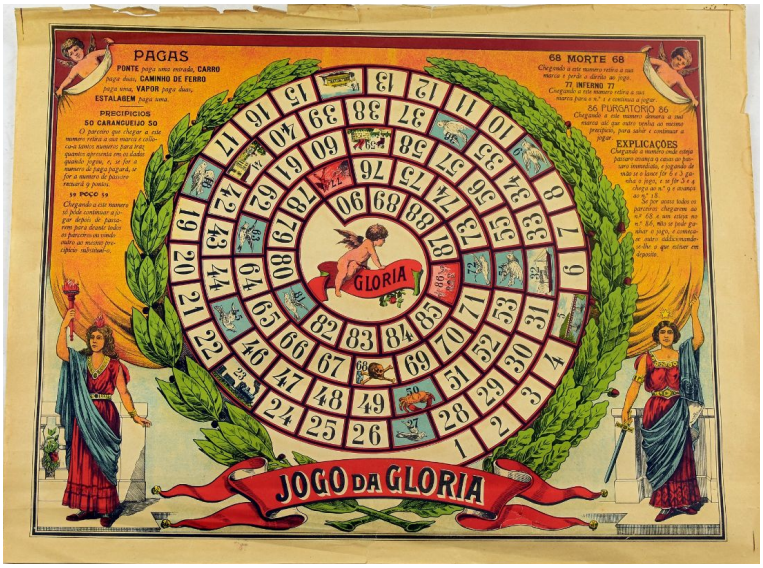
NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Departamento de Informática

Introdução à Programação

The Goose Cup (Torneio da Glória)

Second Programming Project



Academic Year 2022/23

Version 1 (2022/11/28)

Preamble

The programming project is developed in groups of **two students from the same lab classes shift**. A programming project carried out by two students from different lab classes shifts will only be accepted with prior authorization, being necessary to present a good justification for the authorization to be granted.

The programming project is submitted to Mooshak, which accepts submissions until **20:00** on **December 10, 2022**. Read this document with the utmost attention, to understand the problem very well and all the details about the Mooshak contest and the assessment criteria for the programming project.

1 Concepts and Goal of the Programming Project

After so many years of playing the Goose Game (*Jogo da Glória*), it is time to have tournaments, called Goose Cups. In a *cup* with J players, $J - 1$ *games* are played in sequence. All players participate in the first game but, as one player *is eliminated* during the course or at the end of each game, the number of *active* players decreases. The winner of the last game wins the cup.

Figure 1: Board of the Game

All cup games use the same *board*. The traditional board, shown in Figure 1, is composed of 90 *squares*, numbered from 1 to 90. Each player has a coloured *piece* (as those illustrated in Figure 2), which (in each game) starts on square 1. When it is his turn to play, the player rolls two dice to determine how many squares the piece advances. The game is won by whoever arrives at the 90th square first or who, when rolling the dice for the first time in the game, obtains 6 dots on one dice and 3 dots on the other.

For more excitement, there are special squares, called *fines*, *cliffs* and *geese*. If the piece stops on a fine, the player cannot roll the dice for a number of turns. The penalty suffered when the piece stops on a cliff varies greatly with the cliff. For example, the *crab* makes the piece move backwards (instead of moving forwards), the *hell* forces the mark to return to square 1, and the *death* expels the player (preventing him from continuing to play). Unlike fines and cliffs, stopping at a square with a goose is fortunate, because the piece immediately advances 9 more squares.

Figure 2: Pieces

The *ranking* of players is determined by the time they are eliminated (the first player eliminated is the last ranked). While players are active, their performance essentially depends on the number of games won.

The goal of this programming project is to program in Java a very simplified version of the Goose Cup. With the information about the number of players and the colours of their pieces, the board, and the sequence of rolls of the dice, the program must be able to indicate, at any moment of the cup, the ranking of players, on which square each active player's piece is and, if the cup is not over yet, who is the next player to roll the dice and whether or not a given (active) player can roll them when it is his turn to play.

2 Cup Rules

Three to ten players compete against each other, each with a different piece. For simplicity, every player is identified by the colour of his piece and is said to be on a square if his piece is on that square. Players play one at a time. The order in which players play is cyclic and is defined before the cup starts.

For example, if the players have pieces with the colours A, B, C and D (*amber*, *blue*, *charcoal* and *dark chocolate*) and this is the order in which they play: the first to play is the player (who has the piece) A; after A plays, it is B's turn to play; after B plays, it is C's turn; after C plays, it is D's turn; after D plays, it is again A's turn to play.

The cup comprises $J - 1$ games, if J is the number of players. At the beginning of the cup, all players are *active* (no player has been eliminated), so all players participate in the first game. But, during the course or at the end of each game, one of the players is *eliminated* from the cup (never plays again). Only the players who have never been eliminated remain active and participate in the next game (if any). The game in which only two players participate is the last one. The cup ends when the last game ends: the winner of that game wins the cup and the other player is eliminated.

Eliminating a player does not change the order in which the other players play. If the starting order was A, B, C and D (as in the example above) and B was eliminated at the end of the first game, the second game would be played by A, C and D, who would play in that order. The first player to play would be A, then it would be C, then it would be D, then it would be A, etc. When a game starts, the first player to play is always the first in the previously defined order that is still active.

The board (used in all games of the cup) has a number C of squares, numbered from 1 to C (with $10 \leq C \leq 150$). Any square has one, and only one, type. That type can be *normal*, *goose*, *fine*, or *cliff*, existing three variants of cliffs: *crab*, *hell* and *death*. Each fine has an associated penalty (an integer between 1 and 4), which defines how many turns a player who stops at the square is prevented from rolling the dice. Geese, fines and cliffs can only occur between squares 2 and $C - 1$. In this interval, all "multiple of 9" squares (squares 9, 18, 27, etc., without exceeding $C - 1$) have a goose.

When a player is on a square c, rolls the two dice and p spots come out, the player *advances p squares* (where $1 \leq c < C$ and $2 \leq p \leq 12$). This means that the player goes to square $c + p$, unless that square does not exist (because it is larger than C), in which case the player moves to square C. Similarly, a player who is on a square c and who *moves back p squares* goes to square $c - p$, unless that square does not exist (because it is smaller than 1); in that case, the player goes to square 1.

At the beginning of each game, all (active) players are on square 1 and have a penalty of zero moves. When it is his turn to play, the player's behaviour depends on the value of his penalty. The move can change the square where he is, the penalty he has and his state (which could become eliminated). In the description that follows, if it is not stated that the value of the penalty changes, the player's penalty does not change and, if it is not stated that the player is eliminated, the player remains active.

- If the penalty value is positive, the player cannot roll the dice (having to pass them to the next player). The player remains on the same square and the value of his penalty drops by one. **These moves** are considered to **occur as soon as possible**.
- If the penalty value is zero, the player rolls the dice. If it is the player's first move in the game and there are 6 dots on one of the dice and 3 dots on the other, the player's piece goes to square C. In the remaining cases, his piece advances as many squares as the total number of spots that came out and, in the five following situations, the corresponding operations are performed:
  - If the piece stops on a goose, the player advances 9 (more) squares.
  - If the piece stops on a fine, the player's penalty increases between 1 and 4 units, depending on the value that is specified on the board for that square.

- If the piece stops on a crab, the player returns to the square where he was (before he rolled the dice) and then moves back as many squares as the total number of spots that came out.
- If the piece stops on a hell, the player goes to square 1.
- If the piece stops on a death and it is the first time in the game that a piece stops on a death, the player is immediately eliminated from the cup.[1]

As a general rule, when a player is eliminated for having stopped on a death, the game continues with the other players (almost as if the elimination had not taken place). Of course, the eliminated player never plays again (neither in the game in which he was participating, nor in the cup). However, if only one player remains active, the game is over, because that player is definitely the winner of the game (and the winner of the cup). In all other cases, a game only ends when one of the players reaches square C. That player wins the game. At that moment, if no player has been eliminated during the game, one of the players is eliminated and the next game starts with one less participant, except if there is only one active player, in which case the cup ends. The elimination of the player, if any, and the start of the next game or the end of the cup occur immediately after the end of the game. When the cup is over, the winner can no longer play (and his piece remains on the square where it was when the last game ended).

If no one has been eliminated during the game, the player eliminated at the end of the game is the one who is on the smallest square (farthest from square C) at the time the game ends; if there are several players on that square, the eliminated player is, among them, the last one to play in the previously defined order.

When the cup ends, the ranking sorts the players in the reverse order of elimination. The last classified is the first eliminated player (the one who was eliminated during or at the end of the first game), the penultimate classified is the second eliminated player, and so on. The first place is for the winner of the cup, who is never eliminated. While the cup is running, there are several active players who occur, in the classification, above the players already eliminated. An active player i is better ranked than another active player j, if i has won more games than j; if both have won the same number of games, if i is on a square greater than j's square (i.e., if i is closer to square C); if both have won the same number of games and are on the same square, if i plays before j in the previously defined order.

3 System Specification

The application's interface is intended to be simple, so that it can be used in different environments and allow automating the testing process. For these reasons, the input and the output must respect the precise format specified in this section. You can assume that the input obeys the indicated value and format restrictions, that is, that the user does not make mistakes beyond those foreseen in this document.

The program reads lines from the standard input (`System.in`) and from the text file named `boards.txt`, stored in the current directory, writes lines to the standard output (`System.out`) and is case sensitive (for example, the words "exit" and "Exit" are different).

Input Structure

The `boards.txt` file has the following structure (where k denotes the positive number of boards in the file and the symbol $\hookleftarrow$ represents the newline character):

[1] If the piece stops on a death and some piece has already stopped on a death during the game, it is as if the player had stopped on a normal square (between 1 and $C-1$).

$C_1 \hookleftarrow$
$m_1 \hookleftarrow$
$\mathit{fine}_1\ \mathit{penalty}_1 \hookleftarrow$
$\mathit{fine}_2\ \mathit{penalty}_2 \hookleftarrow$
.....
$\mathit{fine}_{m_1}\ \mathit{penalty}_{m_1} \hookleftarrow$
$p_1 \hookleftarrow$
$\mathit{cliff}_1\ \mathit{type}_1 \hookleftarrow$
$\mathit{cliff}_2\ \mathit{type}_2 \hookleftarrow$
.....
$\mathit{cliff}_{p_1}\ \mathit{type}_{p_1} \hookleftarrow$
$C_2 \hookleftarrow$
$m_2 \hookleftarrow$
$\mathit{fine}_1\ \mathit{penalty}_1 \hookleftarrow$
$\mathit{fine}_2\ \mathit{penalty}_2 \hookleftarrow$
.....
$\mathit{fine}_{m_2}\ \mathit{penalty}_{m_2} \hookleftarrow$
$p_2 \hookleftarrow$
$\mathit{cliff}_1\ \mathit{type}_1 \hookleftarrow$
$\mathit{cliff}_2\ \mathit{type}_2 \hookleftarrow$
.....
$\mathit{cliff}_{p_2}\ \mathit{type}_{p_2} \hookleftarrow$
.....
$C_k \hookleftarrow$
$m_k \hookleftarrow$
$\mathit{fine}_1\ \mathit{penalty}_1 \hookleftarrow$
$\mathit{fine}_2\ \mathit{penalty}_2 \hookleftarrow$
.....
$\mathit{fine}_{m_k}\ \mathit{penalty}_{m_k} \hookleftarrow$
$p_k \hookleftarrow$
$\mathit{cliff}_1\ \mathit{type}_1 \hookleftarrow$
$\mathit{cliff}_2\ \mathit{type}_2 \hookleftarrow$
.....
$\mathit{cliff}_{p_k}\ \mathit{type}_{p_k} \hookleftarrow$

where, for every board i (with $i = 1, 2, \ldots, k$):

- C_i is an integer between 10 and 150;
- m_i and p_i are integers between 1 and one third of C_i;
- fine_1, ..., fine_{m_i} are m_i integers in ascending order;
- $\mathit{penalty}_1$, ..., $\mathit{penalty}_{m_i}$ are m_i integers between 1 and 4;
- cliff_1, ..., cliff_{p_i} are p_i integers in ascending order;
- type_1, ..., type_{p_i} are p_i words, each one being "`crab`", "`death`" or "`hell`";
- the numbers fine_1, ..., fine_{m_i}, cliff_1, ..., cliff_{p_i} vary between 2 and $C_i - 1$, they are all distinct and none is a multiple of 9.

The description of a board starts with the number of squares on the board (C_i). The second line has the number of fines (m_i). m_i lines follow, each with a square where there is a fine and the corresponding penalty, separated by a space. The next line has the number of cliffs (p_i). In each of the remaining p_i lines there is information about a cliff: the square where it occurs and its type, separated by a space.

There are no blank lines separating the boards.

The order in which the boards occur in the `boards.txt` file will be used to identify (in the standard input) the board used in the cup: the first board in the file is identified by the number one, the second board in the file is identified by the number two, and so on.

The standard input has the following structure:

$\mathit{colours} \hookleftarrow$
$\mathit{boardNr} \hookleftarrow$
$\mathit{command} \hookleftarrow$
$\mathit{command} \hookleftarrow$
.....
$\mathit{command} \hookleftarrow$
`exit`$\hookleftarrow$

where:

- $\mathit{colours}$ is a sequence of different capital letters (from 'A' to 'Z'), of length between 3 and 10;

- *boardNr* is a positive integer that does not exceed the number of boards in the `boards.txt` file;
- *command* is one of five commands (called *player-command*, *square-command*, *status-command*, *ranking-command*, and *dice-command*) or an invalid command, explained below.

The first line of the standard input specifies how many players participate in the cup, what are their colours and the order in which they play. The second line identifies the board used in the cup, through the order number of that board in the `boards.txt` file. An arbitrary number of commands follow. The last line has a special command, the *exit-command*, which can only occur on the last line because it terminates the execution of the program.

Player-Command

The player-command indicates that we want to know who is the next player to roll the dice, at the current moment of the cup. This command does not change the cup state. Lines with player-commands have:

`player`↩

The program writes a line, distinguishing two cases:

- If the cup is already over, the line has:

  `The cup is over`↩

- In the remaining cases, the line has the following form, where *playerColour* stands for the colour of the next player to roll the dice:

  `Next to play:` *playerColour*↩

Square-Command

The square-command indicates that we want to know on which square the player with the given colour is, at the current moment of the cup. This command does not change the cup state. Lines with square-commands have the following form (with a single space separating the two components):

`square` *playerColour*↩

where:

- *playerColour* is a non-empty sequence of characters.

The program writes a line, distinguishing three cases:

- If *playerColour* is not the colour of one of the players, the line has:

  `Nonexistent player`↩

- If *playerColour* is the colour of an eliminated player, the line has:

  `Eliminated player`↩

- In the remaining cases, the line has the following form, where *playerSquare* stands for the square where the player referred to in the command is:

  playerColour `is on square` *playerSquare*↩

Status-Command

The status-command indicates that we want to know whether, at the current moment of the cup, the player with the given colour can or cannot throw the dice, when (and if) it is his turn to play. This command does not change the cup state. Lines with status-commands have the following form (with a single space separating the two components):

`status` *playerColour*↩

where:

- *playerColour* is a non-empty sequence of characters.

The program writes a line, distinguishing five cases:

- If *playerColour* is not the colour of one of the players, the line has:

  `Nonexistent player`↩

- If *playerColour* is a player's colour and the cup is already over, the line has:

  `The cup is over`↩

- If the cup is not over yet and *playerColour* is the colour of an eliminated player, the line has:

  `Eliminated player`↩

- If the cup is not over yet, *playerColour* is the colour of an active player, and that player can roll the dice, when and if it is his turn to play, the line has the following form:

  playerColour `can roll the dice`↩

- In the remaining cases, the line has the following form:

  playerColour `cannot roll the dice`↩

Ranking-Command

The ranking-command indicates that we want to know the ranking of players, at the current moment of the cup. This command does not change the cup state. Lines with ranking-commands have:

`ranking`↩

The program writes as many lines as the number of players, each one with the information of a different player. Consider that *playerColour* and *nrGamesWon* denote, respectively, the colour and the number of games won by a player. If the player is active, *playerSquare* stands for the square where he is. The form of the line depends on the player's state at the current moment of the cup.

- If the player is active, the line has the following form:

  playerColour`:` *nrGamesWon* `games won; on square` *playerSquare*`.`↩

- If the player has already been eliminated, the line has the following form:

  playerColour`:` *nrGamesWon* `games won; eliminated.`↩

Active players occur before eliminated players. The lines concerning active players are written in descending order of number of games won; in case of a tie in the number of games won, in descending order of square; in case of a tie in the number of games won and in the square, in the order in which they play. The lines concerning the eliminated players are written in reverse order of elimination.

Dice-Command

The dice-command indicates that, at the current moment of the cup, the player who has the right to roll the dice has rolled the dice and how many spots have come out on each dice. Lines with dice-commands have the following form (with a single space separating consecutive components):

`dice` *diceSpots*$_1$ *diceSpots*$_2$↩

where:

- *diceSpots*$_1$ and *diceSpots*$_2$ are integer numbers.

The program must use this information to update the cup state, but it must not write any results, except in the following two cases, in which the cup state does not change and the program writes a line:

- If *diceSpots*$_1$ or *diceSpots*$_2$ is not a number between 1 and 6, the line has:

  `Invalid dice`↩

- If *diceSpots*$_1$ and *diceSpots*$_2$ are numbers between 1 and 6, but the cup is already over, the line has:

  `The cup is over`↩

Exit-Command

The exit-command indicates that we want to terminate the execution of the program. The line with the exit-command has:

`exit`↩

The program ends, writing a line. Two cases are distinguished:

- If the cup is not over yet, the line has:

  `The cup was not over yet...`↩

- If the cup is already over, the line has the following form, where *winnerColour* stands for the colour of the player who won the cup:

  winnerColour `won the cup!`↩

Invalid Commands

Whenever the user writes a line that does not start with the words "player", "square", "status", "ranking", "dice" or "exit", the cup state must not be changed and the program must write a line with:

`Invalid command`↩

4 Examples

Some examples are presented, which assume that the `boards.txt` file has the contents shown in Figure 3.[2] The left column illustrates the interaction: the input is written in blue and the output in black. All input and output lines end with a newline character, which is omitted for the sake of readability. The right column has information for the reader, serving only to remind you of the rules described above. In this column, "rolls" abbreviates "rolls the dice", "**S**" abbreviates "the player stops on square" and "**P**" abbreviates "his penalty is".

Notice that the set of examples is very incomplete: there are many possible situations that are not illustrated.

```
18
2
5 2
10 1
4
11 death
13 crab
15 hell
17 crab
25
3
7 1
17 4
19 2
3
6 hell
20 hell
24 death
33
1
11 3
5
4 death
6 death
13 crab
14 crab
19 hell
```

Figure 3: Contents of the `boards.txt` file

[2] In Mooshak, the contents of the `boards.txt` file is different.

Example 1

```
RGB                                3 players: R, G and B (who play in this order).
1                                  First board: 18 squares, 2 fines and 4 cliffs.
dice 2 2                           R rolls; S 5; P 2 (fine of 2 at 5).
dice 3 2                           G rolls; S 6; P 0.
dice 5 4                           B rolls; S 10; P 1 (fine of 1 at 10). R plays; P 1.
ranking
B: 0 games won; on square 10.
G: 0 games won; on square 6.
R: 0 games won; on square 5.
status R
R cannot roll the dice
status G
G can roll the dice
status B
B cannot roll the dice
player
Next to play: G
square RED
Nonexistent player
dice 0 6
Invalid dice
dice 3 6                           G rolls; S 1 (hell at 15); P 0. B plays; P 0. R plays; P 0.
square G
G is on square 1
status R
R can roll the dice
status B
B can roll the dice
player
Next to play: G
ranking
B: 0 games won; on square 10.
R: 0 games won; on square 5.
G: 0 games won; on square 1.
dice 6 2                           G rolls; S 18 (goose at 9). 1st game ends: G wins; R eliminated.
ranking
G: 1 games won; on square 1.
B: 0 games won; on square 1.
R: 0 games won; eliminated.
dice 1 1                           G rolls; S 3; P 0.
dice 3 1                           B rolls; S 5; P 2 (fine of 2 at 5).
dice 1 3                           G rolls; S 7; P 0. B plays; P 1.
player
Next to play: G
dice 6 1                           G rolls; S 14; P 0. B plays; P 0.
ranking
G: 1 games won; on square 14.
B: 0 games won; on square 5.
R: 0 games won; eliminated.
```

```
player
Next to play: G
rank
Invalid command
dice 2 1                        G rolls; S 11 (crab at 17); P 0.
dice 5 6                        B rolls; S 16; P 0.
ranking
G: 1 games won; on square 11.
B: 0 games won; on square 16.
R: 0 games won; eliminated.
dice 3 3                        G rolls; S 5 (crab at 17); P 0.
square G
G is on square 5
square R
Eliminated player
status R
Eliminated player
dice 2 2                        B rolls; S 18. 2nd game and cup end: B wins; G eliminated.
ranking
B: 1 games won; on square 18.
G: 1 games won; eliminated.
R: 0 games won; eliminated.
square R
Eliminated player
square B
B is on square 18
square G
Eliminated player
square Blue sky
Nonexistent player
player
The cup is over
status B
The cup is over
status G
The cup is over
status dark chocolate
Nonexistent player
dice -1 -1
Invalid dice
Dice 7
Invalid command
dice 1 1
The cup is over
1000
Invalid command
exit
B won the cup!
```

Example 2

`ABCDE`	5 players: A, B, C, D and E (who play in this order).
`3`	Third board: 33 squares, 1 fine and 5 cliffs.
`dice 5 6`	A rolls; **S** 12; **P** 0.
`dice 2 1`	B rolls; **S** 4 (death); is eliminated.
`ranking`	
`A: 0 games won; on square 12.`	
`C: 0 games won; on square 1.`	
`D: 0 games won; on square 1.`	
`E: 0 games won; on square 1.`	
`B: 0 games won; eliminated.`	
`square B`	
`Eliminated player`	
`status B`	
`Eliminated player`	
`dice 5 5`	C rolls; **S** 11; **P** 3 (fine of 3 at 11).
`dice 3 2`	D rolls; **S** 6 (death); **P** 0. D is still active.
`player`	
`Next to play: E`	
`dice 6 5`	E rolls; **S** 12; **P** 0.
`ranking`	
`A: 0 games won; on square 12.`	
`E: 0 games won; on square 12.`	
`C: 0 games won; on square 11.`	
`D: 0 games won; on square 6.`	
`B: 0 games won; eliminated.`	
`dice 1 6`	A rolls; **S** 1 (hell at 19); **P** 0. C plays; **P** 2.
`dice 6 6`	D rolls; **S** 27 (goose at 18); **P** 0.
`dice 1 1`	E rolls; **S** 10 (crab at 14); **P** 0.
`ranking`	
`D: 0 games won; on square 27.`	
`C: 0 games won; on square 11.`	
`E: 0 games won; on square 10.`	
`A: 0 games won; on square 1.`	
`B: 0 games won; eliminated.`	
`dice 4 2`	A rolls; **S** 7; **P** 0. C plays; **P** 1.
`square A`	
`A is on square 7`	
`status C`	
`C cannot roll the dice`	
`dice 4 2`	D rolls; **S** 33. 1st game ends: D wins.
`ranking`	
`D: 1 games won; on square 1.`	
`A: 0 games won; on square 1.`	
`C: 0 games won; on square 1.`	
`E: 0 games won; on square 1.`	
`B: 0 games won; eliminated.`	
`dice 4 3`	A rolls; **S** 8; **P** 0.
`dice 6 3`	C rolls; **S** 33 (1st move). 2nd game ends: C wins; E eliminated.

```
ranking
C: 1 games won; on square 1.
D: 1 games won; on square 1.
A: 0 games won; on square 1.
E: 0 games won; eliminated.
B: 0 games won; eliminated.
dice 1 4                        A rolls; S 6 (death); is eliminated.
player
Next to play: C
dice 2 6                        C rolls; S 18 (goose at 9); P 0.
ranking
C: 1 games won; on square 18.
D: 1 games won; on square 1.
A: 0 games won; eliminated.
E: 0 games won; eliminated.
B: 0 games won; eliminated.
dice 3 6                        D rolls; S 33 (1st move). 3rd game ends: D wins.
player
Next to play: C
ranking
D: 2 games won; on square 1.
C: 1 games won; on square 1.
A: 0 games won; eliminated.
E: 0 games won; eliminated.
B: 0 games won; eliminated.
dice 1 2                        C rolls; S 4 (death); is eliminated. 4th game ends: D wins.
player
The cup is over
ranking
D: 3 games won; on square 1.
C: 1 games won; eliminated.
A: 0 games won; eliminated.
E: 0 games won; eliminated.
B: 0 games won; eliminated.
exit
D won the cup!
```

Example 3

```
MNOABCXYZ                       9 players (who play in the specified order).
2                               Second board: 25 squares, 3 fines and 3 cliffs.
player
Next to play: M
dices 0 0
Invalid command
status X
X can roll the dice
dice 3 2                        M rolls; S 1 (hell at 6); P 0.
exit
The cup was not over yet...
```

5 Programming Project Submission

The programming project is submitted to Mooshak. You must submit a `.zip` file to Problem A of contest **IP2223-P2**. Do not forget that:

- The archive should only contain all the `.java` files that you have created to solve the problem.
- The archive must necessarily contain a `Main.java` file, where the `main` method is.
- The `Main` class must belong to the default package.
- The Java version installed on Mooshak is 8.[3]

Each team must register for the IP2223-P2 contest according to the following rules:

- The **username** (in Mooshak) must be of the form **xxxxx_yyyyy**, where **xxxxx** and **yyyyy** denote the student numbers of the team members.
- The **group** (in Mooshak) corresponds to the lab classes shift.[4]
- The **email** address (to which Mooshak sends the password) must be the institutional address of one of the team members.

For example, the team formed by students with numbers 98765 and 98789, both enrolled in shift P5, is the user with name 98765_98789 and belongs to group P5. Only programs submitted to the IP2223-P2 contest by users who respect these rules will be evaluated.

The IP2223-P2 contest opens on the 30th of November and ends at **20:00** on the **10th of December 2022** (Saturday). You can resubmit a program as many times as you like, up to the submission deadline. Only the program that obtains the highest score in Mooshak will be evaluated; if there are several programs with the highest score, the last one (of those) submitted will be evaluated. If you want the evaluated program to be different, you must send a message to Professor Artur Miguel Dias (amd@fct.unl.pt) up to one hour after the contest closes, indicating the number of the submission that you want to be evaluated.

6 Assessment Criteria

According to the Regulamento de Avaliação de Conhecimentos da FCT NOVA (FCT NOVA Knowledge Assessment Regulation):

- There is fraud when:
  (a) You use or attempt to use, in any way, in a test, exam, or other form of in-person or remote assessment, unauthorized information or equipment;
  (b) You give or receive unauthorized help in exams, tests, or any other component of the knowledge assessment;
  (c) You give or receive help, not permitted by the rules applicable to each case, in carrying out of practical assignments, reports or other assessment elements.

[3] This information is irrelevant if you only use the subset of the Java language taught in classes because, in that case, the program should have the same behaviour on your machine and on Mooshak.

[4] Students from different shifts who are authorized to do the programming project together will receive instructions on which Mooshak group they should select.

- Students directly involved in a fraud are immediately excluded from the course assessment process, without prejudice of a disciplinary or civil procedure.

The document Colaboração Permitida e Não Permitida, available in Moodle, clarifies the points transcribed above.

The evaluation of the programming project has two independent components, whose grades are added to obtain the grade of the programming project:

- **Functionality** (correction of the results produced)**: 12 points** (out of 20)

  A program submitted to the contest that only uses the allowed library classes and that gets P points (out of 120) in Mooshak will be scored P/10 points (out of 20).[5]

  The allowed library classes are **only those that were used in theoretical-practical classes.**

- **Code quality: 8 points** (out of 20)[6]

  A quality code has, among others, the following characteristics:

  - **Several classes that characterize the different entities of the problem well**;
  - Classes, methods, variables and constants with well-defined goals and appropriate access constraints;
  - Simple and well-structured algorithms, implemented with the most suitable instructions;
  - Identifiers that express the concepts they represent, written according to the conventions taught (for example, the name of a class must be a noun that starts with a capital letter);
  - Preconditions in public methods;
  - Correct indentation (remember Eclipse's CTRL-SHIFT-F command), lines with 100 characters (maximum)[7] and methods with 25 lines (maximum);
  - A comment at the beginning of each class, which indicates what the objects of the class represent, and a comment before each method, which briefly explains what the method does.

The **discussions** of the programming projects are mandatory and will take place on the **12th, 13th and 15th of December 2022**, during classes (theoretical-practical or lab). You will receive more information by December 11th.

According to the assessment rules, written in CLIP since the beginning of the semester, the grade of each team member depends on the grade of the programming project and the individual performance in the discussion. Consequently, the grades of the two team elements can be different.

Good luck!

[5] The program can be done incrementally. For example, you can start by assuming that no player stops on a death. Of course, as long as the program does not produce the correct results for all tests in Mooshak, it will not get 120 points.

[6] Notice that the quality of the code has a great weight in the grade of the programming project.

[7] When counting the number of characters in a line of code, consider that a tab is equivalent to 4 characters.