

Puzzle

Jorge Cardoso

24 de Outubro de 2005

Conteúdo

1	Resumo	1
2	Requisitos/Configuração	1
3	Introdução	1
4	Descrição do Programa	1
5	Implementação	2
5.1	Construção dos Blocos	2
5.2	Determinar a Posição de cada Bloco	4
5.3	Desenhar os Blocos no Ecrã	4
5.4	Mover os Blocos	5
5.5	Ajudar o Utilizador a Resolver o Puzzle	6
5.6	Listagem do Código	7
6	Adaptações	10

1 Resumo

O programa `Puzzle` constrói um puzzle 2D de blocos móveis, com peças quadradas, a partir de uma imagem. A aplicação permite que o utilizador resolva o puzzle.

2 Requisitos/Configuração

- Processing (BETA) [2]

3 Introdução

Os puzzles de blocos móveis são um tipo de puzzles em que as peças utilizadas (blocos) têm o seu movimento restringido pelas restantes peças do puzzle. Existem muitas variantes destes tipos de puzzles [1]. A Figura 1 mostra alguns tipos de puzzles de blocos móveis.

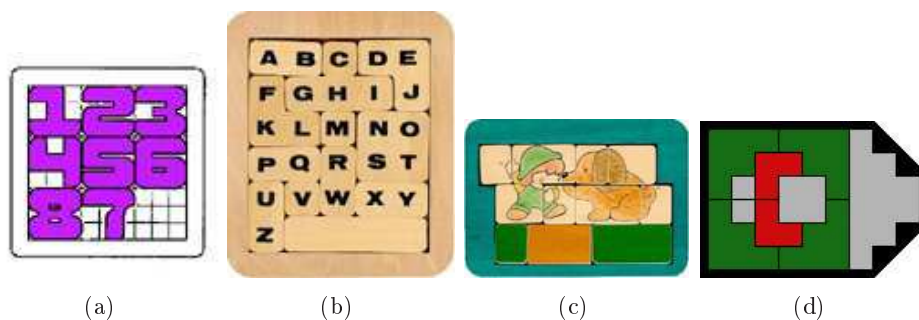


Figura 1: Tipos de puzzles de blocos móveis

As variantes destes puzzles incluem utilizar blocos de dimensão e formas diferentes e áreas de formas diversas onde as peças se movem.

A variante mais simples é a que utiliza peças quadradas que se deslocam numa área rectangular.

4 Descrição do Programa

O programa `Puzzle` constrói um puzzle de blocos móveis na sua variante mais simples. O puzzle construído contém 12 peças quadradas (4 peças em largura e 3 em altura). Para se obter os 12 blocos dividimos uma imagem de 400x300 pixels em blocos de 100x100 pixels.

5 Implementação

5.1 Construção dos Blocos

O nosso programa utiliza uma única imagem de base para gerar os blocos do puzzle. A Figura 2 mostra a imagem utilizada. A imagem tem 400 pixels



Figura 2: Imagem original utilizada para gerar os blocos

de largura por 300 pixels de altura. Uma vez que o nosso puzzle tem 12 peças (4x3) temos de dividir a imagem em blocos de 100x100 pixels tal como exemplificado na Figura 3.



Figura 3: Divisão da imagem original em blocos

Em código podemos fazer isto criando uma `PImage`¹ para a imagem original e um vector de 12 `PImage` para os 12 blocos. O conteúdo das `PImage` dos blocos é obtido copiando uma área da imagem original:

```
/* A imagem inteira */
PImage imgInteira;

/* A imagem em blocos */
PImage blocos[];
```

¹`PImage` é um tipo especial de variável, em Processing, que guarda uma imagem.

```

/* carregar a imagem original */
imgInteira = loadImage("muffin.jpg");

/* criar os blocos */
blocos = new PImage[12];

/* TODO: colocar isto num ciclo! */
blocos[0] = new PImage(100, 100);
blocos[0].copy(imgInteira, 0, 0, 100, 100, 0, 0, 100, 100);

blocos[1] = new PImage(100, 100);
blocos[1].copy(imgInteira, 100, 0, 100, 100, 0, 0, 100,
100);

blocos[2] = new PImage(100, 100);
blocos[2].copy(imgInteira, 200, 0, 100, 100, 0, 0, 100,
100);

blocos[3] = new PImage(100, 100);
blocos[3].copy(imgInteira, 300, 0, 100, 100, 0, 0, 100,
100);

blocos[4] = new PImage(100, 100);
blocos[4].copy(imgInteira, 0, 100, 100, 100, 0, 0, 100,
100);

blocos[5] = new PImage(100, 100);
blocos[5].copy(imgInteira, 100, 100, 100, 100, 0, 0, 100,
100);

blocos[6] = new PImage(100, 100);
blocos[6].copy(imgInteira, 200, 100, 100, 100, 0, 0, 100,
100);

blocos[7] = new PImage(100, 100);
blocos[7].copy(imgInteira, 300, 100, 100, 100, 0, 0, 100,
100);

blocos[8] = new PImage(100, 100);
blocos[8].copy(imgInteira, 0, 200, 100, 100, 0, 0, 100,
100);

blocos[9] = new PImage(100, 100);
blocos[9].copy(imgInteira, 100, 200, 100, 100, 0, 0, 100,
100);

blocos[10] = new PImage(100, 100);
blocos[10].copy(imgInteira, 200, 200, 100, 100, 0, 0, 100,
100);

blocos[11] = new PImage(100, 100);
blocos[11].copy(imgInteira, 300, 200, 100, 100, 0, 0, 100,
100);

```

5.2 Determinar a Posição de cada Bloco

O objectivo do puzzle é o utilizador reconstruir a configuração original das peças de modo a corresponder à imagem original. Assim, num determinado momento, cada um dos blocos pode estar numa posição aleatória. Se numerarmos os blocos, nas suas posições correctas, da esquerda para a direita e de cima para baixo, teremos algo como na Figura 4. Num dado momento, o

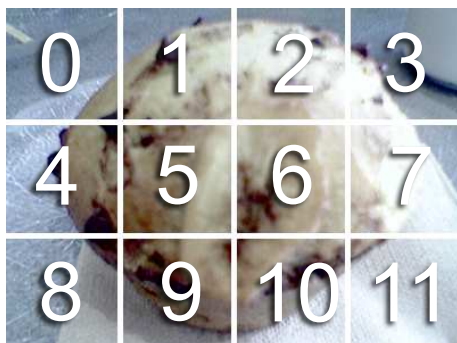


Figura 4: Blocos numerados

bloco 10 pode, no entanto, estar na posição 3, por exemplo.

Para sabermos a posição de cada bloco num dado momento podemos utilizar uma matriz (que representa a área 4*3), em que cada entrada contém o número do bloco que está nesse momento nessa posição:

```
int posicoes [][] = {{0, 1, 2, 3},
                    {4, 5, 6, 7},
                    {8, 9, 10, -1}};
```

Esta configuração corresponde à situação em que cada bloco está na posição correcta.

A última entrada -1 , serve para indicar que a posição está vazia. Neste tipo de puzzles é necessário existir sempre uma área vazia para podermos mover os blocos. No nosso caso, escolhemos a última posição para inicializar a área vazia.

5.3 Desenhar os Blocos no Ecrã

Tendo as posições de cada bloco é fácil desenharmos a configuração no ecrã: Basta percorrer a matriz das posições e desenhar o bloco indicado na matriz na posição do bloco. Uma vez que as áreas do puzzle estão em posições fixas e múltiplas de 100 (Figura 5) é fácil desenhar os blocos:

```
/* desenhar os blocos nas posicoes correspondentes */
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        if (posicoes[i][j] >= 0) {
            image(blocos[posicoes[i][j]], j*100, i*100);
        }
    }
}
```

```

    }
}
}

```

Apenas temos de ter cuidado com o bloco vazio para não desenharmos nada nesse caso. Para isso basta verificar se o valor da posição é maior ou igual a zero (o bloco vazio é indicado como -1).

5.4 Mover os Blocos

Uma vez que o objectivo é que o utilizador mova os blocos de modo a tentar conseguir obter a configuração correcta, temos de possibilitar esse movimento de alguma forma. A forma mais óbvia é através do rato: se o utilizador clicar num bloco adjacente ao bloco vazio, o bloco passa para a posição do bloco vazio e vice-versa. Em Processing, temos acesso aos eventos do rato, de forma assíncrona, através do método de sistema `mousePressed()`. Este método é invocado pelo Processing sempre que o botão esquerdo do rato é pressionado. A posição do ponteiro do rato é obtida através das variáveis `mouseX` e `mouseY`.

No caso da nossa aplicação interessa-nos obter, não a posição do rato, mas o número do bloco em que o rato foi pressionado. No nosso caso, isto é facilmente conseguido dividindo a posição do rato por 100 (divisão inteira):

```

/* determinar em que bloco o rato foi pressionado. Basta fazer
   a divisão inteira por 100..*/
int blocoX = mouseX/100;
int blocoY = mouseY/100;

```

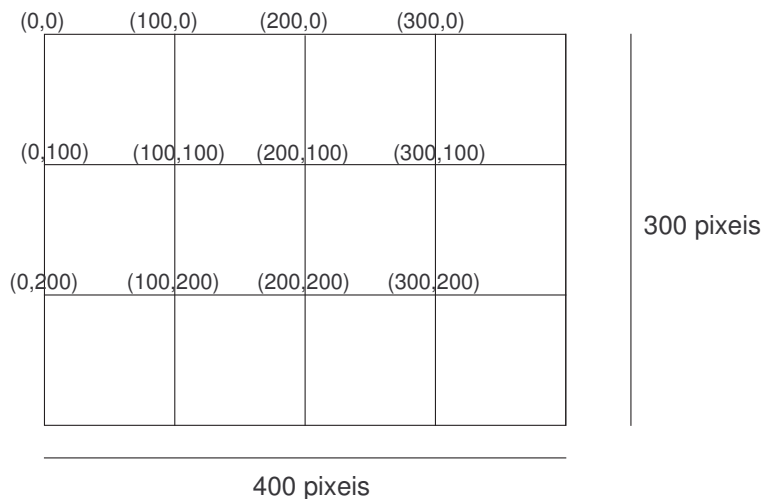


Figura 5: Posições de cada bloco no ecrã

Depois de sabermos em que bloco o utilizador pressionou, temos de determinar se é possível mover o bloco. Apenas podemos movê-lo se existir o bloco vazio numa das posições adjacentes:

```
/* procurar o bloco vazio à volta do bloco pressionado e se o
   encontrarmos, trocar os blocos */
// procurar no bloco à esquerda
if (blocoX > 0) {
    if (posicoes[blocoY][blocoX-1] == -1) {
        posicoes[blocoY][blocoX-1] = posicoes[blocoY][blocoX];
        posicoes[blocoY][blocoX] = -1;
        return;
    }
}
// procurar no bloco à direita
if (blocoX < 3) {
    if (posicoes[blocoY][blocoX+1] == -1) {
        posicoes[blocoY][blocoX+1] = posicoes[blocoY][blocoX];
        posicoes[blocoY][blocoX] = -1;
        return;
    }
}
// procurar no bloco acima
if (blocoY > 0) {
    if (posicoes[blocoY-1][blocoX] == -1) {
        posicoes[blocoY-1][blocoX] = posicoes[blocoY][blocoX];
        posicoes[blocoY][blocoX] = -1;
        return;
    }
}
// procurar no bloco abaixo
if (blocoY < 2) {
    if (posicoes[blocoY+1][blocoX] == -1) {
        posicoes[blocoY+1][blocoX] = posicoes[blocoY][blocoX];
        posicoes[blocoY][blocoX] = -1;
        return;
    }
}
```

5.5 Ajudar o Utilizador a Resolver o Puzzle

De forma a facilitarmos o trabalho do utilizador, vamos dar-lhe a possibilidade de, em qualquer instante, visualizar a imagem na sua configuração original. Para isso vamos definir uma tecla de ajuda: a tecla 'o' ou 'O'. Sempre que o utilizador pressionar esta tecla a imagem original será mostrada no ecrã.

Podemos determinar se uma tecla foi pressionada através da variável de sistema lógica `keyPressed`. Esta variável tem o valor `true` sempre que uma tecla é pressionada. Para sabermos qual a tecla pressionada temos a variável `key`:

```

/* Se o utilizador pressionar a tecla 'o', mostramos a imagem
original */
if(keyPressed) {
    if (key == 'o' || key == 'O') {
        image(imgInteira, 0, 0);
    }
}

```

Este código aparece no final do método `draw()`, de modo a que a imagem original é desenhada por cima de tudo o resto no ecrã, caso o utilizador tenha pressionado a tecla de ajuda.

5.6 Listagem do Código

```

// Puzzle
// Autor: Jorge Cardoso
// Data: 24 Outubro 2005
// Versão 0.5

/* A imagem inteira */
PImage imgInteira;

/* A imagem em blocos */
PImage blocos[];

/* As posicoes de cada bloco no ecrã.
   TODO: Gerar as posicoes aleatoriamente
*/
int posicoes[][] = {{0, 1, 2, 3},
                   {4, 5, 6, 7},
                   {8, 9, 10, -1}};

/* Inicialização */
void setup() {
    size(400, 300);

    /* carregar a imagem original */
    imgInteira = loadImage("muffin.jpg");

    /* criar os blocos */
    blocos = new PImage[12];

    /* TODO: colocar isto num ciclo! */
    blocos[0] = new PImage(100, 100);
    blocos[0].copy(imgInteira, 0, 0, 100, 100, 0, 0, 100, 100);

    blocos[1] = new PImage(100, 100);
    blocos[1].copy(imgInteira, 100, 0, 100, 100, 0, 0, 100,
                  100);

    blocos[2] = new PImage(100, 100);

```

```

    blocos[2].copy(imgInteira, 200, 0, 100, 100, 0, 0, 100,
        100);

    blocos[3] = new PImage(100, 100);
    blocos[3].copy(imgInteira, 300, 0, 100, 100, 0, 0, 100,
        100);

    blocos[4] = new PImage(100, 100);
    blocos[4].copy(imgInteira, 0, 100, 100, 100, 0, 0, 100,
        100);

    blocos[5] = new PImage(100, 100);
    blocos[5].copy(imgInteira, 100, 100, 100, 100, 0, 0, 100,
        100);

    blocos[6] = new PImage(100, 100);
    blocos[6].copy(imgInteira, 200, 100, 100, 100, 0, 0, 100,
        100);

    blocos[7] = new PImage(100, 100);
    blocos[7].copy(imgInteira, 300, 100, 100, 100, 0, 0, 100,
        100);

    blocos[8] = new PImage(100, 100);
    blocos[8].copy(imgInteira, 0, 200, 100, 100, 0, 0, 100,
        100);

    blocos[9] = new PImage(100, 100);
    blocos[9].copy(imgInteira, 100, 200, 100, 100, 0, 0, 100,
        100);

    blocos[10] = new PImage(100, 100);
    blocos[10].copy(imgInteira, 200, 200, 100, 100, 0, 0, 100,
        100);

    blocos[11] = new PImage(100, 100);
    blocos[11].copy(imgInteira, 300, 200, 100, 100, 0, 0, 100,
        100);
}

void draw() {
    background(0);

    /* desenhar os blocos nas posicoes correspondentes */
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            if (posicoes[i][j] >= 0) {
                image(blocos[posicoes[i][j]], j*100, i*100);
            }
        }
    }
}

```

```

/* desenhar a grelha */
stroke(255); // cor branca para a linha

// linhas verticais
for (int i = 0; i < 400; i = i + 100) {
    line(i, 0, i, 300);
}
// linhas horizontais
for (int i = 0; i < 300; i = i + 100) {
    line(0, i, 400, i);
}

/* Se o utilizador pressionar a tecla 'o', mostramos a
imagem original */
if(keyPressed) {
    if (key == 'o' || key == 'O') {
        image(imgInteira, 0, 0);
    }
}
}

void mousePressed() {

    /* determinar em que bloco o rato foi pressionado. Basta
fazer a divisao inteira por 100..*/
    int blocoX = mouseX/100;
    int blocoY = mouseY/100;

    /* procurar o bloco vazio a volta do bloco pressionado e se
o encontrarmos, trocar os blocos */
    // procurar no bloco a esquerda
    if (blocoX > 0) {
        if (posicoes[blocoY][blocoX-1] == -1) {
            posicoes[blocoY][blocoX-1] = posicoes[blocoY][
                blocoX];
            posicoes[blocoY][blocoX] = -1;
            return;
        }
    }
    // procurar no bloco a direita
    if (blocoX < 3) {
        if (posicoes[blocoY][blocoX+1] == -1) {
            posicoes[blocoY][blocoX+1] = posicoes[blocoY][
                blocoX];
            posicoes[blocoY][blocoX] = -1;
            return;
        }
    }
    // procurar no bloco acima
    if (blocoY > 0) {
        if (posicoes[blocoY-1][blocoX] == -1) {
            posicoes[blocoY-1][blocoX] = posicoes[blocoY][
                blocoX];
            posicoes[blocoY][blocoX] = -1;
        }
    }
}

```

```
        return;
    }
}
// procurar no bloco abaixo
if (blocoY < 2) {
    if (posicoes[blocoY+1][blocoX] == -1) {
        posicoes[blocoY+1][blocoX] = posicoes[blocoY][
            blocoX];
        posicoes[blocoY][blocoX] = -1;
        return;
    }
}
}
```

6 Adaptações

1. Fazer os TODO do código.
2. Generalizar o código para imagens de qualquer tamanho.

Referências

- [1] “The Sliding Block Puzzle Page”. <http://www.puzzleworld.org/SlidingBlockPuzzles/default.htm>
- [2] Ben Fry and Casey Reas, “Processing (BETA)”, <http://processing.org/>