



Welcome to FANDEISIA

Enunciado do projecto prático - Primeira Parte

Objectivo

Este projecto tem como objectivo o desenvolvimento de uma aplicação (programa) em **linguagem Java** (versão Java 8) aplicando os conceitos de modelação e Programação **Orientada a Objetos** (encapsulamento, herança, polimorfismo, etc.) e os conceitos de programação Funcional (*mapping*, *filtering*, *streams*).

O projecto está dividido em 3 partes:

- Primeira Parte - apresentada no presente enunciado, incide na modelação, e implementação do modelo e de um conjunto de funcionalidades associadas;
- Segunda Parte - onde se apresentam novos requisitos, que poderão (ou não) requerer a alteração do modelo submetido na Primeira Parte e implementação de novas funcionalidades.
- Terceira Parte - novamente com novos requisitos, aos quais os alunos deverão dar resposta usando conceitos e técnicas de Programação Funcional.

As 3 partes são de entrega obrigatória e terão prazos de entrega distintos. O não cumprimento dos prazos de entrega de qualquer uma das partes do projecto levará automaticamente à reprovação dos alunos na avaliação de primeira época da componente prática.

Objectivos - Primeira Parte

Nesta primeira parte, os alunos terão de:

- Criar um **diagrama de classes** em **UML** que seja suficiente para modelar a realidade apresentada e responder aos requisitos funcionais apresentados ao longo do enunciado.
- Implementar o modelo proposto, usando a linguagem **Java 8**.
- Implementar algumas funcionalidades, também em **Java 8**.
- Criar testes automáticos unitários para uma parte do modelo implementado, usando **JUnit**.

Restrições Técnicas - Primeira Parte

Nesta primeira parte do projecto, o modelo de dados (e respectiva implementação) **não pode usar** o mecanismo **de herança** e também **não pode usar** o mecanismo de **Interfaces**.

Tema

Numa galáxia relativamente perto da nossa existe um planeta chamado **FANDEISIA**. Neste mundo, criaturas de diversas espécies (tais como “anões”, “dragões”, etc.) interagem entre si e com o próprio mundo. O planeta **FANDEISIA** é governado por um nobre ditador tirano (mas benevolente) chamado **Lord Éder** ¹.

O **Lord Éder** gosta de desafiar as criaturas do seu mundo para pequenas competições onde estas podem ganhar prémios. Nos últimos tempos, o Lord Éder tem ficado insatisfeito com a qualidade das competições, queixando-se que as criaturas andam demasiado lentas e com falta de energia. Diz-se pelas ruas da Capital de FANDEISIA que a culpa é do próprio Lord Éder que exige que as criaturas compitam várias vezes por dia, não lhes dando o merecido descanso.

¹ Há quem lhe chame “O Herói de Paris” mas a história da origem dessa alcunha parece ter-se perdido no tempo.

Para não depender do estado das criaturas, o Lord Éder decidiu encomendar ao **DEISI** o desenvolvimento de um jogo para computador onde possa fazer o mesmo género de competição que faz com as criaturas habitantes do **seu** planeta.

O jogo irá funcionar num modelo “jogador humano vs computador” e irá decorrer num mapa bidimensional (2D). Quer o jogador, quer o computador, vão ter uma equipa formada por criaturas e vão poder **influenciar** os movimentos dessas criaturas usando **feitiços**. Durante o jogo as criaturas poderão encontrar tesouros que valem pontos. Vence o jogo a equipa que conseguir alcançar mais pontos. O jogo estará dividido em turnos, jogando as equipas alternadamente.

Cabe então aos alunos de **LP2** deste ano lectivo a implementação deste sistema.

Para que os alunos se foquem na correcta modelação do sistema de simulação, os Professores de **LP2** irão também dar uma perninha no projecto, fazendo a implementação de uma interface gráfica que os alunos terão de usar no seu projecto.

Nesta primeira parte do projecto, a interacção do jogador com o jogo vai ser relativamente curta, estando focada em:

- Criação do exército
- Processamento do turno

Domínio do Problema

Neste capítulo vamos descrever os conceitos envolvidos neste problema. A compreensão destes conceitos é essencial para a criação do modelo de classes do projecto.

As equipas

Neste jogo existem duas equipas:

- A equipa **LDR**, que é composta por criaturas controladas pelo Lord Éder; e
- A equipa **RESISTENCIA**, composta por outras criaturas.

A equipa **LDR** tem o ID 0 e a equipa **RESISTENCIA** tem o ID 1.

Os Tesouros

Os Tesouros são itens que existem “espalhados” pelo mundo e que podem ser apanhados pelas criaturas.

Cada Tesouro equivalente a um número de pontos.

Nesta primeira parte do projecto, todos os tesouros valem o mesmo número de pontos (= 1).

Estrutura dos Turnos do Jogo

- O jogo está dividido em turnos. Em cada turno, um dos jogadores é considerado o jogador activo.
- O primeiro jogador activo é o jogador que controla a equipa **LDR**.
- Em cada turno, o jogador activo tem uma fase de acção, durante a qual pode lançar zero ou mais feitiços. O jogador pode terminar o turno sem ter lançado nenhum feitiço.
- Após o jogador ter dado a sua fase de acção por terminada, devem ser calculados e aplicados os efeitos dos feitiços aplicados pelo utilizador.
- De seguida, todas as criaturas se movem ou tentam mover. As criaturas devem ser sempre processadas pela ordem do seu ID.
- Para cada criatura, deve-se:
 - Determinar as posições onde terão de passar considerando a posição onde estão, a sua orientação, e as particularidades do seu movimento.
 - Após determinar essas posições, deve-se verificar se o movimento é válido.
 - Se o movimento for válido, então deve ser aplicado, movendo a criatura e actualizando o mundo conforme for necessário;

Bruno Cipriano, Rodrigo Correia, Pedro Alves

- Se o movimento não for válido, a criatura deve-se **virar** de forma a mudar a orientação respectiva. Essa criatura não se move mais neste turno;

De seguida, passa-se à próxima criatura, até que tenham sido processadas todas as criaturas.

Este processo repete-se até que o jogo termine.

O jogo termina quando for atingida uma das seguintes três condições:

- Não existirem mais Tesouros para apanhar/capturar no mundo;
- Terem passado 15 turnos sem que nenhuma criatura tenha encontrado um tesouro;
- O número de tesouros/pontos que existem no mundo já não permitirem que a equipa que com menos pontos apanhe (iguale) a equipa com mais pontos

Cálculo do Vencedor

Nesta primeira parte do projecto, a equipa vencedora será aquela que tiver feito mais pontos, considerando os tesouros que foram apanhados pelas suas criaturas. Este cálculo deve ser feito quando for detectado o fim de jogo.

Caso o número de pontos seja igual, considera-se que o jogo terminou empatado.

Mapa do Mundo

O mapa é um rectângulo $M \times N$. O tamanho do mapa poderá variar de simulação para simulação.

Cada posição do mapa é identificada por um par de coordenadas (x,y), sendo a origem do mapa a posição que ocupa o canto superior esquerdo do mesmo. Os valores de **x** irão crescer na horizontal para a direita e os valores de **y** irão crescer na vertical para baixo.

No seguinte exemplo demonstram-se as coordenadas de cada posição num mapa 3 x 4:

0,0	1,0	2,0	3,0
0,1	1,1	2,1	3,1

0,2	1,2	2,2	3,2
-----	-----	-----	-----

Nesta primeira fase do projecto, cada posição do mapa poderá conter:

- Um espaço vazio
- Uma criatura
- Um tesouro

Nota importante: Uma posição do mapa que inicialmente esteja ocupada por uma criatura fica implicitamente vazia caso essa criatura saia do espaço respectivo durante a simulação.

Criaturas

Todas as **criaturas** são caracterizadas por:

- um ID - número inteiro positivo que identifica a criatura no simulador;
- um tipo - uma String (p.e. “Dragão”);
- a sua posição no mundo, representada por um par de coordenadas x,y;
- a sua orientação - “Norte”, “Sul”, “Este” e “Oeste”;

Regras de Movimento - Parte 1

Uma **criatura** move-se no mundo seguindo as seguintes regras:

- O movimento é feito considerando a orientação da criatura;
- A criatura move-se **uma posição de cada vez**;
 - Por exemplo, uma criatura virada para “Este” que esteja em (2, 2) terá tendência a tentar mover-se para a posição (3, 2);
- Quando um movimento é impossível, a criatura vira-se no sentido dos ponteiros do relógio;
- Quando uma criatura se vira, efectua uma volta de 90 °.
 - Exemplos:
 - uma criatura que esteja virada para “Norte” vai virar-se para “Este”.

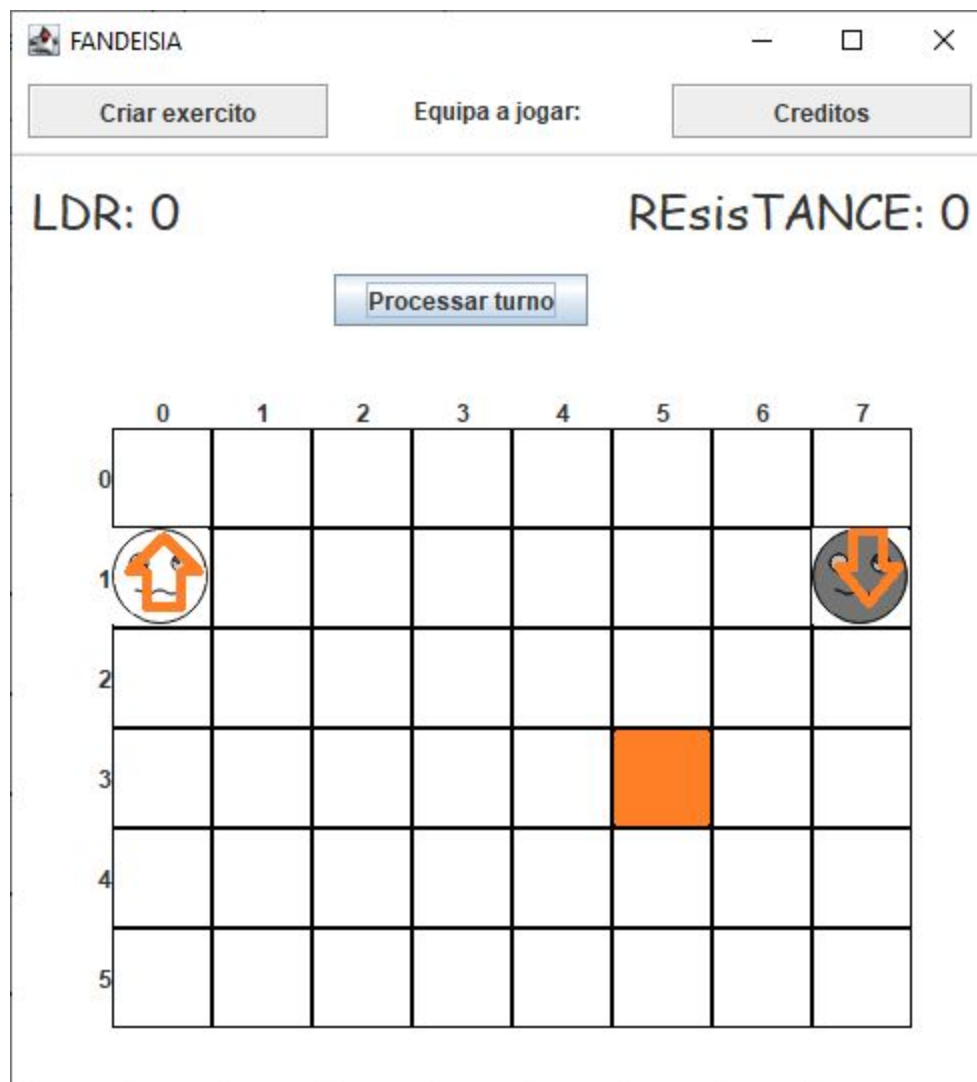
Bruno Cipriano, Rodrigo Correia, Pedro Alves

- uma criatura que esteja virada para “Este” vai virar-se para “Sul”.
 - E assim por diante.
-
- Um movimento é **impossível** se causar uma das seguintes situações:
 - colocar a criatura fora do mapa; ou
 - colocar a criatura numa posição que se encontre ocupada por outra criatura.

Visualizador Gráfico



Para suportar o trabalho neste projecto, vai ser fornecido um Visualizador Gráfico que permite aos alunos/jogadores enviarem as jogadas para a simulação. Este visualizador será fornecido através de um ficheiro .jar que será publicado no moodle.

A imagem seguinte apresenta o aspecto geral do visualizador no seu arranque. Na mesma podemos ver duas criaturas (uma com orientação Norte e outra com orientação Sul) e um tesouro (o rectângulo laranja):



Bruno Cipriano, Rodrigo Correia, Pedro Alves

A seguir ao arranque do visualizador, o jogador humano deve começar por carregar no botão “**Criar exército**”, que lhe irá permitir definir quais as criaturas que pretende ter em jogo, na sua equipa. Na imagem seguinte apresenta-se o ecrã de escolha de exército que aparece ao carregar no botão referido.

	Nome	Descrição	Custo	Quantidade
	Elfo	Pode voar por cima das outras criaturas	10	1
	Dragao	Consegue fazer cenas...	15	0

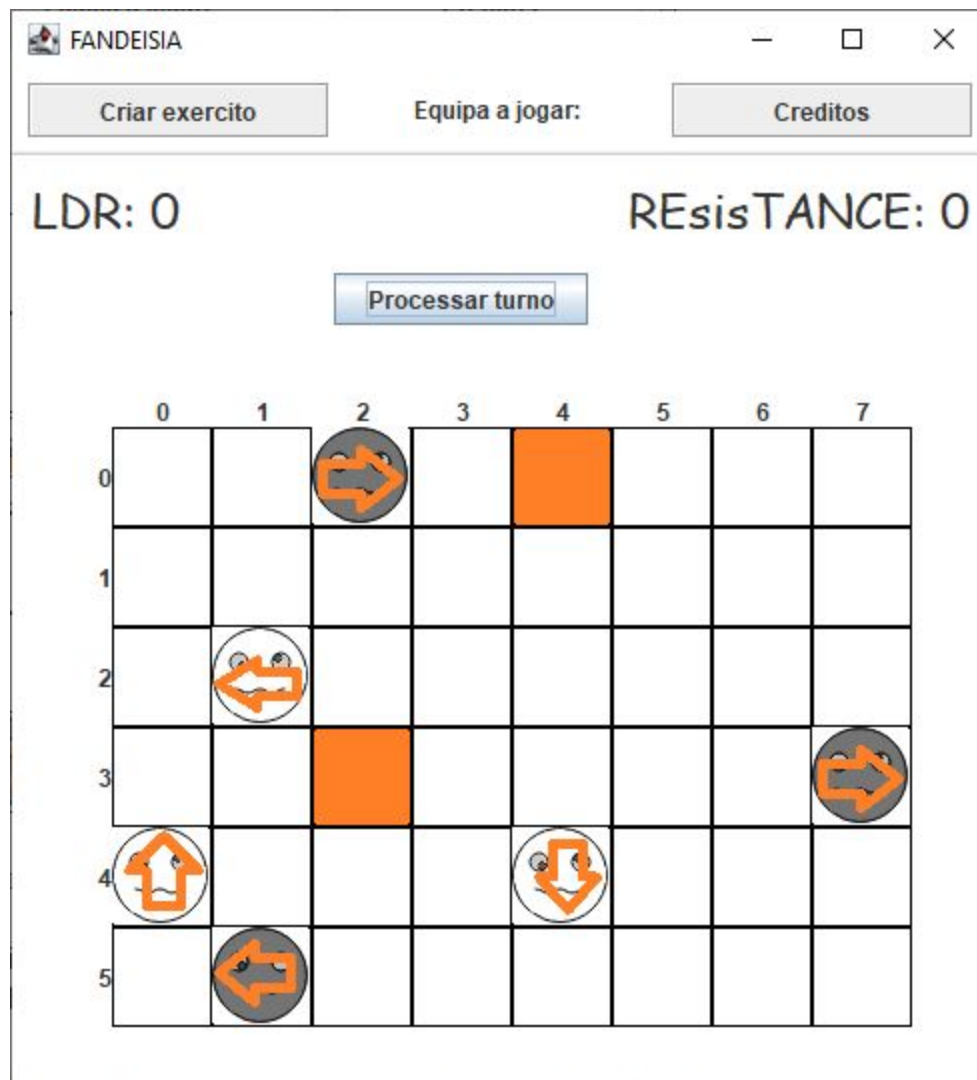
OK Cancelar

(Nota: não é da responsabilidade dos alunos o desenho deste ecrã, mas os dados que aparecem - lista de criaturas - serão dados por uma função que os alunos têm de implementar.)

Ao carregar em “**OK**” neste diálogo, o jogo estará pronto a arrancar.

Nota: nesta primeira Parte do projecto, o exército escolhido para o jogador Humano será também usado pelo jogador automático.

E agora apresenta-se uma *screenshot* do Visualizador após ter sido criado o exército e iniciado o jogo (com 6 peças e 2 tesouros):



Existe uma forma de “configurar” a imagem que o visualizador apresenta para cada peça, de forma a tornar o jogo mais amigável.

Este visualizador já trata de algumas componentes que normalmente fazemos em todos os projectos, nomeadamente a implementação da função `main(...)` que faz aparecer a janela gráfica.

Este visualizador também tem botões que permitem executar as funções do simulador:

- **Criar Exército** - pede ao jogador humano que crie o seu exército;
- **Créditos** - pede ao simulador a informação sobre os autores do projecto e apresenta a mesma numa janela gráfica;

Bruno Cipriano, Rodrigo Correia, Pedro Alves

- **Processar turno** - termina o turno do jogador humano e, de seguida, processa o turno do computador.

Para que o visualizador possa funcionar correctamente, o projecto a desenvolver pelos alunos terá de respeitar algumas **restrições técnicas**, nomeadamente em termos de nomes de classes e de *packages*.

Também existem alguns métodos de implementação obrigatória, que são os métodos que respondem aos botões e que permitem obter informação para desenhar o mundo. A própria imagem que se apresenta para cada criatura é dada por outro método obrigatório.

A lista completa de classes e métodos cuja implementação é obrigatória está no **Anexo I - Informações e Restrições sobre o Visualizador**.

O uso do visualizador fornecido é obrigatório!

Durante o semestre, poderão ser publicadas novas versões do visualizador (p.e. com correcções de bugs e/ou com as alterações necessárias para suportar as várias partes do projecto). Se isto acontecer, será publicada uma mensagem informativa em moodle.

Resultados da execução do sistema

Nesta secção apresentam-se as informações que devem ser produzidas para apresentação das estatísticas finais do jogo.

No final do jogo, o Visualizador irá pedir, ao código dos alunos, a informação descrita abaixo.

Isto será feito através de uma chamada ao método `getResults()` que terá de ser implementado pelos alunos de forma a devolver uma lista de `Strings` com o formato indicado.

Para que seja considerado correcto, o formato (incluindo espaçamento) tem de ser respeitado. Ver mais informações no [Anexo I](#).

Em caso de **EMPATE**, deve ser produzida a seguinte informação:

```
Welcome to FANDEISIA
Resultado: EMPATE
LRD: <nr de pontos da equipa LRD>
RESISTENCIA: <nr de pontos da equipa RESISTENCIA>
Nr. de Turnos jogados: <nr total de turnos jogados>
-----
<ID da criatura com menor ID> : <Tipo da criatura> : <nr de tesouros encontrados ...>
<ID da segunda criatura com o menor ID> : <Tipo da criatura> : <nr de tesouros ...>
<Etc.>
```

Em caso de **VITÓRIA** de uma das equipas, deve ser produzida a seguinte informação:

```
Welcome to FANDEISIA
Resultado: Vitória da equipa <nome da equipa>
<Nome da equipa com mais pontos>: <nr pontos desta equipa>
<Nome da outra equipa>: <nr pontos desta equipa>
Nr. de Turnos jogados: <nr total de turnos jogados>
-----
<ID da criatura com o menor ID> : <Tipo da criatura> : <nr de tesouros encontrados ...>
<ID da segunda criatura com o menor ID> : <Tipo da criatura> : <nr de tesouros ...>
```

<Etc.>

Notas:

- a informação entre <> corresponde a dados dinâmicos/calculados. O resto da informação corresponde a dados estáticos.
- o campo <Tipo da criatura> deverá ser escrito no seu formato palavra (p.e. “Elfo”).

Testes Unitários Automáticos

Nesta componente, os alunos devem implementar pelo menos **4 (quatro) casos de teste** que validem a correcção da implementação do método `void processTurn()` (que se descreve em detalhe no Anexo I).

Por “caso de teste” entende-se a definição de um método que verifique se, para certo “*input*” é obtido o “*output*” / *resultado esperado*.

Estes casos de teste devem ser implementados usando **JUnit**, tal como demonstrado nas aulas.

Os testes devem ser implementados em uma ou mais classes com o nome `TestXYZ` (em que `XYZ` é o nome da classe que está a ser testada). Por exemplo, uma classe chamada `ContaBancaria` teria os seus testes numa classe chamada `TestContaBancaria`. As classes de teste devem estar no mesmo package que as classes que estão a ser testadas.

Entrega

O que entregar

Nesta primeira parte do projecto, os alunos têm de entregar:

- Um diagrama de classes em **UML** (em formato `pdf` ou `png`), assim como eventuais comentários que os alunos decidam fazer no sentido de justificarem as suas escolhas de modelação;
- Ficheiros Java onde seja feita a implementação das diversas classes que façam parte do modelo;
- Ficheiros Java que implementem os testes unitários automáticos (JUnit).

Nota importante: O programa submetido pelos alunos tem de compilar e executar no contexto do visualizador e no contexto do Drop Project.

No visualizador:

- Carregar nos botões **não pode** resultar em erros de *runtime*. Projectos que não cumpram esta regra serão considerados como não sendo entregues. Isto significa que todos os métodos obrigatórios terão de estar implementados e a devolver valores que cumpram as restrições indicadas.

No Drop Project:

- Projectos que não passem as fases de **estrutura** e de **compilação** (e **CheckStyle**) serão considerados como não entregues.

Estrutura do projecto

O projecto deve estar organizado na seguinte estrutura de pastas:

```
AUTHORS.txt (contém linhas NUMERO_ALUNO;NOME_ALUNO, uma por aluno do grupo)
diagrama.pdf (ou.png)
+ src
|---+ pt
|-----+ ulusofona
|-----+ lp2
|-----+ fandeisiaGame
|----- FandeisiaGameManager.java
|----- Creature.java
|----- ... (outros ficheiros java do projecto)
|----- TestXXX.java
|----- ... (outras classes de testes)
```

Como entregar - Repositório git (github)

A entrega deste projecto deverá ser feita usando um **repositório privado git**. Não serão aceites outras formas de entrega do projecto.

Cada grupo deve criar um repositório privado git no *github* [github.com] e partilhar esse repositório com o Professor das aulas práticas.

Os vários alunos do grupo devem estar associados ao repositório *github* do grupo e devem usar o mesmo para trabalhar de forma colaborativa.

Todos os ficheiros a entregar (seja o relatório / UML, seja código) devem ser colocados no repositório git. O ficheiro que contiver o relatório / diagrama deve-se chamar diagrama.pdf (ou diagrama.png) e deve estar na **raiz** do projecto.

Notas:

- O user id / username de cada aluno no *github* tem de incluir o respectivo número de aluno.
- Recomenda-se que o repositório tenha os números de aluno dos vários membros do grupo (p.e. “LP2 - 2100000 2100001”).

A criação deste repositório privado e a sua partilha com o Professor é essencial para a entrega do projecto.

Os alunos terão também acesso a uma página no **Drop Project [DP]** a partir da qual poderão pedir para que o estado actual do seu repositório (ou seja, o *commit* mais recente) seja testado. Nesta primeira parte do projecto, a página do DP a usar é a seguinte:

<https://deisi.ulusofona.pt/drop-project/upload/lp2-projecto-1920-parte-1>

Regra dos Commits - Parte 1

Nesta primeira parte do projecto, deverão ser feitos (pelo menos) **dois (2) commits** por cada aluno do grupo. Os alunos que não cumpram esta regra **serão penalizados em 3 valores** na nota final desta parte do projecto.

Prazo de entrega

A entrega deverá ser feita através de um *commit* no repositório git previamente criado e partilhado com o Professor. Recomenda-se que o repositório git seja criado (e partilhado) o mais rápido possível, de forma a evitar que surjam problemas de configuração no envio.

Para efeitos de avaliação do projecto, será considerado o último *commit* feito no repositório.

A data limite para fazer o último *commit* é o dia **22 de Novembro de 2019 (Sexta-feira)**, pelas **23h55m** (hora de Lisboa, Portugal). Recomenda-se que os alunos verifiquem que o *commit* foi enviado (*pushed*), usando a interface *web* do github. Não serão considerados *commits* feitos após essa data e hora.

Avaliação

A avaliação do projecto será dividida pelas 3 partes:

- Nas três partes existirão baterias de testes automáticos implementadas no sistema **Drop Project** ([DP]) que irão avaliar o projecto do ponto de vista funcional.
- Existirá uma nota em cada parte do projecto.
- Após a entrega final, será atribuída ao projecto uma nota final quantitativa, que será calculada considerando a seguinte fórmula:
 - $0.25 * \text{NotaParte1} + 0.55 * \text{NotaParte2} + 0.20 * \text{NotaParte3}$

Segue-se uma tabela de resumo dos itens de avaliação para a **primeira parte** do projecto:

Tema	Descrição	Cotação (valores)
Modelo de Classes	Análise do Diagrama UML por parte do Professor. O diagrama deve seguir as regras UML apresentadas nas aulas. Deve também estar em conformidade com o código apresentado.	2
Testes automáticos	Os alunos definem os casos de teste automáticos indicados para esta parte do projecto.	1
Avaliação funcional (DP)	O DP irá testar o Simulador dos alunos considerando situações de abertura de ficheiro, situações de jogada (quer válidas, quer inválidas), situações de detecção da condição de paragem (vitória, empate, etc.).	12
Avaliação funcional (Professores)		5

Avaliação - outras informações

Relativamente à análise do diagrama UML e do código do projecto:

- Serão valorizadas soluções que:
 - usem os mecanismos da programação orientada por objectos (encapsulamento, *method overloading*, etc.) nas situações apropriadas.
- Serão penalizadas soluções que:
 - não façam uso de mecanismos OO
 - (p.e. todo o projecto implementado em uma única classe, etc.).
 - cujo código Java que não corresponda ao modelo apresentado em UML.
 - tenham situações de acesso directo a atributos
 - (p.e. alteração directa de atributos de uma classe por parte de métodos de outra classe)
 - implementem métodos que não modificam nem consultam qualquer atributo/variável da classe respectiva.
 - façam uso não justificado de funções ou variáveis “static”.

Cópias

Trabalhos que sejam identificados como cópias serão anulados e os alunos que os submetam terão nota zero (0).

Uma cópia numa das entregas intermédias afastará os alunos (pelo menos) da restante avaliação de primeira época, podendo inclusivamente ter efeitos nas restantes épocas.

Notem que, em caso de cópia, serão penalizados quer os alunos que copiarem, quer os alunos que deixarem copiar.

Nesse sentido, recomendamos que não partilhem o código do vosso projecto (total ou parcialmente). Se querem ajudar colegas em dificuldade, expliquem-lhes o que têm que fazer, não lhes forneçam o vosso código pois assim eles não estão a aprender!

A decisão sobre se um trabalho é uma cópia cabe exclusivamente aos docentes da unidade curricular.

Outras informações relevantes

– Não deve ser implementado qualquer menu (ou interface gráfica) para interação manual com o utilizador. A única interface gráfica prevista é o Visualizador Gráfico disponibilizado pelos docentes.

– Recomenda-se que eventuais dúvidas sobre o enunciado sejam esclarecidas (o mais depressa possível) em aula e/ou no fórum do Moodle de Linguagens de Programação II 2019/2020.

– Os projetos devem ser realizados em grupos de 2 alunos. Em situações excepcionais poderão ser aceites grupos de um aluno - quem o pretender fazer deve efectuar um pedido (justificado) aos docentes da cadeira através de e-mail.

– Os grupos de alunos definidos para a primeira parte do projecto devem ser mantidos para a segunda e terceira partes do projecto. Não serão permitidas entradas de novos alunos nos grupos entre as duas partes do projecto.

Universidade Lusófona de Humanidades e Tecnologias
Linguagens de Programação II
LEI / LIG / LEIRT
2019/20 – 1º Semestre
Primeira Parte
v1.0.0

Bruno Cipriano, Rodrigo Correia, Pedro Alves

- Existirá uma defesa presencial e individual do projeto. A defesa será feita após a terceira entrega. Durante esta defesa individual, será pedido ao aluno que faça alterações ao código do projecto, de forma a dar resposta a alterações aos requisitos. Se o aluno não conseguir realizar pelo menos metade das alterações pedidas, terá nota zero (0) no projecto.
- É possível que sejam feitas pequenas alterações a este enunciado, durante o tempo de desenvolvimento do projeto. Por esta razão, os alunos devem estar atentos ao **Moodle de LP II**.
- Qualquer situação omissa será resolvida pelos docentes da cadeira.

Anexo I - Instruções e Restrições sobre o Visualizador

Como foi referido, o projecto irá correr em cima de um visualizador gráfico, distribuído em forma de `.jar`.

Para que o visualizador funcione correctamente, é necessário respeitar as seguintes restrições:

1) É obrigatório criar as classes seguintes:

- `FandeisiaGameManager` - que será responsável por gerir o jogo;
- `Creature` - que irá representar a criatura, ou seja, terá de conter os atributos relevantes para a caracterização das criaturas que participam no jogo e terá de ter as funções/métodos que implementem os comportamentos relevantes das criaturas.

2) Essas duas classes tem de ser colocadas num package chamado:

```
package pt.ulusofona.lp2.fandeisiaGame
```

3) A classe `FandeisiaGameManager` tem de conter (pelo menos) o construtor sem argumentos.

4) A classe `FandeisiaGameManager` tem de conter (pelo menos) os métodos seguintes:

Assinatura	Comportamento
<pre>public String[][] getCreatureTypes</pre>	<p>Deve retornar os tipos de criatura que existem no jogo e que podem ser escolhidos para os exércitos dos dois jogadores.</p> <p>O retorno deve ser um array 2d de <code>String</code>, em que cada “linha” tem a seguinte informação:</p> <ul style="list-style-type: none">• Nome do tipo (posição 0)• Imagem PNG (posição 1)• Descrição textual (posição 2)

Universidade Lusófona de Humanidades e Tecnologias
Linguagens de Programação II
LEI / LIG / LEIRT
2019/20 – 1º Semestre
Primeira Parte
v1.0.0

Bruno Cipriano, Rodrigo Correia, Pedro Alves

	<ul style="list-style-type: none"> • Custo (posição 3) <p>(O valor do “Custo” não será usado nesta P1).</p>
<pre>public void startGame(String[] content, int rows, int columns)</pre>	<p>Deve inicializar as estruturas de dados relevantes para processar um jogo.</p> <p>O array <code>content</code> irá descrever o conteúdo inicial do mundo (criaturas e tesouros), tendo para isso várias Strings. Cada String vai representar um objecto do mundo. As Strings vão ter um dos seguintes formatos:</p> <p>Para criaturas: <code>"id: <id>, type: <type>, teamId: <teamId>, x: <x>, y: <y>, orientation: <orientation>"</code></p> <p>Para tesouros: <code>"id: <id>, type: treasure, x: <x>, y: <y>"</code></p> <p>Os argumentos <code>rows</code> e <code>columns</code> vão-nos indicar as dimensões do tabuleiro.</p>
<pre>public void setInitialTeam(int teamId)</pre>	<p>Indica qual das equipas vai jogar no primeiro turno do jogo.</p>
<pre>public void processTurn()</pre>	<p>Deve processar um turno do jogo considerando a equipa actual.</p> <p>Inclui o movimento das criaturas.</p>
<pre>public List<Creature> getCreatures()</pre>	<p>Devolve uma lista com todos os objectos <code>Creature</code> que existem no jogo.</p>
<pre>public boolean gameIsOver()</pre>	<p>Deve devolver <code>true</code> caso já tenha sido alcançada uma das condições de paragem do jogo e <code>false</code> em caso contrário.</p>

Universidade Lusófona de Humanidades e Tecnologias
Linguagens de Programação II
LEI / LIG / LEIRT
2019/20 – 1º Semestre
Primeira Parte
v1.0.0

Bruno Cipriano, Rodrigo Correia, Pedro Alves

<code>public List<String> getAuthors()</code>	<p>Devolve uma lista de <code>Strings</code> com os nomes dos autores do projecto.</p> <p>Esta informação será usada para mostrar o conteúdo da janela que aparece ao carregar no botão de “Créditos”.</p>
<code>public List<String> getResults()</code>	<p>Devolve uma lista de <code>Strings</code> que representem os resultados do jogo, conforme descrito na secção dos “Resultados da execução ...”.</p> <p>Este método não pode devolver <code>null</code>. Caso não calculem a informação respectiva, devem devolver uma lista vazia.</p>
<code>public int getElementId(int x, int y)</code>	<p>Deve devolver o ID do objecto/elemento que se encontra na posição indicada pelas coordenadas (x,y) passadas por argumento.</p> <p>Por objecto/elemento entende-se: criatura ou tesouro.</p> <p>Caso não exista nenhuma criatura ou tesouro na posição indicada, o método deve devolver o valor 0 (zero) que representa o vazio.</p>
<code>public int getCurrentTeamId()</code>	Deve devolver o ID da equipa que está activa no turno actual.
<code>public int getCurrentScore(int teamID)</code>	Deve devolver o número actual de pontos da equipa que tem o ID <code>teamID</code> .

5) A classe **Creature** tem de conter (pelo menos) os três (3) métodos seguintes:

Assinatura	Comportamento
<code>public int getId()</code>	Deve devolver o ID da criatura.
<code>public String getImagePNG()</code>	<p>Deve devolver o nome do ficheiro de imagem (formato PNG) que representa a criatura.</p> <p>(As imagens a usar devem ser colocadas na pasta <code>src/images</code> e devem ter tamanho 50x50. As imagens devem ter fundo transparente para que se consiga ver se estão num quadrado branco ou preto).</p> <p>Caso os alunos não pretendam definir nenhuma imagem, a função pode simplesmente retornar <code>null</code>. Isto fará com que o visualizador use uma imagem pré-definida por omissão.</p>
<code>public String toString()</code>	<p>Retorna uma <code>String</code> com a informação sobre a criatura.</p> <p>Sintaxe:</p> <p><code>"<ID> <Tipo> <ID Equipa> <Nr Pontos> @ (<x>, <y>) <Orientação>"</code></p> <p>Onde <code><Nr Pontos></code> corresponde a quantidade de Tesouros encontrados pela criatura.</p>

Nota importante:

Para além dos métodos que são indicados como obrigatórios, as classes `Creature` e `FandeisiaGameManager` poderão precisar de informação extra. Por exemplo, como é que vão controlar qual é a “**equipa actual**”? Faz sentido colocar guardar essa informação em alguma destas classes?

FIM