

**Programação II**

**Relatório do Trabalho Prático**

Jogo Drench

1ºsemestre 2015/2016

Docente: Prof. Vitor Beires Nogueira

Alunos: João Dias nº35476

Eduardo Romão nº35477

Índice

Pág.

Introdução 3

1. Motor de Jogo
   1. Tabuleiro
   2. Jogada
   3. Baixar Peças
   4. Mover Colunas
   5. Fim do Jogo
2. Parte Grafica
3. Menu
4. Score
5. Main

Conclusão 10

# Introdução

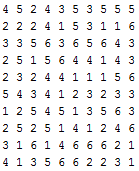
1. Motor de Jogo

Para o motor de jogo foi usado uma manipulação de um array de arrays.

* Tabuleiro

No tabuleiro base do jogo foi usado um array de arrays chamado grelha na classe Board. Este array irá ter o mesmo numero de linhas e colunas, que irão ser escolhidas pelo utilizador.

O preenchimento de tabuleiro é feito com números aleatórios de 1 a 6. Para isto foi importada a classe java.util.Random e foi usado um gerador com a raiz (seed) S, para poder jogar com tabuleiros iguais múltiplas vezes.



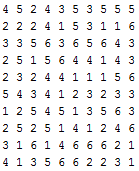
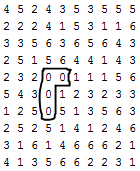
Exemplo de tabuleiro base

* Jogada

A jogada principal deste jogo consistia na escolha por parte do utilizador de uma peça para ser retirada, mas essa peça tinha que estar adjacente a pelo menos uma peça da mesma cor. Depois de essa peça ser retirada todas as peças adjacentes da mesma cor iriam ser retiradas, chamando a isto fecho transitivo da relação de adjacência.

De modo a realizarmos uma jogada no nosso tabuleiro criamos um método recursivo que recebe como argumentos o numero da linha, da coluna e o numero contido na posição onde começa a jogada. Este método irá percorrer peças em todas as direções até encontrar uma peça que seja diferente à inicial e ai voltara ao sitio onde estava. Cada vez que encontra uma peça no espaço de remoção torna a no numero 0. O numero 0 no nosso tabuleiro representa um espaço sem peça.

No seguinte exemplo começamos a nossa jogada nas coordenadas (5,4) e obtivemos o seguinte tabuleiro:

Para verificar que esta jogada e outras semelhantes podiam ser realizadas foi também desenvolvido um método na mesma base que o anterior para verificar que a peça onde se irá começar a jogar tem pelo menos uma adjacente da mesma cor. Este método chamado *verVizinhos* retorna um booleano, true se existir pelo menos uma peça adjacente igual e a partir dai sabemos que a jogada se pode realizar.

* Baixar Peças

Após peças serem removidas do jogo e ficar um “buraco” no tabuleiro, vem nas regras que as peças que estão por cima deste espaço irão cair na vertical para preencher os espaços das peças que desapareceram.

De forma a isto acontecer foram criados dois métodos, *baixar* e *verParaBaixar*. O primeiro método tem como função baixar as peças que tem um espaço em branco por baixo e o segundo tem como função procurar no tabuleiro espaços que tenham o valor 0, ou seja procura os espaços onde não existem peças.

Em *verParaBaixar* através de dois ciclos for o tabuleiro de jogo é todo percorrido e sempre que encontra uma coordenada a que esteja atribuído um zero, é chamado o método *baixar* para essas coordenadas. Nesse método iguala-se a respetiva coordenada ao valor que estava por cima e iguala-se o que está por cima a zero. Este método repete-se ate se chegar à primeira linha, ou seja, linha zero.

* Mover Colunas

Quando todas as peças de uma coluna tiverem desaparecido, ditam as regras que esse espaço será ocupado pela coluna à sua esquerda, como se tivesse uma mola no lado esquerdo do tabuleiro que empurra-se as peças. Para tal acontecer foram desenvolvidos quatro métodos na classe *Plays*.

O primeiro método, *VerColunaVazia* recebe como argumento o numero de uma coluna e retorna true se a coluna tiver vazia, ou seja, se apenas contiver zeros e retorna false caso contrario.

O segundo método,

* Fim do Jogo

Existem duas maneiras do jogo chegar ao fim, caso não hajam mais peças no tabuleiro ou então se não houverem mais jogadas possíveis e isto implica que não hajam peças da mesma cor adjacentes.

Para verificar se existem peças iguais adjacentes umas as outras criamos o método *fim* que recebe o array de arrays tabuleiro e através de dois ciclos for que percorrem o tabuleiro, vai usar a função da classe *Plays*, *verVizinhos*, e sempre que esta função retornar true entra-se num ciclo if que vai fazer o método retornar false indicando que existem pelo menos duas peças adjacentes da mesma cor.

De modo a verificar se o utilizador conseguiu “limpar” o tabuleiro criou-se o método *ganhou* que percorre todo o tabuleiro e se descobrir algum elemento do tabuleiro diferente de zero retorna false, indicando que o tabuleiro ainda contem pelo menos uma peça.