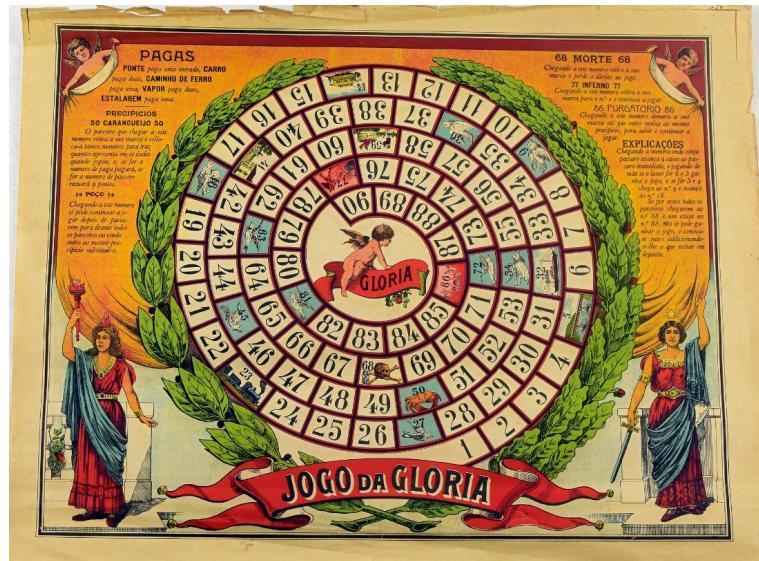


Departamento de Informática

Introdução à Programação

Jogo da Glória

Enunciado do Primeiro Trabalho



Ano letivo 2022/23

Preâmbulo

O trabalho é **individual** e é entregue no Mooshak, que aceita submissões até às **20h00** do dia **7 de novembro de 2022**. Leia este enunciado com a máxima atenção, para perceber muito bem o problema e todos os detalhes sobre o concurso do Mooshak e os critérios de avaliação do trabalho.

1 Conceitos e Objetivo do Trabalho

O Jogo da Glória é um jogo de tabuleiro que marcou a infância de muitos portugueses. O **tabuleiro** tradicional, apresentado na **Figura 1**, é composto por **90 casas**, numeradas de 1 a 90. Cada **jogador** tem uma *marca* (como as ilustradas na **Figura 2**), que começa na **casa 1**. Quando é a sua vez de jogar, o jogador lança **dois dados** para se determinar quantas casas a marca avança. Ganha quem conseguir atingir primeiro a casa 90.

Para haver mais emoção, há casas especiais, chamadas *multas*, *precipícios* e *pássaros*. Se a marca para numa multa, o jogador fica impedido de lançar os dados (uma ou duas vezes) quando voltar a ser a sua vez de jogar. A penalização sofrida quando a marca para num precipício varia muito com o precipício. Por exemplo, o caranguejo faz a marca recuar (em vez de avançar). Ao contrário das multas e dos precipícios, parar numa casa com pássaro é uma sorte, porque se avança imediatamente mais 9 casas.

O objetivo deste trabalho é programar em Java uma versão muito simplificada do Jogo da Glória, na qual há exatamente três jogadores. Com a informação sobre as cores das suas marcas, o tabuleiro e a sequência de lançamentos dos dados, o programa deve ser capaz de indicar, em qualquer momento do jogo, em que casa se encontra cada jogador e, caso o jogo ainda não tenha terminado, quem é o próximo jogador a lançar os dados e se um dado jogador está, ou não, impedido de os lançar quando chegar a sua vez de jogar.



Figura 1: Tabuleiro do Jogo da Glória



Figura 2: Marcas

2 Regras do Jogo

Competem entre si três jogadores, cada um com uma marca diferente. Para simplificar, o jogador é identificado pela cor da sua marca e diz-se que está numa casa se a sua marca estiver nessa casa. Antes do jogo começar, é definida a ordem pela qual os jogadores jogam. Joga um jogador de cada vez. O jogador que joga vai rodando, respeitando a ordem previamente definida e considerando-se que depois do último vem o primeiro.

Por exemplo, se os jogadores tiverem marcas com as cores R , G e B (*red*, *green* e *blue*) e for essa a ordem pela qual jogam: o primeiro a jogar é o jogador (que tem a marca) R ; depois de R jogar, é a vez de G jogar; depois de G jogar, é a vez de B jogar; depois de B jogar, é novamente a vez de R jogar.

O **tabuleiro** tem um número C de casas, numeradas de 1 a C (com $10 \leq C \leq 150$). Quando o jogo começa, todas as marcas estão na casa 1 e o objetivo é atingir a casa C . Qualquer casa tem um, e um só, tipo. Esse tipo pode ser *normal*, *pássaro*, *multa* ou *precipício*. Pássaros, multas e precipícios só podem ocorrer entre as casas 2 e $C - 1$. Nesse intervalo, todas as casas “múltiplas de 9” (as casas 9, 18, 27, etc., sem ultrapassar $C - 1$) têm pássaro.

Quando um **jogador** está numa casa c , lança os dois **dados** e saem p pintas, o jogador *avança p casas* (onde $1 \leq c < C$ e $2 \leq p \leq 12$). Isto significa que o jogador passa a estar na casa $c + p$, *exceto se essa casa não existir* (por ser maior do que C), caso em que o jogador fica na casa C . De forma semelhante, um jogador que esteja numa casa c e que *recue p casas* passa a estar na casa $c - p$, *exceto se essa casa não existir* (por ser menor do que 1); nesse caso, o jogador vai para a casa 1.

No início do jogo, todos os jogadores estão na casa 1 e têm uma pena de zero jogadas (nunca foram multados).

Quando chega a sua vez de jogar, o comportamento do jogador depende do valor da sua pena. A jogada pode alterar a casa onde está e a pena que tem.

- Se o valor da pena for **positivo**, o jogador está **impedido de lançar os dados** (tendo de os passar ao “parceiro do lado”). O jogador permanece na mesma casa e o valor da sua pena baixa uma unidade. Considera-se que **estas jogadas ocorrem assim que é possível**.
- Se o valor da pena for **zero**, o jogador lança os dados e a sua marca avança tantas casas quanto o número total de pintas que saíram.
 - Se a marca ficar numa **casa normal**, a pena do jogador não é alterada. **Se essa casa for C , o jogador ganha o jogo.**
 - Se a marca ficar num **pássaro**, o jogador avança (mais) 9 casas. A sua pena não é alterada.
 - Se a marca ficar numa **multa**, a pena do jogador aumenta duas unidades.
 - Se a marca ficar num **precipício**, o jogador volta à casa onde estava (antes de ter lançado os dados) e depois recua tantas casas quanto o número total de pintas que saíram. A sua pena não é alterada.

O jogo termina assim que um dos jogadores chegar à casa C . Nesse momento, nenhum jogador pode lançar os dados e todas as marcas permanecem nas casas onde se encontram.

3 Especificação do Sistema

Pretende-se que a interface da aplicação seja simples, para poder ser utilizada em ambientes diversos e permitir automatizar o processo de teste. Por estes motivos, a entrada e a saída têm de respeitar o formato preciso especificado nesta secção. **Pode assumir que o input obedece às restrições de valor e de formato indicadas, ou seja, que o utilizador não comete erros, para além dos previstos neste enunciado.**

O programa lê linhas da entrada padrão (`System.in`), escreve linhas na saída padrão (`System.out`) e distingue maiúsculas de minúsculas (por exemplo, as palavras “exit” e “Exit” são diferentes).

Forma do Input

A entrada tem a seguinte estrutura (onde o símbolo \leftrightarrow representa uma mudança de linha):

```

cores ←
numCasas ←
M ←
multa1 multa2 ... multaM ←
P ←
prec1 prec2 ... precP ←
comando ←
comando ←
.....
comando ←
exit ←

```

onde:

- **cores** é uma sequência de três **letras** maiúsculas diferentes (de 'A' a 'Z');
- **numCasas** é um número **inteiro** entre 10 e 150;
- **M** e **P** são números **inteiros** entre 1 e um terço de **numCasas**;
- **multa₁, ..., multa_M** são **M** números **inteiros** por ordem crescente, **separados por um espaço**;
- **prec₁, ..., prec_P** são **P** números **inteiros** por ordem crescente, **separados por um espaço**;
- os números **multa₁, ..., multa_M, prec₁, ..., prec_P** variam entre 2 e **numCasas** - 1, são todos distintos e nenhum é múltiplo de 9;
- **comando** é um de quatro comandos (chamados **comando-jogador**, **comando-casa**, **comando-estado** e **comando-lançamento**) ou um comando **inválido**, explicados a seguir.

A primeira linha da entrada tem as **cores dos jogadores**, pela ordem em que jogam. A segunda linha tem o **número de casas do tabuleiro**. A terceira e a quinta linhas indicam **quantas casas são multas (M)** e **quantas casas são precipícios (P)**. A quarta e a sexta linhas especificam, respetivamente, **quais são as casas onde se aplica a multa** e **quais são as casas onde há precipício**.

Segue-se um número arbitrário de comandos. A última linha tem um comando especial, o **comando-sair**, que só pode ocorrer na última linha porque faz terminar a execução do programa.

Comando-Jogador

O comando-jogador indica que se pretende saber **quem é o próximo jogador a lançar os dados**, no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-jogador têm:

```
player ←
```

O programa escreve uma linha na consola, distinguindo dois casos:

- Se o jogo já tiver terminado, a linha tem:

```
The game is over ←
```

- Nos restantes casos, a linha tem a seguinte forma, onde **corJogador** representa a cor do próximo jogador a lançar os dados:

```
Next to play: corJogador ←
```

Comando-Casa

O comando-casa indica que se pretende saber **em que casa está o jogador com a cor dada**, no momento corrente do jogo. Este comando não altera o estado do jogo. As linhas com comandos-casa têm a seguinte forma (com um espaço a separar as duas componentes):

```
square corJogador←
```

onde:

- *corJogador* é uma sequência não vazia de caracteres.

O programa escreve uma linha na consola, distinguindo dois casos:

- Se *corJogador* não for a cor de um dos jogadores, a linha tem:

```
Nonexistent player←
```

- Nos **restantes casos**, a linha tem a seguinte forma, onde *casaJogador* representa a casa onde o jogador referido no comando se encontra:

```
corJogador is on square casaJogador←
```

Comando-Estado

O comando-estado indica que se pretende saber se, no momento corrente do jogo, o jogador com a cor dada pode, ou não, lançar os dados, quando (e se) chegar a sua vez de jogar. Este comando não altera o estado do jogo. As linhas com comandos-estado têm a seguinte forma (com um espaço a separar as duas componentes):

```
status corJogador←
```

onde:

- *corJogador* é uma sequência não vazia de caracteres.

O programa escreve uma linha na consola, distinguindo quatro casos:

- Se *corJogador* não for a cor de um dos jogadores, a linha tem:

```
Nonexistent player←
```

- Se *corJogador* for a cor de um jogador e o **jogo já tiver terminado**, a linha tem:

```
The game is over←
```

- Se *corJogador* for a cor de um jogador, o jogo ainda não tiver terminado e o **jogador referido no comando puder lançar os dados**, quando e se chegar a sua vez de jogar, a linha tem a seguinte forma:

```
corJogador can roll the dice←
```

- Nos **restantes casos**, a linha tem a seguinte forma:

```
corJogador cannot roll the dice←
```

Comando-Lançamento

O comando-lançamento indica que, no momento corrente do jogo, o jogador que tem direito a lançar os dados lançou os dados e quantas pintas saíram em cada dado. As linhas com comandos-lançamento têm a seguinte forma (com um espaço a separar componentes consecutivas):

```
dice pintasDado1 pintasDado2←
```

onde:

- *pintasDado*₁ e *pintasDado*₂ são números inteiros.

O programa deve usar esta informação para atualizar o estado do jogo, mas não deve escrever qualquer resultado, exceto nos dois casos seguintes, nos quais o estado do jogo não muda e o programa escreve uma linha:

- Se *pintasDado*₁ ou *pintasDado*₂ não for um número entre 1 e 6, a linha tem:

```
Invalid dice←
```

- Se *pintasDado*₁ e *pintasDado*₂ forem números entre 1 e 6, mas o jogo já tiver terminado, a linha tem:

```
The game is over←
```

Comando-Sair

O comando-sair indica que se pretende terminar a execução do programa. A linha com o comando-sair tem:

```
exit←
```

O programa termina, escrevendo uma linha na consola. Distinguem-se dois casos:

- Se o jogo ainda não tiver terminado, a linha tem:

```
The game was not over yet...←
```

- Se o jogo já tiver terminado, a linha tem a seguinte forma, onde *corVencedor* denota a cor do jogador que ganhou o jogo:

```
corVencedor won the game!←
```

Comandos Inválidos

Sempre que o utilizador escrever uma linha que não comece com as palavras “player”, “square”, “status”, “dice” ou “exit”, o estado do jogo não deve ser alterado e o programa deve escrever uma linha com:

```
Invalid command←
```

4 Exemplos

Apresentam-se alguns exemplos. A coluna da esquerda ilustra a interação: o input está escrito a azul e o output a preto. Todas as linhas do input e do output terminam com o símbolo de mudança de linha, que se omitiu para aumentar a legibilidade. A coluna da direita tem informação para o leitor do enunciado, servindo apenas para relembrar as regras descritas anteriormente. Nessa coluna, “C” abrevia “fica na casa” e “P” abrevia “fica com a pena”.

Exemplo 1

```
RGB          Cores dos jogadores: R, G e B (que jogam por esta ordem).
25           O tabuleiro tem 25 casas.
2             Há 2 casas com multa.
7 17          Multas nas casas 7 e 17.
3             Há 3 casas com precipício.
6 20 24        Precipícios nas casas 6, 20 e 24.

player
Next to play: R
square B
B is on square 1
dice 1 6       R lança os dados; C 8; P 0.
dice 6 3       G lança os dados; C 10; P 0.

player
Next to play: B
square G
G is on square 10
status R
R can roll the dice
dice 3 6       B lança os dados; C 10; P 0.
dice 1 1       R lança os dados; C 10; P 0.
dice 6 6       G lança os dados; C 22; P 0.
dice 4 5       B lança os dados; C 19; P 0.
dice 4 3       R lança os dados; C 17; P 2. Casa 17 tem multa.

square G
G is on square 22
square R
R is on square 17
square B
B is on square 19
status B
B can roll the dice
status R
R cannot roll the dice
dice 5 5       G lança os dados; C 25; P 0. O jogo termina.
status B

The game is over
square G
G is on square 25
exit
G won the game!
```

Exemplo 2

```
WYM          Cores dos jogadores: W, Y e M (que jogam por esta ordem).
25           O tabuleiro tem 25 casas.
2             Há 2 casas com multa.
7 17          Multas nas casas 7 e 17.
3             Há 3 casas com precipício.
6 20 24        Precipícios nas casas 6, 20 e 24.

square R

Nonexistent player

dice 2 4      W lança os dados; C 7; P 2. Casa 7 tem multa.
dice 5 3      Y lança os dados; C 18; P 0. Casa 9 tem pássaro.

eXiT

Invalid command

dice 1 1      M lança os dados; C 3; P 0. W joga; P 1.

casa red

Invalid command

player

Next to play: Y

status W

W cannot roll the dice

dice 1 1      Y lança os dados; C 16; P 0. Casa 20 tem precipício.

dice 1 7

Invalid dice

dice 2 1      M lança os dados; C 1; P 0. Casa 6 tem precipício. W joga; P 0.

square W

W is on square 7

player

Next to play: Y

status W

W can roll the dice

dice 4 4      W lançará os dados quando voltar a ser a sua vez de jogar.

dice 2 3      Y lança os dados; C 8; P 0. Casa 24 tem precipício.

square Y      M lança os dados; C 1; P 0. Casa 6 tem precipício.

Y is on square 8

square M

M is on square 1

dice 5 6      W lança os dados; C 25; P 0. Casa 18 tem pássaro. O jogo termina.

dice 0 4

Invalid dice

dice 1 4

The game is over

square Y

Y is on square 8

status Y

The game is over

status marca gira!

Nonexistent player

exit

W won the game!
```

Exemplo 3

```
XYZ  
18  
2  
5 10  
1  
11  
dice 2 2  
dice 2 3  
dice 6 3  
status X  
X cannot roll the dice  
status Z  
Z cannot roll the dice  
player  
Next to play: Y  
dice 1 1  
status X  
X can roll the dice  
status Z  
Z cannot roll the dice  
player  
Next to play: Y  
dice 2 1  
status X  
X can roll the dice  
status Y  
Y can roll the dice  
status Z  
Z can roll the dice  
player  
Next to play: X  
dice 3 2  
square X  
X is on square 10  
status X  
X cannot roll the dice  
status Y  
Y can roll the dice  
status Z  
Z can roll the dice  
square Y  
Y is on square 5  
square Z  
Z is on square 10  
player  
Next to play: Y  
exit  
The game was not over yet...
```

Cores dos jogadores: X, Y e Z (que jogam por esta ordem).
O tabuleiro tem 18 casas.
Há 2 casas com multa.
Multas nas casas 5 e 10.
Há 1 casa com precipício.
Precipício na casa 11.
X lança os dados; **C** 5; **P** 2. Casa 5 tem multa.
Y lança os dados; **C** 6; **P** 0.
Z lança os dados; **C** 10; **P** 2. Casa 10 tem multa. X joga; **P** 1.

Y lança os dados; **C** 8; **P** 0. Z joga; **P** 1. X joga; **P** 0.

X lançará os dados quando voltar a ser a sua vez de jogar.

Y lança os dados; **C** 5; **P** 0. Casa 11 tem precipício. Z joga; **P** 0.

X lança os dados; **C** 10; **P** 2. Casa 10 tem multa.

5 Entrega do Trabalho

O trabalho é entregue no [Mooshak](#). Deverá submeter um arquivo `.zip` ao Problema A do concurso **IP2223-P1**. Não se esqueça que:

- O arquivo deve conter apenas todos os ficheiros `.java` que tiver criado para resolver o problema.
- O arquivo tem necessariamente de conter um ficheiro `Main.java`, onde está o método `main`.
- A classe `Main` tem de estar na raiz do arquivo (tem de pertencer ao pacote principal (`default`)).
- A versão de Java instalada no Mooshak é a 8.¹

Cada aluno tem de se registar no concurso IP2223-P1, de acordo com as seguintes regras:

- O **nome do utilizador** (no Mooshak) tem de ser o **número do aluno**.
- O **grupo** (no Mooshak) é o do respetivo **turno das aulas práticas**.
- O **endereço de email** (para o qual o Mooshak envia a *password*) tem de ser o endereço institucional.

Por exemplo, o aluno com o número 98765, inscrito no turno P3, é o utilizador com o nome 98765 e pertence ao grupo P3. Só serão considerados entregues (e avaliados) os programas dos utilizadores no concurso IP2223-P1 que respeitem estas regras.

O concurso IP2223-P1 abre no dia 27 de outubro e encerra às **20h00** do dia **7 de novembro de 2022** (segunda-feira). Pode ressubmeter o trabalho as vezes que entender, até à hora limite de submissão. Apenas será avaliado o programa que obtiver a **maior pontuação** no Mooshak; se houver vários programas com a maior pontuação, será avaliado, de entre esses, o **último** que tiver sido submetido. Se quiser que o programa avaliado seja outro, tem de enviar uma mensagem ao Professor Artur Miguel Dias (amd@fct.unl.pt) até uma hora após o concurso fechar, indicando o número da submissão que pretende que seja avaliada.

6 Critérios de Avaliação do Trabalho

De acordo com o [Regulamento de Avaliação de Conhecimentos da FCT NOVA](#):

- Existe fraude quando:
 - (a) Se utiliza ou tenta utilizar, sob qualquer forma, num teste, exame, ou outra forma de avaliação, presencial ou a distância, informação ou equipamento não autorizado;
 - (b) Se presta ou recebe colaboração não autorizada na realização dos exames, testes, ou qualquer outra prova de avaliação de conhecimentos individuais;
 - (c) Se presta ou recebe colaboração, não permitida pelas regras aplicáveis a cada caso, na realização de trabalhos práticos, relatórios ou outros elementos de avaliação.
- Os estudantes diretamente envolvidos numa fraude são liminarmente reprovados na disciplina, sem prejuízo de eventual procedimento disciplinar ou cível.

¹Esta informação é irrelevante se só usar as instruções de Java dadas nas aulas porque, nesse caso, o programa deverá ter o mesmo comportamento na sua máquina e no Mooshak.

Em IP, o documento [Colaboração Permitida e Não Permitida](#), disponibilizado no Moodle, clarifica as alíneas transcritas acima. Os alunos que cometerem fraude num trabalho não obterão frequência.

A avaliação do trabalho tem duas componentes independentes, cujas notas se somam para obter a nota do trabalho:

- **Funcionalidade** (correção dos resultados produzidos): **12 valores**

Um programa submetido ao concurso que só use as classes da biblioteca permitidas e que obtenha P pontos no Mooshak terá $P/10$ valores.²

As classes da biblioteca permitidas são **apenas as que foram usadas nas aulas teórico-práticas**.

- **Qualidade do código: 8 valores³**

Um código com qualidade tem, entre outras, as seguintes características:

- **Várias classes que caracterizem bem as diferentes entidades do problema;**
- Classes, métodos, variáveis e constantes com **objetivos bem definidos e as restrições de acesso apropriadas**;
- Algoritmos simples e bem estruturados, implementados com as instruções mais adequadas;
- Identificadores que expressem os conceitos que representam, escritos de acordo com as convenções ensinadas (por exemplo, o nome de uma classe deve ser um substantivo que comece com uma letra maiúscula);
- **Precondições nos métodos públicos;**
- Indentação correta (lembre-se do comando CTRL-SHIFT-F do Eclipse), linhas com 100 caracteres (no máximo)⁴ e métodos com 25 linhas (no máximo);
- Um comentário no início de cada classe, que indique o que os objetos da classe representam, e um comentário antes de cada método, que explique resumidamente o que o método faz.

Bom trabalho!

²O trabalho pode ser realizado incrementalmente. Por exemplo, pode começar por assumir que nenhum jogador para numa casa com multa, com precipício ou com ave. É claro que, enquanto o programa não produzir os resultados corretos em todos os testes do Mooshak, não obterá 120 pontos.

³Note que a qualidade do código tem um peso muito grande na nota do trabalho.

⁴Na contagem do número de caracteres de uma linha de código, considere que um tab equivale a 4 caracteres.