Instituto Politécnico de Beja
Escola Superior de Tecnologia e Gestão
Licenciatura em Engenharia Informática
Programação Orientada por Objectos

Quatro em linha

Trabalho Prático de Avaliação 1

Beja, 8 de Abril de 2019

So I went from day to day
Though my life was in a rut
'Til I thought of what I'd say
Which connection I should cut
– Solsbury Hill – Peter Gabriel

https://www.youtube.com/watch?v=_OO2PuGz-H8

1 Introdução

Este enunciado é algo extenso. Tal significa que deve ser lido mais do que uma vez e com muita atenção. O que é pedido está classificado em três tipos de **REQUISITOS**:

1. **ESSENCIAIS**
2. **NÃO ESSENCIAIS**
3. **QUE IMPLICAM PENALIZAÇÕES**.

Os essenciais permitem obter 10 valores, mas apenas desde que não estejam presentes motivos para penalizações, ou seja, se cumprir totalmente todos os requisitos que podem implicar penalizações. Para não ter penalizações basta ter cuidado a ler o enunciado e respeitar o que é requerido. Assim, poderá garantir que o programa entregue não irá oferecer motivos para a aplicação de penalizações, as quais podem fazer baixar a nota para níveis certamente indesejados. Os requisitos não essenciais permitem obter até 17 valores. Os 18, 19 e 20 valores ficam apenas para quem fez tudo o que é pedido de forma absolutamente impecável e inventou algo mais para fazer, os chamados **EXTRAS**. Esses extras têm de ter uma dificuldade superior ao que é pedido embora devam continuar na linha do que é pedido e utilizando apenas os conteúdos que já foram dados nas aulas. Não vale a pena fazer extras sem ter TUDO o que é pedido feito. Também não vale a pena fazer os requisitos não essenciais sem ter feito os essenciais. Em resumo:

- Só pode ter mais do 10 valores se tiver cumprido totalmente os requisitos essenciais e não tiver penalizações.
- Só pode ter mais do 17 valores se tiver cumprido totalmente os requisitos essenciais e também os não essenciais e não tiver penalizações.

Para mais informações pode e deve ler as regras de avaliação no guia de funcionamento da unidade curricular.

2 O Jogo

Neste trabalho prático, pretende-se ficar com um programa que permita jogar o "quatro em linha", ou *connect four* ou *four in a row* em inglês. O jogo está bem descrito na wikipedia: `https://en.wikipedia.org/wiki/Connect_Four`. Resumidamente, eis as regras:

1. É um jogo para dois jogadores; cada um coloca peças de uma cor;
2. Os jogadores jogam alternadamente colocando uma peça;
3. É jogado num tabuleiro vertical constituído por uma grelha de seis linhas e sete colunas;
4. Cada um coloca as suas peças numa das colunas com uma ou mais posições vazias;
5. Se a coluna onde se coloca a peça estiver vazia, esta cai até ficar na linha mais abaixo; se a coluna estiver cheia, não é possível colocar a peça; se a coluna tiver já uma ou mais peças, a peça jogada cai até ficar junto à primeira posição já ocupada;
6. O jogador que conseguir colocar 4 peças suas em linha, horizontal, vertical ou diagonal, ganha;
7. O jogo termina com a vitória ou quando já não for possível colocar mais peças no tabuleiro.

O que se recomenda é que faça todo o código do zero, embora consultando o código fornecido para o jogo do galo (*Tic-Tac-Toe*).

Seguem-se os requisitos para o trabalho. Antes de escrever o código, leia-os atentamente mais do que uma vez. Antes de entregar o trabalho, leia-os, pelo menos, mais uma vez.

3 Requisitos Essenciais

O incumprimento de qualquer destes requisitos implica uma classificação negativa no trabalho e a não contabilização dos requisitos não essenciais.

Req. 1 - Classes a definir (1,0 valores) A implementação do jogo deve respeitar os seguintes pontos:

1. O tabuleiro deve ser definido por um objecto de uma classe `pt.ipbeja.po2.connectfour.gui.Board` que sabe indicar qual a posição de cada botão;
2. Cada botão deve ser de uma classe `pt.ipbeja.po2.connectfour.gui.CellButton`;
3. O modelo do jogo deve estar numa classe `pt.ipbeja.po2.connectfour.model.ConnectFourModel` ;
4. A objecto da classe `pt.ipbeja.po2.connectfour.model.ConnectFourModel` deve ter um array de arrays do tipo `pt.ipbeja.po2.connectfour.model.Cell`;
5. O tipo `pt.ipbeja.po2.connectfour.model.Cell` deve ser um tipo enumerado com os valores `PLAYER1`, `PLAYER2`, `EMPTY`;

Req. 2 - Código de teste (5,0 valores — 0,625 por método de teste) Escrever código de teste para os seguintes cenários; cada um dos métodos deve ter o nome `test0N` em que `N` é letra do cenário na seguinte lista (teste01, teste02, etc); note que este código de teste apenas utiliza/testa código na *package* `pt.ipbeja.po2.connectfour.model`:

1. Colocação de uma peça numa coluna vazia; a peça deve ficar na linha de baixo (linha cinco);
2. Colocação de uma peça numa coluna contendo já uma peça na linha quatro, a segunda linha a contar de baixo; sendo a linha de baixo a linha cinco, a nova peça deve ficar por cima da que lá estava, portanto na linha três;
3. Tentativa de colocação de uma peça numa coluna cheia; o modelo deve ficar igual;
4. Uma jogada em que o jogador não ganha; esta é a situação mais frequente;
5. Uma jogada em que o jogador ganha com uma linha de quatro na horizontal;
6. Uma jogada em que o jogador ganha com uma linha de quatro na vertical;
7. Uma jogada em que o jogador ganha com uma linha de quatro na diagonal;
8. Uma jogada em que o jogo termina sem vitória; note que neste caso o modelo do teste deve ser iniciado só com uma posição `EMPTY` na linha de cima (linha zero) e depois realizar uma jogada que resulte em empate.

Req. 3 - Apresentação dos Movimentos(1,0 valores) Deve ser possível ver o tabuleiro com peças de duas cores e indicação das peças vazias;

Req. 4 - Implementação dos cenários testados (3,0 valores) os cenários testados no requisito 2 devem ser suportados também pela interface permitindo que dois jogadores utilizem o programa para jogar de acordo com as regras apresentadas no início do enunciado.

4 Requisitos Não Essenciais

Estes requisitos só são contabilizados se os requisitos essenciais estiverem totalmente correctos.

Req. 5 - Apresentação de Pontuação e Novo Jogo (2,0 valores) A pontuação é definida como a quantidade de posições no tabuleiro ($6 \times 7 = 42$) menos as jogadas feitas pelo vencedor até ganhar. Por exemplo se jogou 5 vezes a pontuação será $42 - 5 = 37$. Quando ganha, o vencedor tem de indicar um nome. Esse nome deve ter um máximo de 8 caracteres. No final de cada jogo, deve surgir numa nova caixa de diálogo do tipo `Alert` a pontuação conseguida e a lista das 3 maiores pontuações até à data. Se a pontuação conseguida fizer parte dessa lista, deve surgir assinalada com `***` logo após os pontos. Estas pontuações apenas existem enquanto se joga pelo que deve ser possível iniciar novo jogo utilizando um item "New" num menu "Game".

Req. 6 - Variante (2,5 valores) Implemente a variante "Pop Out" descrita em `https://en.wikipedia.org/wiki/Connect_Four`. Nesta variante, se um jogador tem uma peça sua no lado de baixo da coluna, pode optar por um segundo tipo de jogada: retirar essa peça o que fará com que todas as peças que estão por cima deslizem uma posição para baixo. Deve ser possível optar pelo jogo normal ou por esta variante selecionando um item "Pop Out" no menu "Game", mesmo durante a execução do jogo. Para tal, esse item deve ser do tipo `javafx.scene.control.CheckMenuItem`. Para um exemplo, veja, por exemplo, o vídeo em `https://www.youtube.com/watch?v=qZNqREHSrC0`.

Req. 7 - Undo e Redo (2,5 valores) Adicione a possibilidade do jogo recuar e avançar clicando em itens "Undo" e "Redo" no menu "Game". Para tal, o jogo deve ir mantendo a lista de jogadas efectuadas.

5 Requisitos que podem implicar penalizações

O incumprimento de um ou mais dos seguintes requisitos implica a atribuição da penalização especificada e a automática impossibilidade de obter uma nota superior a 17.

Req. 1 - *View/controller* (*gui*) separada do *model* (-1,0 a -4.0 valores) A *view/controller* (*gui*) apenas deve tratar de comunicar ao *model* o que o jogador fez (por exemplo um clique num botão, uma opção de menu, etc.) e fazer na interface gráfica as alterações pedida pelo *model*. Cada view também pode utilizar *getters* do *model* para obter informação. Toda a informação e lógica do jogo deve estar presente no *model*.

Req. 2 - Packages (-1,0 valores) Todo o código deve estar definido num *package* com o nome `pt.ipbeja.estig.po2.connectfour`. O código de interface deve estar num *package* com o nome `pt.ipbeja.estig.po2.connectfour.gui`. O restante código deve estar num *package* com o nome `pt.ipbeja.estig.po2.connectfour.model`.

Req. 3 - Métodos com mais de 30 pontos e vírgulas (-2,0 a -4,0 valores) Nenhum método deve ter mais de trinta pontos e vírgulas (";"). Note que um ciclo `for` tem dois pontos e vírgulas. Deve preferir métodos pequenos. Deve optar por mais métodos pequenos em lugar de menos métodos grandes.

Req. 4 - Utilização do operador `instanceof` ou do método `getClass` (-4,0 valores) A utilização do operador `instanceof`, do método `getClass` (`https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#getClass--`), ou algo idêntico que permita saber o tipo de dados de um objecto, será penalizado.

Req. 5 - Regras de estilo e elegância do código (-1,0 a -4,0 valores) O código entregue deve respeitar as regras de estilo, nomeadamente **todas** as seguintes:

Utilização de asserts **Sempre que conveniente, os métodos devem utilizar asserts para testar os valores dos parâmetros ou valores de outras variáveis.**

Identificadores em inglês Os nomes de todas as variáveis, métodos e classes devem estar em inglês.

Nomes das variáveis, métodos, constantes e classe Utilização de letras minúsculas/maiúsculas e informação transmitida pelos nomes; por exemplo, `box` é um melhor nome para uma caixa do que `b` ou `xyz`.

Constantes Não deve utilizar constantes literais (por exemplo, 20, -300, 45.4) para representar valores que podem ser melhor representados por um nome. Nesses casos deve definir constantes utilizando a palavra reservada **`final`**.

Os espaçamentos Depois das vírgulas e antes e depois dos operadores.

Indentação coerente e para cada bloco, incluindo posicionamento e utilização coerente das chavetas.

Utilização de parâmetros Sempre que conveniente, os métodos devem utilizar parâmetros de modo a evitar duplicação de código.

Repetição de código Deve ser evitada a repetição de código.

Comentários Antes de cada método deve escrever um comentário javadoc (`/** */`) que explique o que o método faz, os respectivos parâmetros e valor devolvido (se existentes). Os comentários podem estar em português mas tente colocá-los em inglês. Os comentários têm de incluir *tag*s `@param` para cada parâmetro e `@return` quando necessário.

Utilização do `this` Utilização das referências antes do nome das operações. Por exemplo, **`this`**`.add(line)`.

Ocultação de informação Todos os atributos devem ser `private`.

Req. 6 - Auto-avaliação (-2,0 a 0,0 valores) Num ficheiro "auto-aval.txt", deve indicar quais os requisitos que estão **totalmente** cumpridos (os únicos que contam como cumpridos) e a classificação resultante. No mesmo ficheiro **deve indicar quantas horas gastou a fazer este trabalho, incluindo o tempo em aulas e fora das aulas (trabalho autónomo)**.

Req. 7 - Identificação (-1,0 valores) Todos os ficheiros com código (ficheiros *.java) têm de conter, em comentário no início, o nome e número de aluno do autor.

Req. 8 - Quantidade de autores (-20,0 valores) O trabalho deve ser realizado **INDIVIDUALMENTE**, apenas pelo aluno que submete a resolução.

Req. 9 - Nome do projecto (-1,0 a 0,0 valores) O nome do projecto no inteliJ tem de respeitar o seguinte formato. Note que `Primeiro` e `Ultimo` representam o nome do autor. `Numero` representa o número de aluno do autor ou com origem noutras fontes.

```
Numero_PrimeiroUltimo_TP01_PO2_2018-2019
Por exemplo: 1232_VanessaAlbertina_TP01_PO2_2018-2019
```

O trabalho é entregue compactando a directoria do projecto eclipse num ficheiro .zip de forma a que este fique com o nome `Numero_PrimeiroUltimo_TP01_PO2_2018-2019.zip`.

Req. 10 - Originalidade (-20,0 a +2,0 valores) A originalidade das soluções encontradas para resolução dos requisitos essenciais e não essenciais, comparativamente com outros trabalhos entregues ou disponíveis na internet, pode ser valorizada num máximo de 3 valores e penalizada num máximo de 20,0 valores. Para efeitos de aplicação desta valorização ou penalização, a originalidade é determinada pelas diferenças ou semelhanças entre o trabalho a ser avaliado e os restantes trabalhos entregues por outros alunos.

Req. 11 - Entrega e apresentação do trabalho (-20,0 a 0,0 valores) O trabalho entregue tem de ser uma directoria do projecto IntelliJ pronto a funcionar, sob a forma de um único ficheiro zip contendo toda a directoria mais o ficheiro de auto-avaliação. Antes de entregar, verifique que sabe pôr a funcionar o código no ficheiros zip entregue. Para tal parta desse ficheiro, descompacte-o, e leia-o no IntelliJ. Tal poderá ser requerido na apresentação individual do trabalho. Se não conseguir pôr a funcionar o conteúdo do ficheiro zip entregue (no moodle e por email) a classificação no trabalho poderá ser de zero valores. A entrega tem de ser feita num ficheiro no formato zip com o nome indicado no Requisito 9 e sempre por **duas** vias:

1. Na página da disciplina.
2. Por *email*, respeitando as seguintes regras: A entrega por *email* é feita para `trabalhos.p2ARROBAgmail.com`. O *email* a enviar deve conter em *attach* um único ficheiro no formato *zip*. O *subject* do *email* deve ser o `nomeDoProjecto`.
3. Pode entregar mais do que uma vez, desde que dentro do prazo. A última entrega dentro do prazo é a única que conta.

Req. 12 - Data limite de entrega em época normal O trabalho deve ser entregue no moodle e também por email até às **23:55 de 6 de Maio de 2019**. Não deve assumir que o relógio do sistema está igual a qualquer outro. Assim, a entrega depois da uma hora (da madrugada) de 7 de Maio de 2019 implicará zero valores no trabalho.

Finalmente, note que necessita de realizar mais do que o pedido para obter mais de 17 valores. A criatividade também pode justificar uma melhor classificação pelo que extras originais e sofisticados serão uma boa aposta. Note que estas adições só contam para a classificação do trabalho se forem consideradas suficientemente significativas e se **todos** os requisitos estiverem completamente cumpridos e sem penalizações.

Lembre-se que a originalidade (Req. 10 do 5) é outra forma de subir a pontuação e contribuir para obter mais do que 17 valores

6 Nota importante

Todas as contribuições para o trabalho que não sejam da exclusiva responsabilidade dos autores têm de ser identificadas (com comentários no código e referências no relatório) e a sua utilização bem justificada e expressamente autorizada pelo professor responsável. Justificações insatisfatórias, ausência de autorização, ou ausência de referências para trabalhos ou colaborações externas utilizadas serão consideradas fraude sempre que o júri da unidade curricular considere os trabalhos demasiado semelhantes para terem sido criados de forma independente e **tal terá como consequência a reprovação direta na unidade curricular de todos os alunos envolvidos**. Assim, nenhum aluno deve dar cópia do seu código (ainda que em fase inicial) a outro. Por essa mesma razão não é boa ideia partilhar código com os colegas, quer directamente quer através do fórum. Naturalmente, cada aluno pode e deve trocar impressões e esclarecer dúvidas com todos os colegas, mas deve saber escrever todo o código sozinho. Lembre-se que será avaliado em testes prático em frente a um computador. A classificação neste trabalho prático fica dependente de uma eventual apresentação individual do mesmo, tal como previsto no guia da unidade curricular e pode ser alterada em resultado do desempenho do aluno nessa mesma apresentação.

Caso o júri detecte algum tipo de ocorrência que considere anómala, a defesa do trabalho poderá ser anulada sendo repetida e contabilizada apenas esta segunda defesa.

Finalmente, antes de entregar leia com MUITA atenção todo o enunciado. A falha de parte do exigido num requisito implica o não cumprimento desse requisito e consequente penalização. A programação é também a atenção aos detalhes.

Bom trabalho!

João Paulo Barros