Robots Game (object oriented version)

GAME OVERVIEW

The objective of this practical work is to develop a program to play a robots game, similar to the game developed in the first practical work but with some different rules.

The player is placed in a maze made up of posts. The posts are of two types: electrified and non-electrified. There are also some interceptor robots that will try to destroy the player. If the player touches an electrified post or any of these robots, that is the end of the game (and the player!). The robots are also destroyed when they touch an electrified post or they collide with each other, and they become stuck when they collide with a non-electrified post.
Every time the player moves in any direction (horizontally, vertically, or diagonally) to a contiguous cell, each robot moves one cell closer to the new player's location, in whichever direction is the shortest path. The robots have no vision sensors but they have an accurate odour sensor that allows them to follow the player!
The maze has one or more exit gates. The objective of the player is to reach one of the exit gates, in the shortest time possible, before being electrocuted (by touching an electrified post) or caught by a robot.

LEARNING OBJETIVES

The main objectives of this work are that students practice the development of programs, in **C ++ language**, following the **"object oriented" programming paradigm**, selecting data structures and algorithms suitable for the implementation of the game and taking advantage of the data structures and algorithms available in the **Standard Template Library (STL)** of **C++**. Other objectives like the use of files and the development of simple and robust program interfaces are also present.

FUNCTIONAL SPECIFICATIONS

The functional specifications of the program to be developed in this 2nd practical work are similar to those of the 1st practical work, with some modifications.

Program flow

The initial menu has 4 options: 1) Rules; 2) Play; 3) Winners; 0) Exit. The corresponding actions are: 1- show the rules of the game; 2- play the game; 3- show the list of winners (or the message "empty list" if there are no winners, yet). The rules must be saved in an auxiliary text file, named "RULES.TXT".

When the user chooses to play the game, the program must:

- ask the number (an integer value) of the maze to use;
- using that number, build automatically the name of the file where the initial state of the maze is saved (the name must have the format MAZE_XX.TXT, where XX represent the number of the maze, with 2 digits; *see below*) and verify that the file exists;
- the number must be asked repeatedly until either the corresponding maze file is found or the user input is 0 (zero); in this last case, the initial menu must be shown again.

For playing, the maze and the initial position of the player and the robots must be loaded from that file. The symbols used in the file are (*see example in next pages*): **H** - player; **R** - robot; ***** - electrified post; **+** - non-electrified post; **O** - exit gate of the maze. As the maze is loaded, each robot must be assigned a sequential identification number (to be stored in the internal data structures of the program), starting with 1.

After that, the program must repeat the following actions, until either the player or all the robots are destroyed or the player manages to exit the maze, through one of the gates:

- Display the maze. The symbols used for the elements of the maze are the same used in the maze file, plus two additional symbols, **h** and **r**, as described below:
  - ***** = electrified post; **+** = non-electrified post; **O** = exit door;
  - **H** = player (alive); **h** = player (dead); the player dies when he/she touches an electrified post, or is captured by a robot;
  - **R** = robot (alive); **r** = robot (destroyed=dead/stuck); a dead robot is one that collided with an electrified post; a stuck robot is one that collided with another robot (alive or destroyed) or with a non-electrified post (*see below*).
- Ask the player to indicate the movement he/she wants to do.
  - The player can only move to one of the 8 neighbour cells of his/her current cell.
  - The movement is indicated by typing one of the letters indicated below (the position of each letter relatively to the player's position indicates the movement that the player wants to do):

Q	W	E
A	player's position	D
Z	X	C

  - The player has the option to stay in his/her current position by typing S.
  - The above mentioned letters may be typed in uppercase or lowercase. If the user inputs an invalid letter/symbol, the input must be repeated.
  - The player should not be allowed to move to cells occupied by destroyed/stuck robots or by non-electrified posts; if he/she tries to do so, he/she must be informed that the movement is invalid and asked for a new movement.
  - If the player tries to move to a cell occupied by an electrified post, he/she stays in his/her current cell but dies (being represented by a **h**) because he/she touched the post.
  - The player can exit the game at any moment by typing CTRL-Z, in Windows, or CTRL-D, in Linux.
- Move the player to the new position and, if necessary, update his/her state.
- If the player is still alive, move one robot after the other, according to their id number.
  - Each robot can move to one of the 8 neighbour cells of its current cell, as the player.
  - During this step, it may happen that two or more robots move to the same cell, colliding with each other. When several robots collide, they get stuck and they are all represented by a single symbol, an **r**.
  - When a robot collides with other destroyed robots (**r** cells) or with a non-electrified post (**+** cells) it also gets stuck; in this last case, the robot occupies the position of the non-electrified post; note: in both cases, the robot(s) is(are) represented by an **r**.
  - If a robot tries to move to a cell occupied by an electrified post, it stays in its current cell but dies (being also represented by an **r)** because it touched the post.
- After each player or robot movement, detect if the game ended, because the player died or managed to exit the maze through one of the exit gates. It is considered that the player exited the maze when he/she does a movement from an interior cell of the maze to a cell that represents an exit gate; in this case, the symbol of the gate is replaced by the symbol of the player (H).
- If the player survived and managed to exit the maze through one of the gates, ask his/her name and update the list of winners, stored in the corresponding MAZE_XX_WINNERS.TXT file, where XX represent the number of the maze (*see below*). Note: the name may have more than one word but its length is limited to 15 characters.

Files names and contents

The data files used by the program must have the following names:
- Text file that contains a maze: MAZE_XX.TXT, where XX represents the number of the maze (ex: MAZE_01.TXT, …, MAZE_13.TXT, …, MAZE_99.TXT).
- Text file that contains the name of the winners and the time (in seconds) they took to exit the maze, sorted by ascending time: MAZE_XX_WINNERS.TXT (ex: MAZE_04_WINNERS.TXT)

The contents of a maze file is illustrated below; the first line contains the number of lines and columns of the maze, separated by an 'x'.

```
10 x 20
*********O**********
*   R       R       *
*    R +  *   *      *
*       H   *        *
*   **        *   *  *
*   *       *   +    *
*     + +     R  *   *
*  *              ** *
*     R       *     O
*O******************
```

The contents of a winners file is illustrated below (*see above note about the maximum length of the player's name*).

```
Player          - Time
--------------------
Sara Moura      -  512
Rui Sousa       -  513
Ana Silva       -  874
Pedro Costa     -  901
```

PROGRAM DEVELOPMENT

The program must have at least the following classes: **Post**, **Maze**, **Player**, **Robot**, **Game**. The methods of a class should not have objects of other classes as parameters, except for class **Maze**, taking into account that a **Maze** is made of **Post**'s. Class **Game** must have as attributes, a **maze**, a **player** and a data structure for representing the **robots**. The methods of class **Game** are responsible for enforcing the rules of the game and managing the interaction between the "objects" of the game (**player**, **robots** and **maze**); one of the methods must be responsible for reading the file that describes the initial state of the maze and "building" those "objects". You may add other classes, as well as other attributes to class **Game**, if you find them necessary.

The program must be developed using **separate compilation**, that is, placing the definition of each class in a **.h** or **.hpp** file and the code of the methods of each class in separate **.cpp** files.

When writing the program code, you should take into account the suggestions given in class, specially the ones concerning the following issues:
- Choice of the identifiers of types, variables and functions.
- "Magic numbers" should not be used; named constants must be used instead.
- Global variables must not be used.
- Adequate choice of function prototypes, namely, parameters passed by value or by reference and adequate use of **const** qualifiers.
- Code commenting.
- Choice of the data structures used to represent the data manipulated by the program.
- Structure of the code.
- Separation, as much as possible, of data processing from program input/output.
- Code robustness. Precautions should be taken in order to prevent the program to stop working due to incorrect input by the user, specially values outside the allowed ranges, and so on.
- Code efficiency.

Notes:
- the program to be submitted for evaluation **must not use**: a) calls to the **system()** function; b) colored output;
- anything that is not specified in this document can be decided by the members of the group and the decision must be informed in the **ReadMe.txt** file (*see next section*).

WORK SUBMISSION

- Create a folder named **Txx_Gyy**, in which **xx** represents the class number ("turma") and **yy** represents the number of the group, for example, **T01_G15**, for the group 15 of class ("turma") 1; note: both numbers must be represented using 2 digits ("01","02",…, "10","11", …).
- Copy to that folder the source code (**only** the files with extension **.cpp** and **.h** or **.hpp**) of the program.

- o <u>note</u>: the first line of each code file must contain a single line comment with the string **Txx_Gyy** as specified above (ex: **// T01_G15**)

- Copy also some examples of **MAZE_xx.TXT** and **MAZE_XX_WINNERS.TXT** files (max. 3 of each type) that you have used to test your program, and the **RULES.TXT** file.

- Note (regarding the writing of the code): the data files used by the program (namely, the 2 types mentioned above) should not have any associated directory path; open the files using just their name (ex: "MAZE_01.TXT").

- Create a file, **ReadMe.txt** (in simple text format) with the following contents (each one of the sections must be preceeded by the text in uppercase):

  Txx_Gyy (*in the first line, the class and group number, as specified above*)
  GROUP MEMBERS:
  - student_1 name
  - student_2 name
  ADDITIONAL SPECIFICATIONS:
  - (write here any additional specifications that you have used in the development of your program)
  PROGRAM DEVELOPMENT STATE:
  - (say here if all the objectives were accomplished or, otherwise, which ones were not achieved, and also what improvements were made, if any)
  MAIN DIFFICULTIES:
  - (describe here the main difficulties that you faced when developping the program)

- Compress the content of the folder **Txx_Gyy** into a file named **Txx_Gyy.zip** and <u>upload</u> that file in the FEUP Moodle's page of the course. <u>Alternative ways of delivering the work will not be accepted</u>.

- <u>Deadline</u> for submitting the work: **28/May/2021 (at 23:55h)**.