-' (esta linha deverá ser sempre mostrada, incluindo no caso em que todas as linhas são eliminadas durante o jogo).

5 Exemplo

5.1 Dados de Entrada

Vamos admitir que o ficheiro de entrada, que designaremos por `in.ttr`, contém as jogadas que levam à situação final de jogo ilustrada na Figura 7 (listadas nas Tabelas 1 e 2):

```
37
1 2 V
3 6 R
1 1 Y
4 1 R
2 9 O
1 7 Y
2 6 V
2 4 G
1 2 B
1 2 O
4 8 B
4 8 Y
1 4 G
5 2 B
2 4 Y
1 7 O
4 7 V
3 5 G
4 3 V
5 5 O
1 2 Y
2 1 B
5 6 B
4 3 G
2 6 R
```

```
2 6 V
4 8 Y
4 1 V
2 4 V
4 8 V
2 1 O
1 3 O
2 7 O
2 3 O
2 5 O
2 1 V
2 9 O
```

5.2 Resultados

Para o ficheiro de entrada `in.ttr` descrito na secção anterior, o programa deverá escrever na saída a seguinte informação:

```
17
10

|VV         | 17
| YYYYB     | 16
|    OOO    | 15
|    O      | 14
|  VVV      | 13
|   VGG     | 12
|     GG    | 11
|   YY VVV  | 10
|   YY  V   | 9
| BBB  OOOO| 8
| B GGGGYYY| 7
| OOOO   Y  | 6
| BBBBVVBBB| 5
|RRRGGVV B  | 4
| R GG YYYY| 3
|YYYY RR OO| 2
| VVVV RROO| 1
 ----------
```

6 Compilação do Programa

Deve concretizar o ficheiro `Makefile` onde deverá definir o processo de geração do executável correspondente ao programa realizado. **O compilador a utilizar é o** `gcc` **com as seguintes opções de compilação:** `-ansi -Wall -pedantic`**. Este processo deve ser definido na regra** `all` **do ficheiro** `Makefile`. O executável gerado deve ter o nome `proj2`. Para compilar o programa deve executar o seguinte comando:

```
$ make -s all
```

o qual deve ter como resultado a geração do ficheiro executável `proj2`, caso não haja erros de compilação. A execução deste comando não deverá escrever qualquer resultado no terminal (daí a utilização da opção `-s` do comando `make`). Caso a execução deste comando escreva algum resultado no terminal, considera-se que o programa não compilou com sucesso. Por exemplo, durante a compilação do programa, o compilador não deve escrever mensagens de *warning*.

7 Execução do Programa

O programa deve ser executado da forma seguinte:

```
$ ./proj2 < in.ttr > out.txt
```

8 Entrega do Projecto

A entrega do projecto deverá respeitar o procedimento seguinte:

- Na página da disciplina aceda ao sistema para entrega de projectos. O sistema será activado uma semana antes da data limite de entrega. Instruções de como aceder ao sistema serão oportunamente fornecidas.
- Efectue o upload de um ficheiro arquivo com extensão `.tgz` que inclua o seguinte:
  - Ficheiros fonte (`.c`) e cabeçalho (`.h`) que constituem o programa.
  - Ficheiro `Makefile` que permita criar o executável `proj2`.

  Para criar um ficheiro arquivo com a extensão `.tgz` deve executar o seguinte comando na directoria onde se encontram o ficheiro `Makefile` e os ficheiros com extensão `.c` e `.h`, criados durante o desenvolvimento do projecto:

  ```
  $ tar cfz proj2.tgz Makefile *.c *.h
  ```
- Como resultado do processo de upload será informado se a resolução entregue apresenta a resposta esperada num conjunto de casos de teste. **O sistema não permite submissões com menos de 30 minutos de intervalo para o mesmo grupo.** Exemplos de casos de teste serão oportunamente fornecidos.

- Data limite de entrega do projecto: **23h00 do dia 03 de Maio de 2010**. Até à data limite poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a **última** entrega efectuada. Deverá portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretende que seja avaliada. Não serão abertas excepções.

9 Avaliação do Projecto

9.1 Componentes da Avaliação

Na avaliação do projecto serão consideradas as seguintes componentes:

1. A primeira componente avalia o desempenho da funcionalidade do programa realizado. Esta componente é avaliada entre 0 e 16 valores.

2. A segunda componente avalia a qualidade do código entregue, nomeadamente os seguintes aspectos: comentários, identação, estruturação, modularidade, abstracção, entre outros. Esta componente poderá variar entre -4 valores e +4 valores relativamente à classificação calculada no item anterior e será atribuída na discussão final do projecto.

   Nesta componente **será também utilizado o sistema `valgrind` for forma a detectar fugas de memória ("memory leaks") ou outras incorrecções no código, que serão penalizadas**. Aconselha-se por isso que os alunos utilizem este sistema para fazer debugging do código e corrigir eventuais incorrecções antes da submissão do projecto.

   Grupos que não utilizem as flags correctas de compilação têm penalização de 100%.

Durante a discussão final do projecto será averiguada a participação de cada elemento do grupo na realização do projecto, bem como a sua compreensão do trabalho realizado, sendo a respectiva classificação ponderada em conformidade, isto é, elementos do mesmo grupo podem ter classificações diferentes. **Elementos do grupo que se verifique não terem participado na realização do respectivo projecto terão a classificação de 0 (zero) valores.**

9.2 Atribuição Automática da Classificação

A classificação da primeira componente da avaliação do projecto é obtida automaticamente através da execução de um conjunto de testes executados num computador com o sistema operativo **GNU/Linux**. Torna-se portanto essencial que o código compile correctamente e que respeite o formato de entrada e saída dos dados descrito anteriormente. Projectos que não obedeçam ao formato indicado no enunciado serão penalizados na avaliação automática, podendo, no limite, ter 0 (zero) valores se falharem todos os testes. Os testes considerados para efeitos de avaliação podem incluir (ou não) os disponibilizados na página da disciplina, além de um conjunto de testes adicionais. A execução de cada programa em cada teste é limitada na quantidade de

memória que pode utilizar, até um máximo de 32 MBytes, e no tempo total disponível para execução, sendo o tempo limite distinto para cada teste.

Note-se que o facto de um projecto passar com sucesso o conjunto de testes disponibilizado na página da disciplina não implica que esse projecto esteja totalmente correcto. Apenas indica que passou alguns testes com sucesso, mas este conjunto de testes não é exaustivo. É da responsabilidade dos alunos garantir que o código produzido está correcto.

Em caso algum será disponibilizada qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. A totalidade de ficheiros de teste usados na avaliação do projecto serão disponibilizados na página da disciplina após a data de entrega.

9.3 Detecção de Cópias

A avaliação dos projectos inclui a utilização de um sistema para detecção de situações de cópia entre projectos. A submissão de um projecto pressupõe o compromisso de honra que o trabalho incluso foi realizado única e exclusivamente pelos alunos referenciados nos ficheiros submetidos para avaliação. A quebra deste compromisso, ou seja a tentativa de um grupo se apropriar de trabalho de outrém (sejam colegas ou outra pessoa), tem como consequência a reprovação de todos os alunos envolvidos (incluindo quem possibilitar a ocorrência de cópias) à disciplina de IAED.

Toda e qualquer situação de fraude ou facilitação de fraude terá então como consequência a reprovação imediata à disciplina de IAED neste semestre, assim como a comunicação da ocorrência ao respectivo Coordenador de curso e ao Conselho Pedagógico do IST para sanções adicionais de acordo com as regras aprovadas pela UTL e publicadas em Diário da República.

9.4 Considerações adicionais

Todos os programas são avaliados do modo seguinte:

```
$ ./proj2 < in.ttr > out.txt; diff out.txt exp.txt
```

em que o ficheiro `exp.txt` representa o resultado esperado da execução do programa para os dados de entrada definidos pelo ficheiro `in.ttr`. A impossibilidade de verificar automaticamente o resultado da execução de um dado programa implica uma penalização de **100%**. Considera-se que um programa passou um teste com sucesso se o resultado produzido por esse programa for exactamente igual ao resultado esperado, o que significa que o comando `diff` não deverá encontrar diferenças entre o resultado produzido pelo programa submetido e o esperado. Para poder ser avaliado, um projecto deverá compilar correctamente num computador com o sistema operativo **GNU/Linux**, sendo o utilizado o compilador **gcc** da **GNU**. A entrega de código não compilável, ou a não inclusão de qualquer dos ficheiros requeridos, ou a utilização de nomes diferentes para o ficheiro executável conduz a uma classificação de 0 (zero) valores. Não serão aceites quaisquer justificações.