Universidade de Évora

Departamento de Engenharia Informática

Unidade Curricular: Programação II

Smoothy

Professores:

Salvador Abreu

Lígia Ferreira

Rui Oliveira, nº 31511 | Luís Teimas, nº 31561

Évora, Junho de 2016

Introdução

O projecto que nos foi solicitado tem o nome de Smoothy, e será realizado no ambito da unidade curricular Programação 2.

O Smoothy tem como base o jogo mundialmente conhecido Candy Crush, porém é uma versão mais simplificada, e tem as seguintes regras:

- o jogador pode escolher uma peça qualquer para ser retirada, desde que esta esteja adjacente a pelo menos uma outra da mesma cor.
- serão eliminadas todas as peças dessa cor que estejam adjacentes à designada, ou a qualquer outra que lhe seja adjacente, e por aí em diante (duas peças em diagonal não são consideradas adjacentes).
- Quando um grupo de peças é eliminado, estas irão abrir um “buraco” no tabuleiro (este será preenchido com as peças que estão por cima).
- Quando todas as peças duma coluna tiverem desaparecido, as colunas à sua esquerda serão “empurradas” para a direita.
- Termina o jogo quando não houver mais jogadas possíveis (ou porque foram eliminadas todas as peças ou porque já não há grupos de peças adjacentes da mesma cor).

Tendo isto em conta, esperamos fazer o melhor trabalho possivel, e conseguir superar todas as dificuldades que nos serão impostas.

Na resolução do trabalho, não iremos usar cores propriamente ditas, iremos usar numeros. Para este efeito, o numero 0 é considerado um espaço vazio, todos os outros serão “cores”, apesar disto resolvemos continuar a chamar-lhes cores de forma a tornar o trabalho mais simples de compreender, evitando possiveis confusões.

Smoothy

Dado que o tabuleiro é a parte mais importante do trabalho, resolvemos criar uma classe Tabuleiro(), que tem como tributos todas as variveis globais que iremos utilizar ao longo da resolução do trabalho. Esses atributos são:

- int table: o array bi-dimensional que imprime o nosso tabuleiro;
- int c: a variavel que pede o numero de cores;
- int cordX: coordenada X que pedimos ao utilizador;
- int cordY: coordenada Y que pedimos ao utilizador;
- int contador: variavel que soma o numero de jogadas;
- int tamanho: o tamnho do tabuleiro, sendo quadrado, que será tamanho*tamanho;

O construtor para Tabuleiro(), tem o seguinte aspecto:

```
	public Tabuleiro(int tamanho){
	int aux=1;
	do{
		try{
			//utilizador escolhe o tamanho
			System.out.println("Qual o tamanho do tabuleiro: ");
			Scanner input = new Scanner(System.in);
			this.tamanho = input.nextInt();
			if(this.tamanho < 3 || this.tamanho > 6){
				System.out.println("Insira um tamanho entre 3 e 6");
			}else{
				table = new int[this.tamanho][this.tamanho];
				aux=2;
			}
		}
		catch(Exception e){
			System.out.println("Valor errado, insira um numero inteiro");
		}
	}while(aux==1);
}
```

Como se pode verificar, a função ira receber um input do utilizador para decidir o tamanho do tabuleiro, caso não seja um inteiro, o programa irá tratar esse input como uma execpção e irá pedir ao utilizador para inserir outro input.

De seguida resolvemos criar uma função para inicializar o tabuleiro, e chamamos a essa função init(). É uma função relativamente simples, recebe os inputs do utilizador, verifica se c está entre 3 e 6, caso não estejam a excepção é resolvida, e de seguida vai chamar a função preencheTab(c), que iremos explicar mais à frente e no final imprime o tabuleiro com a função printTabuleiro(), que também irá ser explicada mais à frente.

Em relação à função preencheTab(int ncores), dado que não retornada nada, resolvermos torna-la do tipo void, e como já foi referido a cima, esta é a função que preenche o tabuleiro com os numeros ao random.

Para criar esses numeros random, utilizamos uma variavel chamada rand, e é com ela que os numeros vão ser aleatoriamente originados. A função é relativamente simples, tem dois ciclos for e um print, porém tem uma linha de código um pouco mais complexa que iremos explicar de seguida:

```
table[i][j] = rand.nextInt(ncores)+1;
```

Esta é a linha que cria os numeros, e que vai preencher o tabuleiro em i e j (x e y, respectivamente), e o rand vai receber a varivel ncores como atributo, e vai gerar numeros aleatorios consoante o ncores.

Na função init() esta função recebe uma variavel c, dado que estamos a trabalhar com inputs do utilizador. (Nota: utilizamos o "+1" utlizado é para garantir que todos os numeros gerados começam em 1, e não em 0).

Em relação à função printTabuleiro() é novamente uma função relativamente simples, e simplesmente imprime o tabuleiro criado na função init().

De seguida temos uma função chamada jogar() e esta função é, como o nome indica, a função onde o utilizador começa a jogar, a função utiliza dois scanner, um para o input para a coordenada x e outro para o input da coordenada y (inputX e inputY respectivamente), esta função irá verificar se há ou não excepções, caso hajam irá resolve-las, e no fim irá chamar outra função, chamada jogar1() de forma a analizar as jogadas.

Ainda na função jogar(), resolvemos inverter a com que pedimos os valores x e y, como se pode observar no codigo, dado que o nosso referencial para o tabuleiro estava invertido (as coordenadas x eram as verticais e as coordenadas y eram as horizontais), resolvemos fazer isto para envitar confusões.

A função remove() é uma das mais simplies de todo o jogo, apenas iguala as variaveis cordX e cordY a 0, de modo a apagar a coordenada já utilizada.

Agora temos a função mais complexa da classe Tabuleiro, e essa função tem o nome de jogar1(), esta é a função que vai fazer todas as verificações da vizinhança da coordenada escolhida, isto é, vai verificar se o valor a cima, à direita, à esquerda e a baixo é igual à coordenada dos inputs do utilizador. Por exemplo, se o utilizador colocar a coordenada (1,1) a função irá verificar o valor de (0,1), (1,0),(2,1), (1,2). Se forem iguais prodece de uma forma, se forem diferentes procede de outra, no final a função irá chamar a função jogar() novamente para receber outras coordenadas, e por assim em diante até ficarmos sem jogadas possiveis.

```
        if((cordX -1 >= 0 && cordX -1 < table.length && cordY >= 0 && cordY < 
table.length) && (table[cordX][cordY] - table[cordX-1][cordY]) == 0){
          contador++;
          //verifica(cordX, cordY);
          table[cordX-1][cordY] = 0;
          jogar1();
        }
```

O exemplo a cima é o utilizado para verificar a coordenada acima do input do utilizador, em todas estas verificações trartamos das excepções para o caso de as coordenadas "vizinhas" à do input do utilizador passar os limites do tabuleiro.

```
            else{
            contador+=0;
            System.out.println("Não há mais jogadas");
            //return contador;
            remove();
            printTabuleiro();
            System.out.println("Já somou: " + contador + " pontos.");
            jogar();
        }
        return contador;
    }
```

No final da função, quando já não há vizinhanças, a coordenada do utilizador é removida, os pontos contados na jogada serão mostrados, e, como já foi referido, a função irá voltar a jogar() de forma a repetir o processo.

Por fim, iremos retornar o valor do contador, de forma a poder utiliza-lo noutras funções.

Em relação à classe Smoothy, esta é bastante simples, e apenas cria um novo tabuleiro e inicia o jogo.

```
Tabuleiro smoothy1 = new Tabuleiro();
smoothy1.init();
smoothy1.jogar();
```

Conclusão

Tendo terminado o projecto, podemos chegar a algumas conclusões. A primeira é que com este trabalho conseguimos cimentar e até mesmo aumentar os nossos conhecimentos na linguagem java.

Infelizmente não conseguimos cumprir todos os objectivos do trabalho, em primeiro lugar não conseguimos arranjar uma função que faça os valores “cair”.

Em segundo lugar não conseguimos arranjar forma de implementar o fecho transitivo da relação de adjacência, tentamos varias maneiras, mas em nenhuma obtivemos o resultado desejado.

Em terceiro lugar, caso se insira um cordX ou cordY com um valor superior ao do tabuleiro, por exemplo, caso o tabuleiro seja 3 por 3, e se inserirmos o valor 4 tanto em cordX como em cordY, o programa irá parar imediatamente.

Por fim, não conseguimos arranjar maneira de acabar o jogo, caso não haja mais jogadas o programa irá continuar a pedir os valores da coordenada x e da y até o utilizador carregar no “esc” para terminar o programa.

Apesar de tudo isto foi bastante interessante trabalhar neste projecto e estamos satisfeitos com o resultado final.