Relatório

Programação 2

15 de Junho de 2016

Trabalho realizado por:

Marcelo Babau nº30372
Nuno Serol nº31737

Introdução

Com este trabalho, pretendemos desenvolver um jogo (Smoothy) inspira-se no jogo Candy Crush e passa-se num tabuleiro quadrado, no qual há peças de várias cores (vários números, neste caso), colocadas aleatoriamente. Como jogada, o jogador pode escolher uma peça qualquer para ser retirada, desde que esta esteja adjacente a pelo menos uma outra da mesma cor . Nesse caso, 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.

Desenvolvimento

Este trabalho está dividido em quatro classes (*Peça, Tabuleiro, Jogo e tabMain*).

Criámos a classe *Peça*, que representa cada peça do tabuleiro, esta tem três variáveis, que são a linha e a coluna em que se situa no tabuleiro, e o tipo, que neste caso é um número, mas que representa uma cor, que é gerado aleatóriamente.

```
class Peca{
        int linha;
        int coluna;
        int tipoPeca;


        public Peca(int x, int y, int t){
        linha=x;
        coluna=y;
        tipoPeca=t;

}
```

A classe *Tabuleiro*, gera uma ArrayList de peças, em que o método metePecas atribui um tipo aleatório, a cada uma, que é a cor da peça, neste caso um número, para gerar este número aleatório, utilizamos a classe java.util.Random, e assim criamos o tabuleiro de jogo:

```
class Tabuleiro{

        int r;
        List<Peca> tab;
        int c;

        public Tabuleiro(){
        }

        public Tabuleiro(int n, int cc){
                r=n;
                tab= new ArrayList<Peca>();
```

```
            tab = metePecas(cc);
        }


public List<Peca> metePecas(int c){
            Random rand = new Random();

            for(int linha=0 ; linha < r ; linha++){
                for(int coluna = 0; coluna < r ; coluna ++){
                    int n =rand.nextInt((c-1)+1)+1;
                    tab.add(new Peca(linha,coluna,n));
                }
            }
        return tab;
}
```

Dentro desta classe estão também os métodos, *removerPecas*, que remove a peça selecionada, e as suas adjacentes, o método *cairPecas*, que faz com que as peças caiam na vertical, isto é, coluna a coluna, e o método *empurrarPecas*, que faz com que as colunas se movam horizontalmente para a direita, na presença de uma coluna sem peças à sua direita.

```
public void removerPecas(List<Peca> lp){
        for (int n=0; n<=lp.size()-1; n++){
            Peca pecaRemo = lp.get(n);
            if (pecaExiste(pecaRemo)) {
                for (int x=0; x<=tab.size()-1; x++) {
                    if (pecaRemo.getLinha()==tab.get(x).getLinha() &&
pecaRemo.getColuna()==tab.get(x).getColuna()) {
                        tab.get(x).setPtipo(0);
                    }
                }
            }
        }
            cairPecas();

            empurrarPecas();
        }
```

A classe *Jogo* é consituida por vários métodos, como o método *startjogo* que serve para dar inicio ao jogo, esta classe começa por pedir ao utilizador as coordenadas da peça que pretende retirar, verifica se esta peça pertence ao tabuleiro, para poder realizar a jogada, verifica também se tem alguma peça adjacente, pois sem pelo menos uma, a jogada é invalida.

```
class Jogo{
	int score;
	Tabuleiro t;
	List<Peca> listaRemover = new ArrayList<Peca>();

	public Jogo(Tabuleiro t){
		this.t=t;

	}



public void startjogo(Tabuleiro tab){
	int linha;
	int coluna;
	score = 0;

	while(true){
		tab.printTabuleiro();
		System.out.println();
		System.out.println("As cordenadas da peça. insira a LINHA ");   //linha
		Scanner s= new Scanner(System.in);
		linha=s.nextInt();

		System.out.println("Agora a COLUNA  \n");     //coluna
		coluna=s.nextInt();

		Peca pedida = tab.returnPeca(linha, coluna);

		int r=tab.getR();
		if (linha>=r||coluna>=r) {
			System.out.println("Peça não existe, escolher outra \n");
			startjogo(tab);
		}

		else if((!adjDireita(tab, pedida, tab.getR()))&&(!adjEsquerda(tab,
pedida))&&(!adjBaixo(tab, pedida,tab.getR()))&&(!adjCima(tab, pedida))){

			System.out.println("Não tem adjacentes \n");
			startjogo(tab);
		}
		else{
			listaRemover.add(pedida);q

			int f=0;
			while(true){
				if (f==listaRemover.size()) {
					break;
				}

				moviments(tab,listaRemover.get(f),r);
```

```
                    f=f+1;

                }

                tab.removerPecas(listaRemover);
                listaRemover.clear();

            }

            int n = pecasRestam(tab);

            score = r*r-n;
            jogada=jogada+1;


            if (terminarJogo(tab,r)) {
                terminarJogo(tab,r);
                System.out.println("Score "+score);
                break;
            }
        }
    }
```

Tem também o método *moviments*, em que adicionamos a uma lista a peça escolhida pelo utilizador, e as suas adjacentes, no caso destas serem iguais, terem a mesma cor, ou o mesmo número neste caso.
Um exemplo de como funciona este método é:

```
    public boolean adjDireita(Tabuleiro taba, Peca p, int r){

        Peca direita = taba.returnPeca(p.getLinha(), p.getColuna()+1);
        if (!verList(direita)) {
            if (p.getColuna()+1<r){ // se o x for menor que o tamanho lado
                //int [][] tab=taba.tabReturn();

                if (p.getTipo()==direita.getTipo()) {
                    //System.out.println("tem dir");
                    return true;
                }
                else
                    //System.out.println("N tem dir");
                    return false;}                    }
            else{
                //System.out.println("else dir");
                return false;
            }
        }else{
            //System.out.println("ja ta  na lista remover");
            return false;
        }
    }
```

```
public void moviments(Tabuleiro ta, Peca p, int r){
	int k=r;

	if(adjDireita(ta, p, k)){
		Peca direita = new Peca(p.getLinha(), p.getColuna()+1 , p.getTipo());
		listaRemover.add(direita);

	}
```

Classe *tabMain* serve para executar todas as outras classes, e pede também ao utilizador o número do lado do tabuleiro quadrado, e o número de cores diferentes com que queremos jogar.

```
class tabMain{
	public static void main(String args[]){

		while(true){
			Scanner sc= new Scanner(System.in);

			System.out.println("Numero do lado \n");
			int  nLado=sc.nextInt();


			System.out.println("Numero de cores diferentes \n");
			int  nCores=sc.nextInt();

			Tabuleiro taba = new Tabuleiro(nLado,nCores);

			Jogo jg= new Jogo();
			jg.startjogo(taba);
			System.out.println("Jogar outra vez ? 1-SIM, 0-NÃO");
			int again=sc.nextInt();
			if (again==0) {
				break;
			}
			else{
				continue; }
			}

		}
```

Conclusão

Este trabalho tem uma interface textual, na qual são pedidas ao utilizador as coordenadas, em primeiro lugar a linha, e em seguida a coluna, da peça a remover. Aceita apenas coordenadas de peças válidas, enviando uma mensagem de texto ao utilizador, para voltar a repetir a jogada, em caso contrário.

Conseguimos quase a totalidade das funcionalidades a implementar, excepto o movimento *empurrarPecas*, que não está a funcionar como pretendido, este deveria fazer com que as colunas se movam horizontalmente para a direita, na presença de uma coluna sem peças à sua direita.