O Grande Jogo do DEISI, Primeira Parte V1.0.0

Bruno Cipriano, Lúcio Studer, Rodrigo Correia, Pedro Alves

O Grande Jogo do DEISI

Enunciado do projecto prático - Primeira Parte

Objectivo

Este projecto tem como objectivo o desenvolvimento de uma aplicação (programa) usando as **linguagens Java** (versão Java 16) e **Kotlin** (versão 1.5) 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. Alguns dos novos requisitos terão de ser implementados 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.

O Grande Jogo do DEISI, Primeira Parte V1.0.0

Bruno Cipriano, Lúcio Studer, Rodrigo Correia, Pedro Alves

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 16**.
- Implementar algumas funcionalidades, também em **Java 16**.
- Criar testes automáticos unitários para uma parte do modelo implementado, usando **JUnit 4**.

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

Pretende-se desenvolver um jogo de tabuleiro (virtual) no qual **programadores** participam numa corrida, tentando esquivar-se de erros e problemas causados ou potenciados por escolhas duvidosas durante o processo de desenvolvimento de software. Por outro lado, podem ter a sorte de encontrar boas práticas de programação que o ajudarão a progredir mais rapidamente.

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.

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.

Os programadores

Todos os programadores são caracterizados por:

- O seu nome;
- Os nomes das suas linguagens de programação favoritas;

Para além disso, para efeitos da participação nos jogos, são também relevantes as seguintes informações:

- O ID - um número inteiro positivo que identifica o programador no simulador;
- A cor do avatar que representa o programador em jogo.

Nota: Os IDs dos programadores podem não ser consecutivos. Por exemplo, o primeiro programador pode ter o ID=1, o segundo programador ter o ID=3, o terceiro ter o ID=42, e assim por diante.

Tabuleiro do Jogo

O jogo decorre num tabuleiro dividido em N casas. Cada casa está numerada de 1 até N. O tamanho do tabuleiro(N) poderá variar de jogo para jogo.

No seguinte exemplo mostra-se um tabuleiro de tamanho N = 10:

1	2	3	4	5	6	7	8	9	10

A casa da posição 1 é a “casa partida”.
Num tabuleiro de tamanho N, a casa da posição N é a “meta”.

Nesta primeira fase do projecto, cada casa do tabuleiro poderá:

- Estar vazia;
- Estar ocupada por um ou mais programadores.

Regras do Jogo

- O jogo terá entre 2 e 4 jogadores.

- Todos os jogadores começam o jogo posicionados na “Casa partida”.

- O jogo está dividido em turnos.

- Em cada turno, joga um dos programadores.

- As jogadas são feitas por ordem do ID:
    - o jogador que tiver o ID mais pequeno joga primeiro.

- Jogar implica lançar um dado de 6 lados, para determinar qual o número de casas que o programador irá avançar neste turno.

- Após lançar o dado, o programador vai-se mover até à casa A + M, onde A representa o número da sua casa actual e M representa o número que saiu no dado.

- Ao chegar à casa A + M, o movimento do jogador termina e o turno passa para o jogador seguinte.

- Caso um programador ultrapasse a casa final do jogo, então deve recuar o número de casas em excesso.

- Exemplo:
    - O tabuleiro tem 100 casas;
    - O programador do turno actual está na casa 99;
    - O dado é lançado e sai o número 3;
    - Nesta situação, há 2 casas em excesso (100 - 99 - 3 = - 2);
    - O programador recua então para a posição 100 - 2 = 98.

- Nota: será garantido pelos testes do DP que o recuo nunca será tal que o programador vá para uma posição não existente do tabuleiro.

- O jogo termina quando for atingida a seguinte condição:

- Um dos programadores chegar à casa final do jogo.

O Grande Jogo do DEISI, Primeira Parte V1.0.0

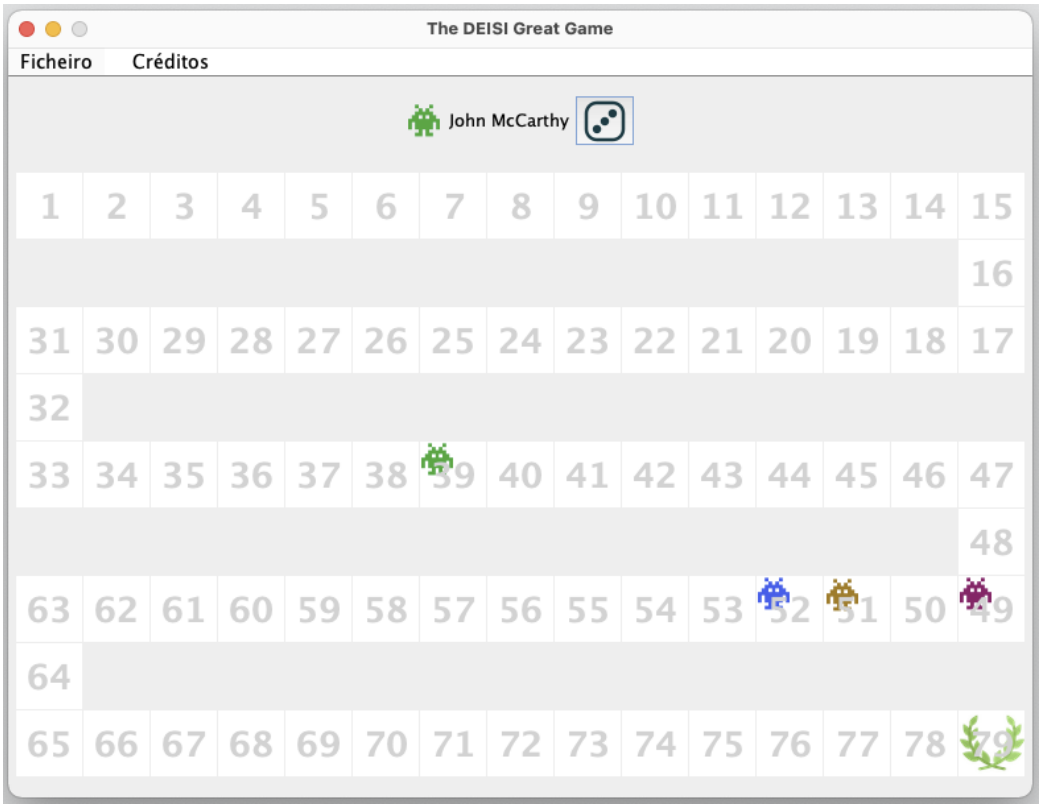
Bruno Cipriano, Lúcio Studer, Rodrigo Correia, Pedro Alves

Visualizador Gráfico

Para suportar o trabalho neste projecto, vai ser fornecido um Visualizador Gráfico que permite aos alunos/jogadores interagirem com o motor do jogo. Este visualizador será fornecido através de um ficheiro .jar que será publicado no moodle.

A configuração do IDE para utilizar o visualizador está explicada num documento disponibilizado no Moodle intitulado “LP2 2021/2022 - Como configurar o IDE para utilizar uma Biblioteca Externa em formato jar”. Esse documento está na mesma pasta onde encontraram este enunciado.

A imagem seguinte apresenta o aspecto geral do visualizador após seleccionar o número de jogadores (neste caso foram escolhidos 4 jogadores).



No topo do tabuleiro é apresentado o programador que vai jogar agora. Ao clicar no botão com o dado, é gerado um número aleatório entre 1 e 6 e esse jogador movimentar-se-á esse número de casas, conforme as regras definidas anteriormente.

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:

- **Créditos** - pede ao simulador a informação sobre os autores do projecto e apresenta a mesma numa janela gráfica;
- Existe também uma opção "**Reiniciar**" no menu "Ficheiro" - esta opção permite recomeçar o jogo actual.

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 tabuleiro. 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.

Dados de entrada da aplicação (*Input*)

Nesta primeira parte do projecto, os dados de entrada do programa são os seguintes:

- Tamanho do tabuleiro;
- IDs, Nomes, Linguagens Favoritas dos jogadores/programadores que vão participar no jogo;
- A cor do boneco que representa cada jogador.

Para mais informações, consultar no **Anexo I** a descrição da função `createInitialBoard(...)`

Resultados da execução do sistema

No final do jogo, o Visualizador irá pedir ao código dos alunos a informação descrita abaixo. Isso será feito através de uma chamada ao método `getGameResults()` 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.

```
O GRANDE JOGO DO DEISI

NR. DE TURNOS
<NR DE TURNOS>

VENCEDOR
<NOME DO PROGRAMADOR VENCEDOR>

RESTANTES
<NOME DO SEGUNDO MELHOR CLASSIFICADO> <POSIÇãO>
<NOME DO TERCEIRO MELHOR CLASSIFICADO> <POSIçÃO>
<...>
```

A lista dos “RESTANTES” deve apresentar todos os jogadores excepto o vencedor. Os nomes destes jogadores devem aparecer pela ordem da sua proximidade ao final do jogo. O jogador que estiver mais perto da meta deve aparecer em primeiro lugar. Pode assumir que, no final do jogo, cada posição terá no máximo 1 programador.

Notas:

- a informação entre <> corresponde a dados dinâmicos/calculados. O resto da informação corresponde a dados estáticos.

- cada linha de texto do formato indicado deve ser um elemento da lista devolvida.

Testes Unitários Automáticos

Nesta componente, os alunos devem implementar pelo menos **4 (quatro) casos de teste** que validem a correcção da implementação do método `boolean moveCurrentPlayer(...)` (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 4**, tal como demonstrado nas aulas. Cada caso de teste deve ser implementado num método anotado com `@Test`.

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.

Nota: para que os testes sejam considerado válidos, as condições seguintes têm de se verificar:

- A função de teste não pode estar vazia;
- A função de teste tem de ter pelo menos um assert;
- A função de teste tem de chamar código do projecto;
- O assert tem de comparar o resultado de uma função do projecto com um resultado esperado.

Entrega

O que entregar

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

- Um diagrama de classes em **UML** (em formato `png`);
- Um ficheiro README.MD (ver detalhes na próxima sub-secção);
- 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** serão considerados como não entregues.
- Projectos que não passem a fase de **CheckStyle** serão penalizados em 3 valores.

Ficheiro README.md

O ficheiro README.md do repositório github deve contar os seguintes artefactos:

- Apresentação da imagem do diagrama UML do projecto;
- Eventuais comentários que os alunos decidam fazer no sentido de justificarem as suas escolhas de modelação.

Para incluirem a imagem do vosso diagrama no README.md devem usar o código seguinte:

```
![](diagrama.png?raw=true "Diagrama UML")
```

Nota: Projectos que não tenham o ficheiro README.md ou cujo ficheiro README.md não inclua o diagrama UML terão uma penalização de 3 valores na nota final.

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
|------------+ deisiGreatGame
|--------------- GameManager.java
|--------------- Programmer.java
|--------------- ...   (outros ficheiros java do projecto)
|--------------- TestXXX.java
|--------------- ... (outras classes de testes)
```

O Grande Jogo do DEISI, Primeira Parte V1.0.0

Bruno Cipriano, Lúcio Studer, Rodrigo Correia, Pedro Alves

Como entregar - Repósitorio 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 UML, seja código) devem ser colocados no repositório git. O ficheiro que contiver o diagrama deve-se chamar **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/lp2-2122-projecto-1a-epoca-p1

Regra dos Commits - Parte 1

Nesta primeira parte do projecto, deverão ser feitos (pelo menos) **dois (2) *commits* não triviais** (que tenham impacto na funcionalidade do programa), 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 **15 de Novembro de 2021 (Segunda-feira)**, pelas **09h00m AM** (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.
- Nesta primeira parte aplica-se a nota mínima de **oito (8)** valores
  - Quem não alcançar essa nota mínima ficará excluído da avaliação prática de primeira época.
- Após a entrega final, será atribuída ao projecto uma nota final quantitativa, que será calculada considerando a seguinte fórmula:
  - `0.20 * NotaParte1 + 0.55 * NotaParte2 + 0.25 * NotaParte3`
  - Na nota final existe a nota mínima de 9.5 valores.

O Grande Jogo do DEISI, Primeira Parte V1.0.0

Bruno Cipriano, Lúcio Studer, Rodrigo Correia, Pedro Alves

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, 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) no **discord** de Linguagens de Programação II 2021/2022:

- Brevemente serão dadas mais informações;

– 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.

– 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, irá reprovar 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:
    - `GameManager` - responsável por gerir o jogo;
    - `Programmer` - representa os programadores que estão em jogo;
    - `Main` - apenas para ser aceite pelo Drop Project, pode estar vazia.
    - Nota: podem ser criadas mais classes além destas, se acharem que são úteis para a vossa implementação

2) É obrigatório criar um tipo enumerado (i.e. `enum`) chamado `ProgrammerColor`, que deve ter os seguintes valores possíveis:
    - `PURPLE`
    - `BLUE`
    - `GREEN`
    - `BROWN`

3) Todas as classes e o `Enum` têm de ser colocadas num package chamado:

```
package pt.ulusofona.lp2.deisiGreatGame
```

4) A classe `GameManager` tem de conter (pelo menos) o construtor sem argumentos.
    - Este construtor tem de ter a visibilidade `public`.

5) A classe `GameManager` tem de conter (pelo menos) os métodos seguintes:

Assinatura	**Comportamento**
`public boolean createInitialBoard(String[][] playerInfo, int boardSize)`	Cada elemento do array **playerInfo** vai ter a informação de um programador: - [0] => O ID do Jogador - [1] => O Nome do Jogador - [2] => A sua lista de linguagens (as linguagens serão separadas por ";") - [3] => A cor do boneco que identifica o jogador. Os valores possíveis são: "Purple", "Green", "Brown" e "Blue". O **`int boardSize`** vai indicar o tamanho do tabuleiro (N). A função deve validar os dados recebidos. Caso algum dos dados seja inválido, a função deverá retornar false. Dados a validar: ● Os IDs dos programadores. Não podem haver dois jogadores com o mesmo ID. Para além disso, o ID tem de ser um valor que pertença à gama esperada. ● Os nomes dos programadores. Não podem ser null nem estar vazios. ● A cor de cada jogador deve ser uma das quatro possíveis. ● Não podem haver 2 jogadores com a mesma cor. ● O número de jogadores. ● O tabuleiro tem de ter, pelo menos, duas posições por cada jogador que esteja em jogo.

Assinatura	**Comportamento**
`public String getImagePng(int position)`	Deve devolver o nome do ficheiro de imagem (formato PNG) que representa no tabuleiro a posição cujo número é dado pelo argumento `position`. (As imagens a usar devem ser colocadas na pasta src/images e devem ter tamanho 50x50). De forma a tornar o jogo mais amigável para o utilizador, recomendamos que tenham uma imagem pelo menos para a última posição (meta). Podem usar a imagem "glory.png" incluída no projeto ou usar a vossa própria imagem. Caso o `position` seja inválido (p.e. Maior do que o tamanho do tabuleiro), a função deve retornar null.
`public ArrayList<Programmer> getProgrammers()`	Devolve uma lista com todos os objectos `Programmer` que existem em jogo.
`public ArrayList<Programmer> getProgrammers(int position)`	Devolve uma lista com os objectos `Programmer` que se encontrem numa determinada posição do tabuleiro. Caso o `position` seja inválido ou caso não existam programadores na posição indicada, a função deve devolver null.
`public int getCurrentPlayerID()`	Devolve o ID do programador que se encontra activo no turno actual.

Assinatura	**Comportamento**
`public boolean moveCurrentPlayer(int nrPositions)`	Move o programador do turno actual tantas casas quantas as indicadas no argumento `nrPositions`. A jogada tem de ser validada, considerando as seguintes regras: ● O argumento `nrPositions` não pode ser menor do que 1 ou maior do que 6, porque o dado tem 6 lados. Caso alguma destas regras não seja cumprida, então a função deve devolver false e o turno continua a ser do jogador actual. Em caso contrário, a função deve devolver true.
`public boolean gameIsOver()`	Deve devolver `true` caso já tenha sido alcançada uma das condições de paragem do jogo e `false` em caso contrário.
`public ArrayList<String> getGameResults()`	Devolve uma lista de `Strings` que representam os resultados do jogo, conforme descrito na secção dos “Resultados da execução …”. Este método não pode devolver `null`. Caso não calculem a informação respectiva, devem devolver **uma lista vazia**.
`public JPanel getAuthorsPanel()`	Devolve um JPanel que pode ter o conteúdo gráfico que os alunos queiram que seja apresentado ao carregar na janela de “Créditos”. O JPanel devolvido pelos utilizadores será apresentado numa janela 300x300. Caso os alunos não pretendam implementar o JPanel, devem devolver `null`.

6) A classe **`Programmer`** tem de conter (pelo menos) os três (3) métodos seguintes:

Assinatura	**Comportamento**
`int getId()`	Deve devolver o ID do programador.
`String getName()`	Deve devolver o nome do programador.
`ProgrammerColor getColor()`	Deve devolver a cor do boneco que representa o programador no tabuleiro.
`String toString()`	Retorna uma `String` com a informação sobre o programador. Esta `String` será apresentada pelo visualizador como tooltip do objecto gráfico que representa cada jogador. **Sintaxe**: `"<ID> | <Nome> | <Pos> | <Linguagens favoritas> | <Estado>"` Onde: ● <Pos> é um número inteiro que identifica a posição actual do programador no tabuleiro. ● <Estado> deve ter o valor "Em Jogo" (caso o jogador ainda esteja em jogo) ou "Derrotado" (caso o jogador tenha saído do jogo). ● <Linguagens favoritas> é uma String com os nomes das linguagens favoritas, separadas por "`;` " (ponto e vírgula seguido de espaço). A lista deve ser ordenada alfabeticamente pelo nome da linguagem. **Nota:** Para programadores que saiam do jogo, <Pos> deve ter a posição onde estavam quando perderam o jogo. **Nota:** Nesta primeira parte do projecto, os programadores nunca saem de Jogo.

Nota importante:

Para além dos métodos que são indicados como obrigatórios, as classes `Programmer` e `GameManager` poderão precisar de informação extra. Por exemplo, como é que vão controlar qual é o "jogador/programador actual"? Faz sentido guardar essa informação em alguma destas classes?

FIM