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

A Good Snowman Is Hard To Build - PO2 edition

Trabalho Prático de Avaliação

Beja, 14 de abril de 2025 (revisto em 22 de abril de 2025)
Este enunciado pode sofrer pequenas alterações até à data limite de entrega.
Deve estar atento ao Teams

Look around
Leaves are brown
There's a patch of snow on the ground
– A Hazy Shade of Winter, Paul Simon
https://www.youtube.com/watch?v=yB1oT3d0b18

https://www.youtube.com/watch?v=TxrwImCJCqk

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;
- **Pequenas** falhas ou omissões nos requisitos essenciais podem, eventualmente, ser compensadas por requisitos não essenciais correta e totalmente implementados.

2 O Jogo

Neste trabalho prático, pretende-se ficar com um programa que permita jogar uma versão simplificada do jogo *A Good Snowman Is Hard To Build*, um jogo de puzzle desenvolvido por Alan Hazelden, lançado em 2015. As regras principais são as seguintes:

Manipulação de bolas de neve O jogador controla um monstro preto descaracterizado cujo objetivo é construir bonecos de neve. Para isso, faz rolar bolas de neve de diferentes tamanhos num ambiente com neve;

Movimento O monstro empurra bolas de neve num ambiente em grelha (*tile-based*). Fazer rolar uma bola de neve pequena sobre uma posição com neve transforma-a numa bola de neve média. Fazer rolar uma bola de neve média sobre a neve transforma-a numa bola de neve grande. As bolas de neve grandes continuam a ser grandes, independentemente da quantidade de neve sobre a qual rolam.

Construção do boneco de neve Um boneco de neve completo requer três bolas de neve empilhadas por ordem decrescente de tamanho: grande em baixo, média no meio e pequena em cima.

Conceção do puzzle Os puzzles giram em torno de descobrir a sequência correta de movimentos para obter as bolas de neve com os tamanhos certos e nas posições certas para serem empilhadas. Isto implica planear cuidadosamente os movimentos para evitar tornar uma bola de neve demasiado grande prematuramente ou deixá-la ficar presa.

Desfazer e reiniciar O jogo é tolerante, oferecendo *undo* (desfazer jogada) e também a possibilidade de reiniciar para recomeçar o puzzle atual. Tal incentiva a experimentação e a aprendizagem com os erros.

Estes são alguns links para conhecer melhor o jogo:

- `https://www.youtube.com/watch?v=uWJIkfiKmQk`
- `https://agoodsnowman.com/`
- `https://en.wikipedia.org/wiki/A_Good_Snowman_Is_Hard_to_Build`).

O jogo que se pretende implementar deve ter as seguintes características:

1. O jogo decorre numa grelha em que cada uma das posições é um `PositionContent` com um dos seguintes valores:

- `NO_SNOW` (chão livre sem neve);
- `SNOW` (chão livre com neve);
- `BLOCK` (zona intransponível);
- `SNOWMAN` (uma bola pequena em cima de uma bola média que está em cima de uma bola grande, ou seja três bolas empilhadas);

2. O tabuleiro de jogo deve ser indicado pelo jogador e devem existir pelo menos dois tabuleiros de jogo (dois níveis), cada um com (1) as suas posições com um dos valores listados no número anterior, (2) a posição das bolas de neve e (3) a posição do monstro;
3. Há três tamanhos de bola de neve mais a possibilidade de empilhamento de uma bola mais pequena em cima de uma bola maior;
4. O monstro tem de empurrar bolas de neve para fazer um boneco de neve com três bolas de neve empilhadas, com a maior em baixo, a média no meio e a mais pequena no topo;
5. Empurrar uma bola de neve para cima de uma posição com neve cria uma bola de neve maior, mas uma bola de neve grande nunca fica maior mesmo que passe por uma zona de neve;
6. Não é possível empurrar uma bola de neve para cima de uma bola de neve menor nem para cima de uma posição intransponível;
7. Pode empurrar uma bola pequena que esteja em cima de uma bola média ou grande; a bola pequena ficará na posição seguinte;
8. Pode empurrar uma bola média que esteja em cima de uma bola grande; a bola média ficará na posição seguinte;
9. Se empurrar uma bola pequena para cima de uma bola média que está já em cima de uma bola grande fará um boneco de neve e terminará o nível de jogo;
10. As bolas de neve devem ser objetos da classe `Snowball` que sabem em que linha e coluna estão no tabuleiro e também qual o seu tamanho;
11. As bolas de neve devem ser objetos da classe `Snowball` que sabem em que linha e coluna estão no tabuleiro e também qual o seu tipo; o empilhamento de bolas de neve também pode ser visto como um tipo de bola de neve;
12. O tipo de bola de neve pode ser indicado por um tipo enumerado com os valores `BIG`, `AVERAGE`, `SMALL`, `BIG-AVERAGE` `BIG-SMALL` e `AVERAGE-SMALL`; mas também pode utilizar uma classe para cada um destes tipos;
13. O monstro deve ser um objeto da classe `Monster` que sabe em que linha e coluna está no tabuleiro.

Para fazer a sua versão do jogo tem obrigatoriamente de basear o seu código no *template* `JavaFX-PO2-2023-2024` que tem sido utilizado nas aulas, respeitar a separação `ui` e `model` e a forma de comunicação entre as *packages* `model` e `ui` através da interface `View`. Caso não respeite a estrutura do *template* será avaliado com zero valores neste trabalho prático.

Mesmo assim, note que ainda sobra muito espaço para a sua criatividade!

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.

Nos requisitos seguintes, o movimento do monstro deve ser programado indicando a direção para onde o monstro pretende deslocar-se, por exemplo, para ir para a direita será `game.moveMonster(Direction.RIGHT);`. Note que cada uma nos pontos nos requisitos seguintes pode implicar várias linhas de código.

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 objeto de uma classe `BoardModel` que contém o `board`: `List< List<PositionContent> > board`.
2. O tipo `MobileElement` deve ser definido por uma classe abstracta que contém a posição.
3. Cada objeto do tipo `PositionContent` deve ter um dos seguintes valores: `NO_SNOW`, `SNOW`, `BLOCK` ou `SNOWMAN`;
4. As bolas de neve devem ser objetos da classe `Snowball` que sabem em que linha e coluna estão no tabuleiro, com os tamanhos e tipos já referidos;
5. O monstro deve ser um objeto da classe `Monster` que sabe em que linha e coluna está no tabuleiro;

Req. 2 - Testes a definir (4,5 valores) Deve ter um método de teste para os seguintes cenários, em que os métodos de teste devem ter o nome indicado:

void testMonsterToTheLeft() Testar movimento do monstro para uma posição livre para a esquerda: completar `testMonsterToTheLeft();`

void testCreateAverageSnowball() Testar criação de uma bola de neve de tamanho médio, empurrando uma bola pequena sobre uma posição com neve;

void testCreateBigSnowball() Testar criação de uma bola de neve de tamanho grande, empurrando uma bola media sobre uma posição com neve;

void testMaintainBigSnowball() Testar que uma bola de neve de tamanho grande se mantém do mesmo tamanho mesmo quando empurrada sobre uma posição com neve;

void testAverageBigSnowman() Testar que empurrar uma bola de neve média para cima de uma posição com uma bola grande cria um boneco de neve incompleto.

void testCompleteSnowman() Testar que empurrar uma bola de neve pequena para cima de uma posição com uma bola grande e uma bola média cria um boneco de neve completo.

Req. 3 - Apresentação dos Movimentos(1,0 valores) Adicione num `Pane` abaixo do mapa e na mesma janela deste, um campo de texto onde são colocadas as jogadas efetuadas, por exemplo: `(2, A) -> (2, B)`. Para tal, o tabuleiro deve ter as letras a indicar as colunas e os números a indicar as linhas.

Req. 4 - Deteção de fim (0,5 valores) O jogo deve detetar que o boneco de neve foi criado. Nessa altura deve surgir uma caixa de diálogo que permite recomeçar o jogo nesse mesmo nível ou noutro diferente.

Req. 5 - Gravação dos movimentos para ficheiro (1,5 valores) Sempre que é construído um boneco de neve, deve ser gerado um ficheiro com o nome `snowmanAAAAMMDDHHMMSS.txt` em que `AAAAMMDDHHMMSS` indica a data e instante da gravação do ficheiro. O ficheiro deve conter o seguinte:

1. O mapa utilizado;
2. O conteúdo do painel referido no Req. 3;
3. A quantidade de movimentos efetuados;
4. A posição em que foi criado o boneco de neve.

Req. 6 - Imagens (1,5 valores) Adicione **imagens ou texto** na grelha de forma visualizar os diversos tipos de posição, o boneco de neve completo e incompleto e o monstro. pode utilizar diretamente imagens, labels ou botões.

4 Requisitos Não Essenciais

Estes requisitos só são contabilizados se os requisitos essenciais estiverem totalmente corretos. Se existir uma resolução para todos esses requisitos e estas apresentarem falhas que os docentes avaliadores considerem pequenas, a resolução de requisitos não essenciais será valorizada até um máximo de 12 valores na classificação final do trabalho.

Req. 7 - Apresentação de Pontuação (2,0 valores) A pontuação é definida como a quantidade de movimentos efetuados, o nome do nível e o nome do jogador. Esse nome deve ter um máximo de 3 caracteres. No final de cada jogo, deve surgir num painel à direita do tabuleiro a pontuação conseguida, o nome do nível e a lista das 3 (ou menos se necessário) maiores pontuações até à data. Se a pontuação conseguida fizer parte dessa lista, deve surgir assinalada com `TOP` logo após os pontos. Para que a pontuação fique completa, o jogador tem de indicar o respectivo nome no início do jogo. Cada pontuação deve ser um objeto de uma classe com o nome `Score`.

Req. 8 - Leitura de Níveis (2,0 valores) Adicione a possibilidade de, em qualquer momento, o jogador abrir (ler) um nível de jogo à escolha de um ficheiro de texto, ou seja, um mapa e iniciar um novo jogo. O formato do ficheiro é livre mas tem de permitir fazer mapas com todos os tipos de `PositionContent` indicados no Req. 1.

Req. 9 - Música de fundo (1,0 valores) Adicione uma música de fundo que começa logo que surge o mapa. A música pode estar em *loop*.

Req. 10 - *Undo* e *Redo* (2,0 valores) Adicione a possibilidade do jogador recuar (*undo*) e avançar (*redo*) a jogada efetuada (repondo o estado do jogo que estava). Para tal, o jogo deve ir mantendo a lista de jogadas efetuadas. O *undo* e *redo* devem ser itens num menu. Deve ser possível fazer *undo* e *redo* repetidamente.

Exemplos de extras:

- Leitura de percursos animada — Adicione a possibilidade da leitura de ficheiros com o formato do Req. 5 e fazer *replay* do jogo, a mover sozinho o monstro.

- Imagens com boa qualidade e/ou outros melhoramento estéticos na interface;
- Escolha de entre vários aspetos gráficos para a interfce ("interface skins").

5 Adicional para entregas em época de recurso

Se entregar em época de recurso, todos os requisitos essenciais são os mesmos e com a mesma pontuação. No entanto, os requisitos não essenciais já listados serão desvalorizados 50% pelo que irão permitir um máximo de 3,5 valores. Para os restantes 3,5 valores (para totalizar $10+3{,}5+3{,}5=17$ valores) terá de completar os seguintes requisitos não essenciais adicionais:

Req. RECURSO-01 - Reprodução automática (2.0 valores) Quando é feito um boneco de neve, surge a opção de reproduzir de forma automática todos os movimentos do monstro, ou seja, poderá ser feito um *replay* automático do jogo;

Req. RECURSO-02 - Movimentos em diagonal (1,5 valores) O monstro também poderá realizar movimentos em diagonal e empurrar bolas em diagonal.

6 Requisitos que podem implicar penalizações e bonificações (RPB)

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.

RPB 1 - View/controller separada do Model Cada view/controller apenas deve tratar de comunicar ao *model* o que o jogador fez (por exemplo um clique num botão) e fazer na interface as alterações pedidas 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*. Deve ser possível remover o código relativo à interface (arrumado na `package pt.ipbeja.po2.snowman.gui`).

RPB 2 - Testes de tipos e polimorfismo (-2,0 a 2,0 valores) Não pode utilizar testes ao tipo dos objetos (por exemplo, utilizando o operador `instanceof` ou o método `getClass`). Em seu lugar, deve utilizar herança e polimorfismo com ligação dinâmica.

RPB 3 - Packages (-1,0 valores) Todo o código deve estar definido num *package* com o nome `pt.ipbeja.estig.po2.snowman`. O código de interface deve estar num *package* com o nome `pt.ipbeja.estig.po2.snowman.gui`. O restante código deve estar num *package* com o nome `pt.ipbeja.estig.po2.snowman.model`.

RPB 4 - Métodos com mais de 20 pontos e vírgulas (-4,0 a -2,0 valores) Nenhum método deve ter mais de **vinte** 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.

RPB 5 - Regras de estilo e elegância do código (-4,0 a 1,0 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`.

RPB 6 - Auto-avaliação (-2,0 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)**.

RPB 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.

RPB 8 - Quantidade de autores (-20,0 valores) O trabalho deve ser realizado **individualmente** OU em **grupos de dois**. Recomenda-se que seja feito em grupos de dois.

RPB 9 - Nome do projeto (-1,0 valores) O nome do projeto no IntelliJ 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_TP_PO2_2024-2025
Por exemplo: 1232_VanessaAlbertina_TP_PO2_2024-2025
```

O trabalho é entregue compactando a diretoria do projeto IntelliJ num ficheiro .zip de forma a que este fique com o nome `Numero_PrimeiroUltimo_TP_PO2_2024-2025.zip`.

RPB 10 - Originalidade (-20,0 a +3,0 valores) A originalidade das soluções encontradas para a 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 ou encontrados noutros locais.

RPB 11 - Entrega e apresentação do trabalho (até -20,0 valores) O trabalho entregue tem de ser projeto IntelliJ pronto a funcionar, sob a forma de um único ficheiro zip contendo toda a diretoria 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 RPB 9 e no moodle.

RPB 12 - Data limite de apresentação em época normal O trabalho deve ser entregue no moodle e também por email até às **23:59 de 6 de junho de 2025**. Não deve assumir que o relógio do sistema está igual a qualquer outro pelo que deve entregar sempre pelo menos 15 min ANTES do tempo limite. Assim, a não entrega até à 01:00 de ~~13~~ 7 de junho de 2025 implica zero valores no trabalho.

RPB 13 - Data limite de apresentação em época de recurso O trabalho deve ser entregue no moodle e também por email até às **23:59 de 1 de julho de 2025**. Não deve assumir que o relógio do sistema está igual a qualquer outro pelo que deve entregar sempre pelo menos 15 min ANTES do tempo limite. Assim, a não entrega até à 01:00 de 2 de julho de 2025 implica zero valores no trabalho.

RPB 14 - Código base (-20,0 valores a 0,0 valores) O código entregue tem obrigatoriamente de partir do código base fornecido.

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 também 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 (RPB 10) é outra forma de subir a pontuação e contribuir para obter mais do que 17 valores.

7 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 código (ainda que em fase inicial) a colegas, mas pode ajudar dizendo O QUE FEZ e COMO FEZ, embora SEM mostrar ou dar o código**. Nunca é boa ideia partilhar código com os colegas, quer diretamente 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 práticos 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.

A utilização de *LLMs* (ChatGPT ou outros softwares de inteligência artificial) tem de ser declarada no ficheiro de auto-avaliação e também em comentário identificando as partes do código que integram código feito por esses sistemas.

Caso o júri detete 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