Licenciatura em Engenharia Informática

Introdução à Programação

Trabalho prático 2010 / 2011

(versão 1.1, 15 Dezembro 2010 – **actualizações a vermelho, com fundo amarelo**)

Se ingressou este ano na LEI, este enunciado descreve, provavelmente, o primeiro projecto de produção de software que vai ter que realizar durante a sua estadia na FCT. Se é repetente, cá está de novo a oportunidade esperada! É uma oportunidade de ouro para poder finalmente tirar várias dúvidas e de aprender com a prática, resolvendo as várias dificuldades e os pequenos (e não só) problemas que vão naturalmente surgir.

A programação é uma actividade que só se aprende a fazer e a apreciar na prática, fazendo e experimentando.

Aproveite o tempo até ao fim do prazo de entrega para tirar todas as dúvidas com os docentes e monitores da disciplina. Converse com os colegas e discuta soluções, sem esquecer as limitações impostas pelo Código de Ética.

O projecto visa o desenvolvimento de um jogo de estratégia para dois jogadores, inspirado no filme *Mars Attacks!*, do realizador Tim Burton. No filme, o planeta Terra é invadido por marcianos com armas irresistíveis e um sentido de humor cruel. No jogo que vamos implementar simularemos batalhas épicas entre os marcianos e os humanos. Verifica-se, e perdoe-nos o *spoiler* se ainda não viu o filme (é de 1996, se cerca de 14 anos não foram suficientes para o ver, por favor não nos leve a mal ☺), que a única "arma" de defesa eficaz dos humanos contra os marcianos é tocar um disco de *yodel*. O barulho faz com que as cabeças dos marcianos expludam. Resumindo, o jogo vai ter dois exércitos adversários: o exército de marcianos, equipado com armas que carbonizam os infelizes em quem os marcianos acertam; e um coro de *yodellers*, que cantarão até à morte (deles, ou dos desgraçados marcianos que se cheguem suficientemente perto para os ouvir).

Neste problema, vai ter que programar um jogo de computador. Não se assuste, para além de não o obrigarmos a ouvir os temíveis *yodellers*, vai ver que o projecto é divertido e não muito difícil.

Vamos agora ver com mais detalhe o que se pretende. Leia bem e esclareça os requisitos do sistema, antes de começar a pensar sequer em programar.

Este documento tem quatro partes:

1. SUMÁRIO, onde se explica o funcionamento geral do sistema pretendido;

2. COMANDOS, onde se explicam os vários comandos que o sistema deve ser capaz de processar;

3. EXEMPLOS, onde se ilustra, com pequenos exemplos, a execução do programa, ficheiros gerados, etc;

3. MÉTODO DE DESENVOLVIMENTO DO PROJECTO, onde se dão algumas sugestões sobre como realizar o trabalho.

1. SUMÁRIO

O sistema a implementar deve possibilitar a realização de batalhas interactivas, em que dois jogadores poderão, à vez, ir fazendo as suas jogadas e observando os resultados das mesmas. Além disso, e porque estas batalhas podem ser emocionantes, o sistema deve permitir gravar uma batalha, bem como, posteriormente, carregar uma batalha antiga e fazer *playback* da mesma até ao ponto da gravação, para que, eventualmente, os jogadores a possam depois continuar a jogar. Para suportar a reprodução e gravação de batalhas, o sistema possibilitará, respectivamente, o carregamento de *scripts* com a sequência de jogadas de um jogo antigo, e a gravação das jogadas realizadas durante um jogo em *scripts*. Estes *scripts* serão definidos em ficheiros de texto, num formato especial que descrevemos adiante. O carregamento de *scripts* permite continuar jogos que por qualquer razão não terminaram.

Um *script* é constituído por uma sequência de comandos dados pelos jogadores. Cada comando corresponde a uma ordem dada a um soldado de um dos dois exércitos, no campo de batalha. Estas ordens permitem, essencialmente, deslocar soldados e fazê-los atacar os adversários. Antes do início da batalha, apenas é possível dar ordens de criação de soldados. Na fase de construção de exércitos, a ordem de criação dos soldados é irrelevante, ou seja, os jogadores podem ir acrescentando soldados ao seu exército pela ordem que bem entenderem. A partir do início da batalha,que começa sempre com uma jogada dos marcianos (afinal, são eles os invasores), as ordens são dadas alternadamente a cada um dos exércitos, um bocado como numa partida de xadrez. Quando a batalha começa, deixa de ser possível criar novos soldados. A sua aplicação deverá ser capaz de executar um *script* (ou seja, de executar as ordens de uma batalha até ao fim do *script*), permitindo aos utilizadores que continuem a jogar a partir do ponto em que gravaram a sua batalha. A batalha termina quando todos os soldados de um dos exércitos estiverem mortos, ou quando um dos jogadores se render.

1.1 Os soldados do jogo *Mars Attacks!*

Antes de mais, convém perceber o espaço em que os soldados se deslocam. O campo de batalha pode ser visto como um espaço cartesiano bi-dimensional, em que cada soldado ocupa uma posição representada por um par de coordenadas (representadas por números inteiros), tendo também uma orientação (pode estar virado para *Norte*, *Sul*, *Este* ou *Oeste*). No campo de batalha, existem dois exércitos compostos por um número variável de soldados (*yodellers* e marcianos). Cada soldado terá a sua posição, expressa em coordenadas cartesianas. Para simplificar, assuma que pode haver mais do que um soldado na mesma posição. Consoante a sua experiência, os soldados podem ser mais ou menos poderosos. A experiência de um soldado aumenta, durante a batalha, de cada vez que ele abate um adversário. Além de pertencerem a exércitos diferentes, os soldados humanos e marcianos diferem no tipo de armamento. Os humanos vão armados para o campo de batalha com garganta, e muita, para cantar os seus *yodels* o mais alto que conseguirem. Os marcianos usam uma espécie de espingarda de raios.

As secções que se seguem descrevem os soldados de cada exército.

1.1.1 Yodellers

A informação que caracteriza um soldado humano (***yodeller***) é a seguinte:

Nome

Coordenadas actuais de localização

Orientação

Capacidade de projecção da sua voz

Número de marcianos já abatidos

Informação sobre se está vivo, ou morto

1.1.2 Marcianos

A informação que caracteriza um ***marciano*** é a seguinte:

Nome

Coordenadas actuais de localização

Orientação

Distância máxima a que consegue atingir um alvo

Número de yodellers abatidos

Informação sobre se está vivo, ou morto

1.1.3 Mais alguns detalhes sobre os soldados

Em ambos os casos, o *nome* é dado por uma string (e não deve haver nomes repetidos, entre humanos e marcianos). A localização é dada por um par de coordenadas cartesianas, de tipo inteiro, e a orientação representa o ponto cardeal para onde o soldado está virado. Os soldados das duas espécies diferem na forma como fazem os seus ataques.

- Quando um *yodeller* canta, mata todos os marcianos que estejam a uma distância igual ou inferior à sua *capacidade de projecção de voz*. Assuma que essa capacidade é medida em metros e que a voz é projectada preenchendo completamente um círculo centrado no *yodeller* e com um raio igual à tal *capacidade de projecção de voz*.
- Quando um marciano dispara, consegue matar apenas um soldado humano com cada tiro. Para acertar o tiro, tem de estar virado na direcção do humano e a uma distância igual ou ~~alcance~~ inferior à *distância máxima a que consegue acertar num alvo*.

Por cada adversário abatido, um soldado ganha pontos de experiência: 5% de incremento na sua *capacidade de projecção de voz*, no caso dos yodellers, ou 5% de incremento na *distância máxima a que consegue acertar num alvo*, no caso dos marcianos.

Infelizmente, durante uma batalha os soldados de um lado e de outro vão sendo abatidos. Rezam as crónicas que *yodellers* abatidos não cantam, não se mexem e, nem tudo é mau, também não podem ser abatidos de novo. Curiosamente, observa-se um fenómeno parecido com os marcianos abatidos. Parecem perder a cabeça, e ficam com um ar mortiço. Já não dão tiros, não se mexem, enfim, deixam de dar luta. De um e de outro lado, as coincidências parecem não ter fim: os mortos não recebem ordens nem do mais severo general!

1.2 Modo de jogo

O jogo consiste em 3 fases distintas. Na primeira fase, ambos os jogadores podem configurar o seu exército, criando para tal os soldados e colocando-os nas posições iniciais de jogo. A ordem pela qual os exércitos são criados não é importante. Nada obriga a que os exércitos tenham uma configuração pré-estabelecida. Por outras palavras, as regras permitem fazer exércitos com tantos ou tão poucos soldados quantos os jogadores

desejarem, com a capacidade e localização inicial que se quiser, desde que se respeitem as restrições referidas até aqui. Durante esta fase é possível fazer consultas ao estado do jogo, mas não dar ordens aos soldados.

A segunda fase é a fase do jogo, propriamente dito. A partir do momento em que o jogo se inicia, deixa de ser possível criar novos soldados para qualquer dos exércitos, mas passa a ser possível dar ordens aos soldados existentes. O jogo é feito em jogadas alternadas, começando pelos marcianos. Em cada jogada, o jogador pode dar exactamente uma ordem a um dos seus soldados. Se a ordem for inválida, o jogador não perde a vez. Ou seja, apenas passa a vez ao adversário quando fizer uma jogada válida. Os comandos são sempre dados a um soldado em particular do exército que tem a vez. Para serem válidos, o soldado tem de existir, fazer parte do exército, e estar vivo. No final de cada comando (em particular dos comandos de ataque) é necessário ver se a batalha terminou. A batalha termina quando um dos exércitos deixa de ter soldados vivos, ou quando algum dos jogadores se rende. Durante esta fase, podem ser feitas consultas ao estado da batalha.

Finalmente, a terceira e última fase do jogo é a que ocorre quando uma batalha termina. A partir do momento em que a batalha terminou, já não se pode dar ordens aos soldados. Ainda assim, podem-se fazer consultas ao estado do jogo, para saber, por exemplo, quais os soldados que sobreviveram à batalha, ou quais os soldados que mais adversários abateram durante a batalha.

Em qualquer momento do jogo é possível guardar a progressão até então num ficheiro de dados, ou carregar de um ficheiro, a progressão gravada numa sessão anterior.

1.3 Materiais a entregar

Todos os **ficheiros .java, num único arquivo ZIP**, e a **listagem completa desses ficheiros em papel**.

O arquivo ZIP deve ser entregue através do moodle. O arquivo deverá ter um nome com o seguinte formato: *TurnoPrático_NAluno1_NAluno2*. Por exemplo dois alunos do turno prático *P4*, com os números *12345* e *13456* deverão entregar no moodle um zip com o nome *P4_12345_13456*.

A listagem em papel, devidamente comentada, será entregue na secretaria do DI, bem identificada. Essa indicação deve incluir, além dos **números e nomes dos alunos autores**, o **turno teórico-prático** a que pertencem e o **nome do docente das aulas teórico-práticas**.

O trabalho está dimensionado para ser realizado por grupos de 2 alunos, não sendo admitidos grupos com mais que 2 alunos. Serão realizadas discussões, com grupos seleccionados pelo respectivo docente, na semana prevista para o efeito (última semana de aulas do semestre). Frequentemente, os docentes seleccionam TODOS os grupos para fazer discussão.

1.4 Prazo de entrega

O trabalho deverá ser entregue **quer em papel (na secretaria do DI), quer via moodle, até às 11h30 do dia 10 de Janeiro de 2011 (segunda-feira)**.

2. COMANDOS

Nesta secção indicam-se e explicam-se os vários comandos que o sistema deve ser capaz de interpretar e executar. Nos exemplos apresentados, o texto a **`negrito`** representa o feedback escrito pelo sistema, na consola, ao executar o comando com sucesso.

2.1 Comando *NovoJogo*

Este comando serve para começar um novo jogo, perdendo-se os dados de algum jogo que esteja a decorrer, quando o comando é invocado. O comando é inserido no sistema da forma seguinte, não necessitando de *informação adicional*:

```
NJ
```

Exemplo (criação de um novo jogo):

```
NJ
Jogo iniciado.
```

Quando se cria um novo jogo, ambos os exércitos ficam com 0 soldados. Os soldados terão de ser criados, um por um, mais tarde. Esta instrução nunca falha, escrevendo sempre a mensagem **"Jogo Iniciado."**

2.2 Comando *CarregaJogo*

Este comando serve para carregar um *script* com um jogo gravado no sistema. O comando é inserido no sistema da forma seguinte (incluíndo a *informação adicional* que o caracteriza):

```
CJ
Nome do Script
```

O *nome do script* é uma String, com o nome do ficheiro onde o *script* a carregar está guardado.

Exemplo (carregamento do *script* com o jogo gravado `BatalhaDaCaparica`):

```
CJ
BatalhaDaCaparica
Jogo carregado.
```

O comando deve ler as jogadas que estão no ficheiro indicado e carregá-las em memória, de forma conveniente. Após o carregamento em memória, deve executar o *script*. A execução de um *script* deve ser efectuada começando na sua primeira jogada, e executando os comandos nele contidos, um por um.

O formato dos ficheiros que contêm *scripts* deve seguir um conjunto de normas, que são explicadas mais adiante. É importante perceber que, ao carregar um jogo, o jogo que possa neste momento estar em memória é deitado fora. Portanto, em cada momento, existe exactamente um jogo em memória, quer ele se esteja a desenvolver interactivamente, quer tenha sido recuperado através do comando *CarregaJogo*.

Após o carregamento de um jogo, os jogadores podem continuar a jogar, a partir do estado em que o jogo tinha ficado, no momento em que foi gravado. ~~Em caso de sucesso no carregamento do jogo, o~~ O programa deve afixar a mensagem **"Jogo carregado."**. ~~Em caso de insucesso, a mensagem a apresentar deverá ser "Erro na leitura do ficheiro.".~~

2.3 Comando *GravaJogo*

Este comando serve para gravar a sequência de jogadas do jogo actual num ficheiro de *script*, que mais tarde poderá ser carregado com o comando *CarregaJogo*. O comando é inserido no sistema da forma seguinte (incluíndo a *informação adicional* que o caracteriza):

```
GJ
Nome do Script
```

O *nome do script* é uma String, com o nome do ficheiro onde o *script* será gravado.

Exemplo (gravação do jogo de nome `BatalhaDaCaparica`):

```
GJ
BatalhaDaCaparica
Jogo gravado com sucesso.
```

O comando deve escrever as jogadas que estão em memória no ficheiro indicado. Note que apenas as jogadas válidas que alterem o estado do jogo devem ser guardadas em memória e depois passadas para ficheiro. Em particular, as jogadas a gravar consistem nos comandos:

- **C**ria**S**oldado (secção 2.4)

- **I**nicia**B**atalha (secção 2.5)
- **M**ove**S**oldado (secção 2.6)
- **AT**aca (secção 2.7)
- **RE**ndição (secção 2.8)

Os restantes comandos não são relevantes para os *scripts* com jogos gravados. O formato dos ficheiros que contêm *scripts* deve seguir um conjunto de normas, que são explicadas mais adiante. É importante perceber que, ao gravar um jogo, o jogo que neste momento está em memória se mantém inalterado, de modo a poder continuar a jogar após a gravação. ~~Caso o jogo seja gravado com sucesso, d~~Deverá ser apresentada ao utilizador a seguinte mensagem: **"Jogo gravado com sucesso."** ~~Caso contrário, a mensagem a apresentar será: "Erro na gravacao do jogo."~~

2.4 Comando *CriaSoldado*

Este comando permite criar um novo soldado numa determinada posição, com uma determinada orientação e perícia inicial e adicionar esse soldado ao respectivo exército. Este comando apenas pode ser invocado antes de se iniciar a batalha. A partir do momento em que começa a batalha, não é possível reforçar os exércitos. O comando é inserido no sistema da forma seguinte (incluíndo a *informação adicional* que o caracteriza):

```
CS
Nome do exército
Nome do soldado
Coordenadas da localização onde o soldado deve ser colocado inicialmente
Orientação inicial
Perícia inicial
```

O *nome do exército* é uma String. Pode ser uma destas duas: *"yodellers"*, ou *"marcianos"*.

O *nome do soldado* é uma String. Por exemplo, *"Private Ryan"*, ou *"Marciano de Niro"*.

As *coordenadas de localização* são dois números inteiros representando, respectivamente, as coordenadas *x* e *y* da localização do soldado.

O valor da *perícia inicial* do soldado é o alcance a que este consegue atingir um adversário. Deve ser um número real, maior que zero.

Exemplo (soldado humano, chamado Chuck Yodel-eh-ih-uh Norris, criado na posição x = 10, y = 20, virado para Norte, com perícia inicial de 15.0):

```
CS
yodellers
Chuck Yodel-eh-ih-uh Norris
10 20
Norte
15
Soldado adicionado com sucesso ao exercito de yodellers.
```

Se tudo correr bem, o soldado será adicionado ao exército de humanos, e a mensagem **"Soldado adicionado com sucesso ao exercito de yodellers."** é afixada. Bem entendido, a mensagem será semelhante no caso do exército de marcianos, mas nesse caso termina em **"exercito de marcianos."** Se o nome do soldado já existir (em qualquer dos exércitos), a orientação não for uma das

quatro possíveis, ou a perícia não for maior que zero, o comando falha, com a mensagem **"Erro na criacao do soldado."**

2.5 Comando *IniciaBatalha*

Este comando permite iniciar uma nova batalha, com os exércitos criados até à sua chamada com o comando *CreateSoldier*. Só se pode iniciar uma batalha uma vez por jogo, depois de todos os soldados de ambos os exércitos estarem criados e antes de qualquer desses soldados se mexer, ou atacar. O comando é inserido no sistema da forma seguinte (não requer informação adicional):

```
IB
```

Exemplo (início de batalha):

```
IB
Ack ack ack ack-ack! Viemos em Paz!
```

Se tudo correr bem, a batalha começa e a mensagem **"Ack ack ack ack-ack! Viemos em Paz!"** é afixada. Se não for possível iniciar a batalha, por exemplo por esta já ter começado, ou por não haver soldados em algum dos exércitos, o comando falha, com a mensagem **"Erro na inicializacao da batalha."**

2.6 Comando *MoveSoldado*

Este comando permite movimentar um soldado, com uma determinada orientação e percorrendo uma determinada distância. Em cada jogada, cada soldado apenas se pode mover, no máximo, 5 metros, seja em que sentido for. No final da deslocação, o soldado moveu-se a distância indicada, no sentido de orientação indicado, tendo ficado virado para essa orientação. O comando é inserido no sistema da forma seguinte (incluíndo a *informação adicional* que o caracteriza):

```
MS
Nome do exército
Nome do soldado
Orientação na deslocação
Distância a percorrer
```

O *nome do exército* é uma String. Pode ser uma destas duas: *"yodellers"*, ou *"marcianos"*.

O *nome do soldado* é uma String. Por exemplo, *"Private Ryan"*, ou *"Marciano de Niro"*.

A *orientação* é uma String. Pode ser uma destas quatro: *"Norte"*, *"Sul"*, *"Este"*, ou *"Oeste"*.

A *distância a percorrer* é um inteiro. Pode ser um valor inteiro entre *-5* e *5*. Valores positivos indicam que o soldado se desloca na respectiva orientação. Valores negativos significam que o soldado se desloca "de costas" (ou seja, como se estivesse a andar em marcha atrás). Isto pode ser útil, por exemplo, para recuar alguns metros, sem virar as costas ao adversário. Em qualquer dos casos, o soldado fica virado para *orientação*. Por exemplo, se a ordem for para se deslocar 3 metros para norte, isso implica somar 3 à coordenada y do soldado. Se a ordem for para se deslocar -2 metros para Oeste, isso significa somar 2 à coordenada x do soldado, mas o soldado ficará virado para Oeste, no final da deslocação.

Exemplo (mover soldado marciano de nome Marcirina Shayk 2 metros para sul):

```
MS
marcianos
Marcirina Shayk
Sul
```

```
2
Soldado movimentado com sucesso.
```

Caso tudo corra bem, deverá ser afixada a mensagem **"Soldado movimentado com sucesso."** Se não for a vez de jogar deste exército, o nome do soldado não existir no exército, o soldado com esse nome estiver morto, ou o valor da deslocação não for maior ou igual que -5 e menor ou igual que 5, o comando falha e uma mensagem de erro deve ser exibida: **"Erro na movimentacao do soldado."**

2.7 Comando *ATaque*

Este comando faz com que um soldado tente atacar. A forma de ataque é diferente, consoante a espécie do soldado. Já sabe, os *yodellers* cantam, os marcianos dão tiros. Vejamos a diferença, com mais detalhe.

Quando recebem a ordem de ataque, os humanos cantam um *yodel*. Este ataque mata todos os marcianos que estejam a uma distância igual ou inferior à perícia do yodeller. Por exemplo, se o *yodeller* tiver uma perícia com o valor 15, todos os marcianos que estiverem a uma distância menor ou igual a 15 metros do *yodeller* serão mortos pelo ataque. O ataque não produz qualquer efeito sobre soldados humanos que estejam por perto, por muito desafinado que o soldado cante... ☺ Por cada marciano morto no ataque, o *yodeller* recebe 5% de bónus. Por exemplo, se num ataque um *yodeller* com perícia 10,0 matar 3 marcianos, receberá mais 3*0,05*10,0 = 1,5 pontos a somar à sua perícia, passando assim a ter uma perícia de 11,5. No próximo ataque, tirará partido da nova perícia, e será sobre a nova perícia que os eventuais novos bónus serão calculados. Se não existir nenhum marciano no raio de alcance do ataque do *yodeller*, o ataque falha e nenhum marciano é abatido.

Quando recebem ordem de ataque, os marcianos tentam disparar sobre o humano mais próximo que esteja no seu campo de visão. Por exemplo, um marciano virado para norte apenas pode alvejar humanos que estejam a norte dele, ou seja, que tenham a componente *y* da sua posição superior à componente *y* da posição do marciano. Se existir um humano no campo de visão do marciano e a distância entre ambos for igual ou inferior à perícia do marciano, o ataque terá sucesso e o marciano receberá um bónus de 5%. Se não existir nenhum humano à vista do marciano, ou nenhum dos que estão à vista estiver suficientemente próximo, o marciano falha o tiro. Em todo o caso, por mais confuso que possa parecer o campo de batalha, não há qualquer risco de um marciano acidentalmente acertar noutro marciano.

Quer seja humano, quer seja marciano, pode sempre acontecer que o jogador tente fazer um ataque numa situação em que o soldado escolhido não esteja em condições de concluir o ataque com sucesso. Nesse caso, nenhum soldado adversário será atingido e o nível de perícia do atacante não sofrerá qualquer alteração. Em qualquer ataque, apenas se consegue atacar adversários que ainda estejam vivos. Como queremos saber quem são os combatentes mais valiosos na batalha, convém ir guardando informação do total de adversários obtidos por cada soldado.

O comando é inserido no sistema da forma seguinte (incluíndo a *informação adicional* que o caracteriza):

```
AT
Nome do exército
Nome do soldado
```

O *nome do exército* é uma String. Pode ser uma destas duas: *"yodellers"*, ou *"marcianos"*.

O *nome do soldado* é uma String. Por exemplo, *"Private Ryan"*, ou *"Marciano de Niro"*.

Exemplo (o bravo soldado Schweik, *yodeller*, recebe ordem para atacar):

```
AT
yodellers
bravo soldado Schweik
Temos de atacar agora, chefe! Aniquila! MATA! MATA! MATA! MATA!
```

```
0 soldados abatidos.
```

Caso tudo corra normalmente, a mensagem a afixar será o grito de guerra de cada exército (**"Temos de atacar agora, chefe! Aniquila! MATA! MATA! MATA! MATA!"**), no caso dos humanos, ou **"Ack ack ack ack-ack! Nao fujam! Nos somos vossos amigos!"**, no caso dos marcianos), seguido de **"N soldados abatidos"**, em que **N** é um número inteiro, maior ou igual que 0. No exemplo anterior, apesar de o ataque ser bem efectuado, não havia marcianos suficientemente perto para que o ataque surtisse melhor efeito. Se não for a vez de jogar deste exército, o nome do soldado não existir no exército, ou o soldado com esse nome estiver morto, o comando falha e uma mensagem de erro deve ser exibida: **"Erro no ataque do soldado."**

Pode acontecer que, na sequência de um ataque, o exército adversário seja derrotado, ou seja, todos os soldados que ainda restavam desse exército sejam mortos. Se isso acontecer, o programa deve afixar mensagens apropriadas. Em primeiro lugar, afixa-se a mensagem do exército derrotado. A seguir, a do vencedor. Cada exército tem as suas mensagens de vitória e derrota, que a seguir reproduzimos:

Marcianos

"Ack ack ack ack-ack! Estes tipos cantam TAAAAAO mal! :-(" (derrota)

"Ack ack ack ack-ack! Calem-se de uma vez! :-)" (vitória)

Humanos

"Cheira-me a esturro... :-(" (derrota)

"Primeiro, conquistaremos Marte. Depois, os IDOLOS! HURRAH! :-)" (vitória)

2.8 Comando de *REndição*

Este comando permite a qualquer jogador render-se, dando a vitória ao adversário.

O comando é inserido no sistema da forma seguinte (incluíndo a *informação adicional* que o caracteriza):

```
RE
Nome do exército
```

O *nome do exército* é uma String. Pode ser uma destas duas: *"yodellers"*, ou *"marcianos"*.

Exemplo (o exército marciano rende-se):

```
RE
marcianos
Ack ack ack ack-ack! Estes tipos cantam TAAAAAO mal! :-(
Primeiro, conquistaremos Marte. Depois, os IDOLOS! HURRAH! :-)
```

Após a rendição de um exército, é escrita a mensagem de derrota do exército derrotado, seguida da mensagem de vitória do exército vencedor. As mensagens são as que se seguem:

Marcianos

"Ack ack ack ack-ack! Estes tipos cantam TAAAAAO mal! :-(" (derrota)

"Ack ack ack ack-ack! Calem-se de uma vez! :-)" (vitória)

Humanos

"Cheira-me a esturro... :-(" (derrota)

"Primeiro, conquistaremos Marte. Depois, os IDOLOS! HURRAH! :-)" (vitória)

Se não for a vez de jogar deste exército, o comando falha e uma mensagem de erro deve ser exibida: **"Erro na tentativa de rendicao."**

2.9 Comando *ListaCampoDeBatalha*

Este comando serve para escrever na consola o estado actual do campo de batalha. Começa por descrever os *yodellers* que ainda estão vivos, seguindo-se os marcianos que também estejam vivos. É indicado ao sistema da forma seguinte:

```
LCDB
```

Para cada soldado, humano ou marciano, é listada a seguinte informação:

Nome do soldado

Coordenada X

Coordenada Y

Orientação actual

Experiência actual

Total de adversários abatidos

Exemplo (numa altura em que sobram apenas três humanos e um marciano):

```
LCDB
yodellers
Chuck Norris;2;4;Norte;15.3;12
Julia Pinheiro;4;7;Este;298.7;214
Sylvester Stallone;24;8;Sul;8.2;3
marcianos
Marciano Surdo Mas Cheio de Pontaria;530;-5;Oeste;603.6;587
```

A informação relativa a cada soldado deve ser listada numa linha separada, por ordem alfabética do nome do soldado. Os dados de cada soldado devem aparecer separados pelo carácter **';'**. Se algum exército não tiver soldados, na linha a seguir ao nome do exército deve aparecer a mensagem **"Exercito vazio."**

2.10 Comando *ListaMortos*

Este comando permite listar os soldados dos dois exércitos que perderam a vida durante a batalha. Novamente, começamos pelos *yodellers*, seguindo-se os marcianos, em ambos os casos ordenados alfabeticamente por nome. É indicado ao sistema da forma seguinte:

```
LM
```

Para cada soldado, humano ou marciano, é listada a seguinte informação:

Nome do soldado

Total de adversários abatidos

Exemplo (três soldados humanos, e dois marcianos, todos mortos):

```
LM
yodellers
Aquiles;0
Frank Sinatra;2
Hercules;0
marcianos
Marcialexandre O Grande;3
Marciatila O Huno;0
```

A informação relativa a cada soldado deve ser listada numa linha separada, por ordem alfabética do nome do soldado, sendo os dados separados pelo carácter ';'. Caso não haja baixas em algum dos exércitos, na linha a seguir ao nome do exército deve-se escrever **"OK!"**

2.11 Comando *ListaHeróis*

Este comando mostra o nome do soldado de cada exército que tenha abatido mais adversários. Em caso de empate, escolhe, de entre os melhores, aquele que tenha maior perícia. Existindo ainda um empate, escolhe um qualquer, de entre os melhores. Note que os heróis podem estar vivos ou mortos.

É indicado ao sistema da forma seguinte:

```
LH
```

Para cada soldado, humano ou marciano, é listada a seguinte informação:

Nome do soldado

Total de adversários abatidos

Exemplo:

```
LH
yodellers
Julia Pinheiro;214
marcianos
Marciano Surdo Mas Cheio de Pontaria;587
```

A informação relativa a cada soldado deve ser listada numa linha separada, sendo os dados separados pelo carácter ';'. Se não existirem soldados num exército (vivos ou mortos), a mensagem a afixar, na linha a seguir ao respectivo exército, deve ser **"Exercito vazio."**

2.12 Comando *Ajuda*

Este comando permite listar os comandos permitidos pelo sistema. É indicado da forma seguinte:

```
A
```

Exemplo (pedir lista de comandos disponíveis):

```
A
```

O sistema deve apresentar a lista dos comandos disponíveis.

2.13 Comando *Sair*

Este comando permite terminar a execução do programa. É indicado da forma seguinte:

```
S
```

Exemplo:

```
S
```

O sistema deve apresentar uma mensagem de despedida (**"Ate breve!"**), terminando de seguida a execução do seu programa.

3 Exemplos

3.1 Sessão interactiva

O exemplo começa por mostrar as opções disponíveis, com o comando `A`. A seguir, começa um novo jogo, com a criação de dois exércitos, um com três humanos, outro com dois marcianos. No exemplo, os marcianos começam por conseguir abater dois humanos mas, no final, a vitória vai para os humanos, in extremis. Pelo meio, acontece um ou outro erro de interacção, com as respectivas mensagens. Não é objectivo deste exemplo cobrir exaustivamente todas as situações possíveis, mas deverá testar todas no seu projecto.

```
A
Marte Ataca! Viemos em paz! Precisa de ajuda?
NJ – Novo Jogo
CJ – Carrega Jogo
GJ – Grava Jogo
CS – Cria Soldado
IB – Inicia Batalha
MS – Move Soldado
AT – ATaca
RE - REndicao
LCDB – Lista Campo De Batalha
LM – Lista Mortos
LH – Lista Herois
A - Ajuda
S – Termina Jogo
NJ
CS
yodellers
Joseh-ih-Yodel-eh-ih-yodel-eh-ih-uh Cid
12 18
Este
```

15
Soldado adicionado com sucesso ao exercito de yodellers.
CS
yodellers
Chuck Yodel-eh-ih-uh Norris
10 20
Norte
15
Soldado adicionado com sucesso ao exercito de yodellers.
yodellers
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira
-3 18
Este
50
Soldado adicionado com sucesso ao exercito de yodellers.
CS
yodellers
Joseh-ih-*Yodel-eh*-ih-*yodel-eh*-ih-*uh* Cid
12 18
Este
15
Erro na criacao do soldado.
CS
marcianos
Joseh-ih-*Yodel-eh*-ih-*yodel-eh*-ih-*uh* Cid
212 18
Este
15
Erro na criacao do soldado.
CS
marcianos
Lucky Luke Marciano
47 20
Oeste
40
Soldado adicionado com sucesso ao exercito de marcianos.
CS
marcianos

```
ET O Extraterrestre
45 12
Sul
10
Soldado adicionado com sucesso ao exercito de marcianos.
IB
Ack ack ack ack-ack! Viemos em Paz!
LCDB
yodellers
Chuck Yodel-eh-ih-uh Norris;10;20;Norte;15;0
Joseh-ih-Yodel-eh-ih-yodel-eh-ih-uh Cid;10;18;Este;15
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira;-3;19;Este;50
marcianos
ET O Extraterrestre;45;12;Sul;10
Lucky Luke Marciano;47;20;Oeste;40
LM
yodellers
OK
marcianos
OK
AT
marcianos
Lucky Luke Marciano
Ack ack ack ack-ack! Nao fujam! Nos somos vossos amigos!
1 soldados abatidos.
LM
yodellers
Joseh-ih-Yodel-eh-ih-yodel-eh-ih-uh Cid;0
marcianos
OK
MS
yodellers
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira
Este
5
Soldado movimentado com sucesso.
AT
marcianos
```

```
Lucky Luke Marciano
Ack ack ack ack-ack! Nao fujam! Nos somos vossos amigos!
1 soldados abatidos.
AT
marcianos
Lucky Luke Marciano
Erro no ataque do soldado.
GJ
GloriosaBatalha
Jogo gravado com sucesso.
AT
yodellers
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira
Temos de atacar agora, chefe! Aniquila! MATA! MATA! MATA! MATA!
2 soldados abatidos.
Ack ack ack ack-ack! Estes tipos cantam TAAAAAO mal! :-(
Primeiro, conquistaremos Marte. Depois, os IDOLOS! HURRAH! :-)
LCDB
yodellers
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira;2;19;Este;55.125
marcianos
Exercito vazio.
LM
yodellers
Chuck Yodel-eh-ih-uh Norris;0
Joseh-ih-Yodel-eh-ih-yodel-eh-ih-uh Cid;0
marcianos
ET O Extraterrestre;0
Lucky Luke Marciano;2
Exercito vazio.
NJ
CJ
GloriosaBatalha
RE
yodellers
Cheira-me a esturro... :-(
Ack ack ack ack-ack! Calem-se de uma vez! :-)
```

```
MS
marcianos
ET O Extraterrestre
Norte
2
Erro na movimentacao do soldado.
S
Ate breve!
```

3.2 Ficheiro de script

O exemplo que se segue mostra um ficheiro de script, gerado durante o exemplo anterior ("GloriosaBatalha"). Os comandos transcritos para o script são apenas os de *criação de soldados*, *início de batalha*, *movimentação de soldados*, *ataques* e *rendição* (este último não aparece neste exemplo, mas podia aparecer). A ordem pela qual estes comandos aparecem transcritos é exactamente a mesma pela qual foram dados, inicialmente. Além disso, como pode observar, os comandos que não foram executados com sucesso não são transcritos para o ficheiro. O ficheiro começa com um número inteiro a indicar o número de operações contido no ficheiro.

```
9
CS
yodellers
Joseh-ih-Yodel-eh-ih-yodel-eh-ih-uh Cid
12 18
Este
15
CS
yodellers
Chuck Yodel-eh-ih-uh Norris
10 20
Norte
15
CS
yodellers
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira
-3 18
Este
50
CS
marcianos
Lucky Luke Marciano
47 20
```

```
Oeste
40
CS
marcianos
ET O Extraterrestre
45 12
Sul
10
IB
AT
marcianos
Lucky Luke Marciano
MS
yodellers
Tonye-ih-yodel-eh-yodel-eh-ih-uh Carreira
Este
5
AT
marcianos
Lucky Luke Marciano
```

4 MÉTODO DE DESENVOLVIMENTO DO PROJECTO

Nesta secção, vamos discutir a metodologia de execução sugerida para o projecto. É muito importante ser metódico em qualquer actividade com alguma duração, em particular, num projecto de desenvolvimento de software, com é o caso deste trabalho de "Introdução à Programação". Existe uma sequência de actividades que devem ser feitas de forma disciplinada, para se obter um "produto" final de qualidade, reduzindo os erros, facilitar a sua detecção, facilitar a legibilidade do programa, e ... ter uma boa nota!

4.1 FASE 1 - Compreensão e Esclarecimento do enunciado e dos objectivos do projecto

Leia bem o enunciado, eliminando todas as questões que lhe ocorram. Se persistirem dúvidas, coloque-as por escrito. Mantenha um documento escrito chamado "dúvidas sobre o trabalho / funcionamento do jogo "*Mars Attacks!*", com as suas dúvidas e com as respostas que for obtendo. É essencial saber, em cada momento, que dúvidas tem, e quais as respostas às dúvidas que teve, mas já esclareceu. Como esclarecer dúvidas? Com os docentes e monitores da disciplina. Aproveite os horários de dúvidas e as aulas práticas.

Nesta fase eliminará as maiores dúvidas sobre o enunciado e sobre o que se pretende, mas claro que vão persistir dúvidas e surgir novas dúvidas durante as fases seguintes. Por isso, prepare-se para manter o seu documento de dúvidas até ao fim do prazo de entrega.

4.2 FASE 2 - Análise da estrutura geral do sistema

Nesta segunda fase, terá como objectivo definir a estrutura global do seu programa. Como bem sabe, um programa em Java consiste num conjunto de classes Java, cujos objectos representam as várias entidades do

domínio do problema, e numa classe especial Main, que apenas contém o programa principal (método main), e algumas funções auxiliares (por exemplo, gestão do input / output). No caso que temos em mão, vai existir uma classe do domínio para cada entidade necessária para construir o seu sistema, por exemplo, *Yodeller* (soldado humano), *Martian* (soldado marciano), *Script* (sequência de comandos dados aos soldados), etc, ...

Os objectos dessas classes representarão o conceito, manterão a informação relativa a cada indivíduo desse conceito, e fornecerão um conjunto de operações apropriadas. Defina nesta fase quais são as várias classes de que necessita e quais são as operações (métodos e construtores) que cada classe necessita, no contexto do sistema em causa. Certifique-se de que as operações que identificou pertencem mesmo à classe em que as colocou. Identifique também as variáveis e constantes que achar necessárias para representar o estado de cada objecto de cada classe, assim como o seu tipo.

Não se esqueça de pensar bem como representar a informação dentro de cada objecto. Tente explorar o facto de se poderem usar objectos de uma classe como valores de variáveis de outros objectos.

Note que nesta fase não terá que programar, mas apenas que definir, para cada classe, qual o conjunto de operações (métodos e construtores), assim como as variáveis e constantes que achar necessárias para representar o estado de cada objecto dessa classe.

Quando tiver dúvidas sobre onde colocar uma determinada operação (em que classe), consulte os docentes da disciplina.

Para que tudo fique bem documentado, para cada variável indique o seu tipo e explique a finalidade. Para cada método indique o tipo do seu resultado e o tipo dos seus parâmetros (se existirem). Explique ainda a sua finalidade (para que serve) de forma clara e intuitiva. Indique ainda que métodos são modificadores e que métodos são de consulta (observadores).

Para além das classes mais "óbvias", provavelmente precisará de classes auxiliares, para programar, por exemplo, colecções de objectos de certo tipo. Por exemplo a sequência de instruções num *script* deverá ser representada por um objecto que mantenha uma colecção de comandos. Cada classe que definir para representar tais colecções deve ter um conjunto de operações associadas que inclui em geral uma operação para inserir um elemento da colecção, pesquisar um elemento na colecção (dado o nome, por exemplo), etc., etc. Pense bem que colecções precisa para programar o sistema. Cada tipo de colecção será representado por uma classe distinta.

O resultado desta fase é um esqueleto de programa em Java, em que as classes só têm variáveis, métodos (cujo corpo é vazio), e comentários explicativos. É um programa que provavelmente não passa no compilador (que dá erros, pois faltam coisas), mas que é o esqueleto da sua solução. E o esqueleto é o mais importante. Se for sólido e correcto, a aplicação vai-se manter de pé quando concluída, caso contrário, vai sair uma alforreca.

No final desta fase, mostre o resultado da sua planificação aos docentes da disciplina. Estes poderão dar alguns conselhos, ou transmitir-lhe confiança sobre a adequação da sua proposta.

Apenas depois desta fase estar concluída é que deve começar a programar.

4.3 FASE 3 - Construção do sistema

Nesta fase, vai começar a programar o seu sistema. Não tente fazer tudo ao mesmo tempo. Por exemplo, deixe a parte de leitura e escrita no ficheiro para o fim: pode testar a maior parte das funcionalidades sem isso estar feito.

Defina a sequência de tarefas que vai ter que fazer para programar o sistema. Sugerimos aqui uma sequência possível, mas pode escolher outra, se preferir. Esta parece-nos a mais indicada.

1. Pode começar por programar o interpretador de comandos, na classe `Main`, que permite ao sistema interagir com o utilizador. Isso é algo simples, e que fica feito de uma vez por todas. O objectivo desta fase é um obter um programa que aceita os comandos e os dados associados, apesar de não fazer nada (excepto sair, com o comando *Sair*). A vantagem de construir logo o interpretador de comandos, é que vai poder começar a testar as várias funcionalidades usando a linha de comando, à medida que as vai implementando.

2. Programe tudo o que necessita para manter a informação sobre um soldado, quer para os humanos, quer para os marcianos. Teste a(s) classe(s) que representa(m) os soldados com um main especial, criado para esse efeito, onde pode e deve testar aturadamente os construtores, selectores e modificadores da(s) interface(s) da(s) classe(s).

3. Programe tudo o que seja necessário para manter informação sobre o estado dos exércitos e operações para criar soldados. Para testar bem esta parte, pode também implementar a operação de listar o campo de batalha, para poder consultar o estado dos exércitos. Teste esta operação de listar definindo "manualmente" o estado dos exércitos. Numa primeira fase, poderá criar os exércitos desordenados, para a seguir implementar a sua ordenação por ordem alfabética de nome. Pode experimentar tudo isto através de programas de teste. Se tudo funcionar bem, pode integrar no seu interpretador de comandos, que nesta fase deverá permitir criar os exércitos e listar os exércitos.

4. Agora que já consegue inicializar os dois exércitos e listar ambos, correctamente, programe tudo o necessário para poder movimentar os soldados. Novamente, socorra-se da operação para listar o campo de batalha para visualizar o resultado da implementação das movimentações. Recorra a programas de teste para experimentar esta funcionalidade, e integre-a no interpretador de comandos.

5. Provavelmente, nesta altura vai querer programar as operações de ataque. O princípio é sempre o mesmo. Acrescente mais uma funcionalidade, integre-a no interpretador de comandos, e vá testando tudo.

6. Progressivamente, vá introduzindo na sua aplicação a capacidade de executar mais tipos de instruções. Mais uma vez, não tente fazer tudo de uma vez. Vá acrescentando e testando gradualmente o seu programa, usando a técnica de desenvolvimento que sugerimos acima.

7. Quando terminar a fase anterior, já grande parte do projecto estará concluído. Revisite as várias funcionalidades e verifique que funciona tudo como previsto.

4.4 Indicações gerais

É muito importante tentar ter sempre um programa funcional, que compile sem erros e faça qualquer coisa, mesmo que ainda não tudo o que é requerido. Assim terá mais segurança e uma base sólida para partir para a fase seguinte. É a regra da "versão estável", que deve ser seguida por qualquer desenvolvedor de software. Tenha sempre uma versão estável do seu sistema! Nunca comece a trabalhar em novas fases, sem que as anteriores estejam sólidas como aço.

Programe em grupo com o seu colega de grupo. Sentados à volta do mesmo computador e discutindo a solução. Não tente dividir o trabalho em partes muito separadas, depois as coisas não colam, e martelar peças para encaixarem à força não vai ser boa ideia.

Para facilitar o desenvolvimento do seu sistema, e eliminar erros desnecessários, deverá ter em atenção os seguintes aspectos:

1. Não se esqueça de mostrar o resultado da FASE 1 a um docente ou monitor. Não comece a programar antes de terminar a FASE 1!

2. Teste as suas classes método a método. Se for preciso crie pequenos programas principais para testar (os métodos de) cada classe.

3. Não escreva quilómetros de código. Cada método não deve ter mais que 25 linhas de código, e na média deve ter cerca de 15, ou mesmo menos. Defina métodos auxiliares (privados) sempre que tal for aconselhável para tornar o seu programa legível.

4. Use bons nomes para os identificadores. Siga as regras de estilo disponíveis no Moodle.

5. Quando definir a classe `Main` e o respectivo método `main`, escreva um programa que execute toda a interacção com o utilizador, mas que não realize nenhuma tarefa na realidade – como já sugerido acima na FASE 1(1). Vá adicionando tarefas ao método main conforme for desenvolvendo as várias funcionalidades do sistema. A classe Main deverá ser organizada em vários métodos estáticos que implementam cada comando do sistema. Lute ferozmente contra a tentação de programar um método main muito grande, mas incompreensível. Não quer que os docentes fiquem esmagados com quilómetros de programa, pois não?

6. A classe `Main` é a única que pode utilizar operações de *input* / *output* de ou para o ecrã ou de e para ficheiros (`println`, etc). Nenhum método de uma classe auxiliar pode conter operações de *input* / *output*. Quando chegar a altura, discuta com os docentes como ler e escrever a informação necessária para o ficheiro.

7. Para os objectos que contêm colecções de outros objectos (guardados em vector), crie métodos de consulta e modificadores dos elementos dos vectores que sejam necessários. Nunca permita o acesso externo directo a um vector que seja criado dentro de um objecto.

8. Todas as variáveis de estado dos objectos TÊM que ser privadas (`private`). Isto inclui as variáveis que sejam de tipo vector.

9. Não se esqueça da regra da "versão estável". No fim de cada fase de desenvolvimento ou alteração ao programa, ele deve compilar e funcionar como se espera nessa fase. Nunca esteja muitos dias sem que o seu programa compile sem erros, ou não faça o que era previsto fazer nessa fase! As versões estáveis são uma rede de segurança a que se pode agarrar. Nunca apague o código de cada versão estável, para poder voltar a essa se versão, se se perder no desenvolvimento ou se começar a ficar com um programa muito confuso.

10. Se você não percebe como o SEU programa funciona, muito menos os docentes o vão perceber quando o forem avaliar! Não escreva código que não percebe como funciona. Em caso de dúvida, estude, consulte livros, os docentes, os colegas, quem quiser. Mas não programe por tentativa e erro: isso não vai levar a lado nenhum, senão ao chumbo. Vai ver que à medida que vai ganhando prática, vai cometendo menos erros e percebendo melhor o que está a acontecer. MUITO IMPORTANTE! Comente sempre o seu código.

11. Quando o compilador ou o seu programa der um erro, tente perceber bem porque o erro aconteceu, para não repetir o mesmo erro muitas vezes.

12. Esta lista de indicações pode ainda crescer um pouco, veja as novas versões que vão saindo, pois podem conter informações úteis.