



Programação II

Relatório do Trabalho Prático

Jogo Smoothy
2º semestre 2015/2016

Docentes:	Prof. Salvador Abreu	
	Prof. ^a Lígia Ferreira	
Alunos:	João Dias	nº 35476
	Eduardo Romão	nº 35477

Índice

	Pág.
Introdução.....	3
1. Motor de Jogo.....	4
1.1 Tabuleiro	
1.2 Jogada	
1.3 Baixar Peças	
1.4 Mover Colunas	
1.5 Fim do Jogo	
2. Parte Grafica.....	7
3. Menu.....	8
4. Score.....	9
5. Main.....	10
Conclusão.....	11

Introdução

O objetivo deste trabalho prático consiste no desenvolvimento e implementação do jogo Smothy (versão simplificada do popular jogo “Candy Crush”) utilizando a linguagem de programação orientada a objetos, Java. Este jogo, que pode ser jogado apenas por um jogador, consiste na remoção de peças de um tabuleiro gerado aleatoriamente, até não se verificar a presença de mais peças ou não existirem mais jogadas possíveis, sendo que as peças apenas podem ser removidas se tiverem peças adjacentes da mesma cor.

Na realização deste trabalho foi decidido o uso de um array de arrays para o desenvolvimento do tabuleiro base e o uso de um array de arrays de botões para a parte gráfica.

Para o desenvolvimento deste jogo, inclusive a parte gráfica, foram aplicados os conhecimentos sobre a linguagem de programação Java adquiridos durante as aulas e recorrendo, adicionalmente, à pesquisa e estudo da documentação disponibilizada na internet.

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 número 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 utilizado um gerador com a raiz (seed) S, para poder jogar com tabuleiros iguais múltiplas vezes.

4	5	2	4	3	5	3	5	5	5
2	2	2	4	1	5	3	1	1	6
3	3	5	6	3	6	5	6	4	3
2	5	1	5	6	4	4	1	4	3
2	3	2	4	4	1	1	1	5	6
5	4	3	4	1	2	3	2	3	3
1	2	5	4	5	1	3	5	6	3
2	5	2	5	1	4	1	2	4	6
3	1	6	1	4	6	6	6	2	1
4	1	3	5	6	6	2	2	3	1

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 criámos um método recursivo que recebe como argumentos o número da linha, da coluna e o número contido na posição onde começa a jogada. Este método irá percorrer as peças em todas as direções até encontrar uma peça que seja diferente da inicial e aí voltará ao sítio onde estava. Cada vez que encontra uma peça no espaço de remoção torna-a no número 0. O número 0 no nosso tabuleiro representa um espaço sem peça.

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

4	5	2	4	3	5	3	5	5	5
2	2	2	4	1	5	3	1	1	6
3	3	5	6	3	6	5	6	4	3
2	5	1	5	6	4	4	1	4	3
2	3	2	4	4	1	1	1	5	6
5	4	3	4	1	2	3	2	3	3
1	2	5	4	5	1	3	5	6	3
2	5	2	5	1	4	1	2	4	6
3	1	6	1	4	6	6	6	2	1
4	1	3	5	6	6	2	2	3	1



4	5	2	4	3	5	3	5	5	5
2	2	2	4	1	5	3	1	1	6
3	3	5	6	3	6	5	6	4	3
2	5	1	5	6	4	4	1	4	3
2	3	2	0	0	1	1	1	5	6
5	4	3	0	1	2	3	2	3	3
1	2	5	0	5	1	3	5	6	3
2	5	2	5	1	4	1	2	4	6
3	1	6	1	4	6	6	6	2	1
4	1	3	5	6	6	2	2	3	1

Para verificar se esta jogada e outras semelhantes podiam ser realizadas foi, adicionalmente, desenvolvido um método na mesma base que o anterior para verificar se 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 deste ponto sabemos que a jogada se pode realizar.

- Baixar Peças

Após ocorrer a remoção de algumas peças do jogo surge um “buraco” no tabuleiro, ou seja, um espaço sem peças. De acordo com as regras do presente jogo as peças que estão por cima deste espaço irão cair na vertical para preencher os lugares que eram, anteriormente, ocupados pelas peças removidas.

Para que tal se verifique foram criados dois métodos, *baixar* e *verParaBaixar*. A função do primeiro método consiste em baixar as peças que possuem um espaço em branco por baixo e o segundo tem como função procurar no tabuleiro espaços que tenham o valor 0, ou seja, procurar os espaços onde não existem peças.

Em *verParaBaixar* através de dois ciclos *for* o tabuleiro de jogo é todo percorrido e sempre que é encontrada 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 até 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 existisse uma mola no lado esquerdo do tabuleiro que empurra as peças. Para tal acontecer foram desenvolvidos quatro métodos na classe *Plays*.

O método *PosicaoDaColunaVazia* vai percorrer uma a uma as colunas do tabuleiro tendo como objetivo encontrar uma coluna a zeros. Ao percorrer cada coluna o método auxilia-se de outros dois métodos, *VerColunaVazia* e *VerificarColuna*.

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.

VerificarColuna é o método que nos indica se a coluna foi ou não movida anteriormente nessa mesma jogada, ou seja, este método impede que as mesmas colunas se repitam, evitando assim um erro de um ciclo infinito. Para tal, recorreu-se a um ArrayList, com um auxilio de um contador (*cont*), que guarda a cada jogada as colunas movidas para a direita.

Encontrada a posição da coluna vazia, utiliza-se o método *moverDireita* para trocar a coluna à esquerda com a coluna encontrada. Recebendo assim como argumento a posição. Para efetuar a mudança, foi utilizado o auxilio de um ciclo *for*.

- Fim do Jogo

Existem duas maneiras do jogo chegar ao fim: inexistência de mais peças no tabuleiro ou inexistência de jogadas possíveis, o que implica que não se verifique a presença de peças adjacentes da mesma cor.

Para verificar se existem peças adjacentes iguais umas às outras criámos 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*. Sempre que

esta função retornar *TRUE* procede-se a um 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 foi desenvolvido 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 contém pelo menos uma peça.

2. Parte Gráfica

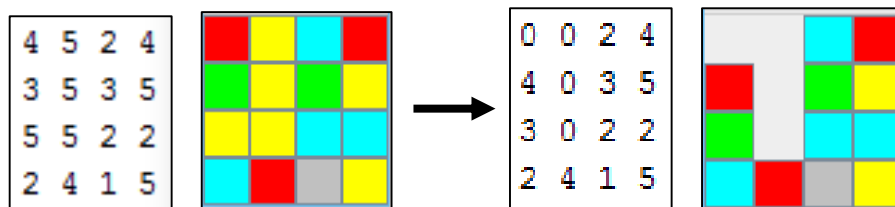
Na parte gráfica do trabalho para além de ser criado uma janela para o menu, foi também criado uma janela para o tabuleiro de jogo. Nesta última janela que vai ser abordada nesta parte do relatório, foi decidido fazer uma abordagem usando botões (JButton).

Na respetiva janela foi colocado um JPanel e foi decidido utilizar o GridLayout de modo a organizar um array de arrays de botões em forma de grelha. A janela vai ter um tamanho de 800x800 e um fundo branco. Também foi adicionado no topo da janela uma barra de ferramentas na qual se encontra um item que mostra ao utilizador o seu score atualizado.

De modo a colocarmos os botões na janela foram criados dois métodos, *buttonStart* (que é utilizado apenas uma vez, inicialmente, para colocar os botões no tabuleiro) e *buttonColor* (utilizado quando o tabuleiro é atualizado e a disposição dos botões é alterada).

Em *buttonStart* é recebido o tabuleiro em forma de array de arrays. Em seguida foram usados dois ciclos *for* onde é criado um botão para cada coordenada e adicionado ao JPanel, até todos os botões serem colocados. Também é adicionado a cada botão um *actionListener* para detetar cada clique em algum dos botões sendo que é atribuída uma cor a cada botão consoante o número presente no tabuleiro para a respetiva coordenada, usando o método *color*.

Este último método recebe como argumento o número presente no tabuleiro e o botão dessa coordenada. Consoante o número assim é atribuída uma cor ao botão, sendo que caso o número seja zero o botão torna-se invisível e caso contrário visível. Pode comprovar-se o que foi referido anteriormente no exemplo seguinte:



No exemplo acima apresentado pode verificar-se que quando no tabuleiro base temos um zero, o botão nessa coordenada fica invisível apesar de continuar na janela.

Após o utilizador executar uma jogada em vez de ser criado um novo tabuleiro com novos botões, é apenas utilizado o método *buttonColor* que percorre novamente o tabuleiro através de dois ciclos *for* e usando o método *color* altera a cor dos botões presentes na janela atualizando a visibilidade dos mesmos.

Nesta parte do trabalho para ser possível detetar alguma ação do utilizador, ou seja, um clique num dos botões, foi utilizado o método *actionPerformed* que recebe como argumento um evento. Este método atribui as variáveis *x* e *y* às coordenadas do botão carregado (sabe-se isto através do uso do *getSource()* para o evento). Adicionalmente, à variável *verdade* é atribuída o valor *TRUE* para confirmar que houve de facto uma ação por parte do utilizador.

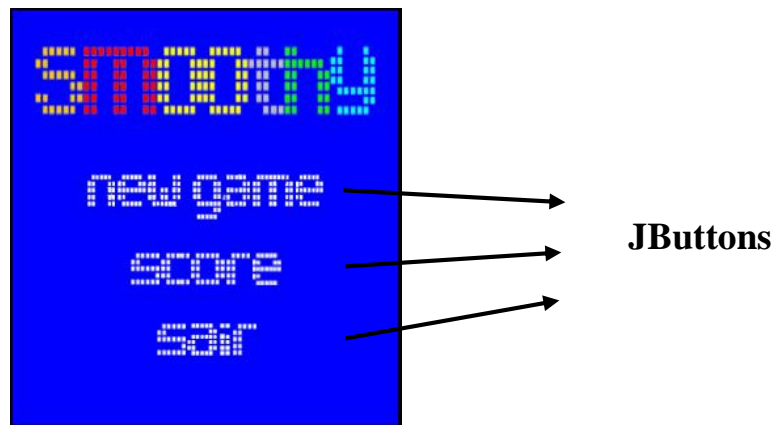
Após se ter as coordenadas do botão clicado em *x* e *y* através do método *valButton* colocam-se estes valores num array e retorna-se este array que vai, posteriormente, ser utilizado no *main* do programa para realizar a jogada.

3. Menu

Para a parte relacionada com o menu de jogo foi criado uma nova janela que possuirá três botões, um para iniciar um novo jogo, outro para consultar os recordes e por último um botão para sair e fechar o programa.

A janela possui uma dimensão de 300x325 e tem como fundo uma imagem que está guardada na directoria /resource e tem como nome fundo.png. Tendo a imagem um fundo transparente foi definido o azul como cor de fundo.

A posição de cada botão é definida de forma absoluta pois não foi utilizado nenhum layout e cada botão contém uma imagem que também está guardada na directoria /resource, para além de a cor de fundo ser transparente e de não ter bordas.



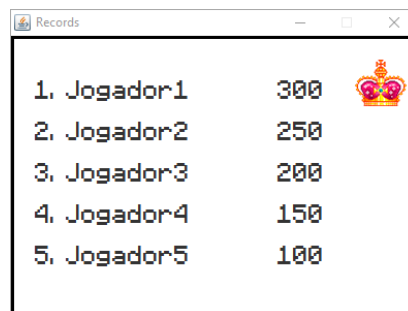
4. Score


Para o cálculo da pontuação de jogo do utilizador foi-nos pedido que utilizássemos a fórmula $(tamanho \times tamanho) - peças\ no\ tabuleiro$. Contudo, chegámos à conclusão que esta equação seria algo injusto pois o que contava na pontuação era apenas o tamanho do tabuleiro e não o número de cores diferentes que é o que realmente demonstra o grau de dificuldade de jogo. Por isso foi utilizada a seguinte equação $((tamanho \times tamanho) - peças\ no\ tabuleiro) \times número\ de\ cores$. Esta equação é definida no método *score* na classe *EndGame* e usa dois ciclos *for* para percorrer o tabuleiro e contar as peças restantes.

Para o utilizador visualizar os melhores resultados obtidos e registar novos recordes usamos como auxiliar o método *readFile*. Este método coloca numa lista os cinco melhores resultados que se encontram no ficheiro .txt *ScoreRecord*.

ShowScore é o principal método usado para visualizar a janela com os recordes. Esta janela, com o auxílio do componente swing, dispõe os resultados e os respetivos nomes. Através do *GridLayout* dispomos a informação mais organizada, deixando uma imagem .gif ao lado do maior score.

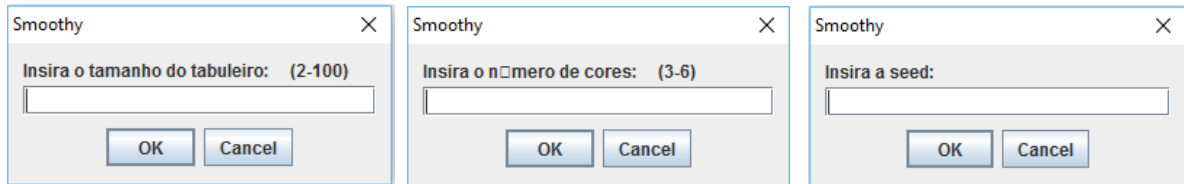
No final do jogo de modo a verificar se o resultado obtido está entre os cinco melhores, e deve ser escrito no ficheiro .txt, foi criado o método *record*. Este método recebe o novo resultado como argumento e compara-o com o menor resultado presente no ficheiro. Se for maior que o pior significa que vai dar entrada nos cinco melhores resultados e tem de ser escrito. Para tal, foi utilizado um *bufferwriter* que ao inicializar apaga todo o ficheiro. Após ser escrita a nova pontuação na posição cinco, através de um ciclo *while* vai se verificando se a pontuação superior é mais pequena e se tal for o caso troca-se, assim sucessivamente.



1. Jogador1	300	
2. Jogador2	250	
3. Jogador3	200	
4. Jogador4	150	
5. Jogador5	100	

5. Main

A *main* do nosso jogo está contida na classe *Body*. No construtor da classe é pedido ao utilizador o tamanho do tabuleiro, o número de cores e a seed do jogo, com o auxílio de JOptionPane.



Na *main* começa-se por chamar o menu criando um ciclo que se repete até uma das opções no menu ser selecionada. Para isto optámos por uma estratégia que consiste no retorno de booleanos ao clique nos botões.

Ao clicar no botão Score o *ActionPerformed* coloca a variável *scr* com o valor *TRUE* o que vai provocar a entrada do ciclo *if* respetivo no *main*. A mesma estratégia é usada para a seleção de um novo jogo, mas usando a variável *verdade*.

Dentro do *if*, respetivo a um novo jogo, é criado um objeto para todas as outras classes e um novo tabuleiro é gerado incluindo a sua respetiva representação gráfica, usando sempre os parâmetros introduzidos pelo utilizador.

Após as condições iniciais serem colocadas na área de jogo, o programa entra num ciclo que é repetido até o jogo terminar ou ser fechado. Durante o jogo, enquanto o botão respetivo à jogada a efetuar não for clicado a função fica a aguardar dentro de outro ciclo. Quando um dos botões permitidos é clicado, a função sai do ciclo e executa uma jogada a partir da respetiva coordenada do botão.

Dando por terminado o jogo, quando a janela de jogo desaparece, é pedida a introdução do nome do utilizador caso o score seja um recorde e volta a aparecer a janela do menu onde o utilizador pode clicar no botão Sair para encerrar o jogo.

Conclusão

Finalizado o trabalho podemos concluir que atingimos todos os objetivos, inclusive a implementação da parte gráfica do jogo. Conseguimos colocar em prática grande parte da matéria lecionada durante o semestre e pesquisar novas práticas que enriqueceram o nosso conhecimento na linguagem de programação Java.

Tendo também verificado, a existência ou não de erros no nosso programa, pensamos ter alcançado um código sem ou quase sem erros.

De notar que no sistema Linux o nosso programa não funciona, pois, o Linux não suporta a imagem .gif presente na janela dos recordes, o que faz com que o programa dê erro.