Robots Game

GAME OVERVIEW

The objective of this practical work is to develop a program to play a robots game.

The player is placed in a maze made up of high-voltage fences and posts. There are also some interceptor robots that will try to destroy the player. If the player touches the maze or any of these robots, that is the end of the game (and the player!). The robots are also destroyed when they touch the fences/posts or when they collide with each other.
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!
There is one hope: make the robots hit the maze or each other. If all of them are destroyed, the player wins.

LEARNING OBJETIVES

The development of this program will give students the opportunity to practice their skills of C/C++ programming, namely:

- use of program control structures (selection and repetition);
- use of several types of data structures (*strings*, *arrays/vectors, structs, files*);
- use of functions;
- development of simple program interfaces;
- management of invalid keyboard inputs;
- formatting outputs.
- reading from and writing to text files.

PROGRAM SPECIFICATION

The program must execute in console mode and run in any operating system, without the need of external libraries.

Program flow

The program must start by doing the following:

- Show a menu with 3 options: 1) Rules; 2) Play; 0) Exit. The action corresponding to each option is obvious.
- 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 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 must be loaded from that file. The symbols used in the file are (*see example in next page*): '**H**'-player; '**R**'-robot; '*****'-fence or post. 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:

- 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:

- ***** = electrical fence or post;
- **H** = player (alive); **h** = player (dead); the player dies when he/she collides with a fence or a post, or is captured by a robot;
- **R** = robot (alive); **r** = robot (destroyed=dead/stuck); a dead robot is one that collided with a fence or a post; a stuck robot is one that collided with another robot (alive or destroyed) (*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 robots; if he/she tries to do so, he/she must be informed that the movement is invalid and asked for a new movement.
  - 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 status (when he/she died).
- If the player is still alive, move one robot after the other, according 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) it also gets stuck.
  - If a robot collides with fences/posts it dies, being also represented by an '**r**', and the fence/post cell at the position of the collision loses its capability to electrocute.
- After each player or robot movement, detect if the game ended, either because the player died or all the robots are destroyed.
- If the player survived, 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 win, 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
**********
*  R     R *
*   R * * * *
*       H * *
*   **    * *
*   *     * *
*     * *   *
*  *        **
*     R     * *
**********
```

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

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.
- Choose the best way to pass parameters to the functions; use call-by-reference and the "const" qualifier whenever adequate.
- 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.

Suggestion: start with a simple maze, smaller that the one illustrated in previous page.

Notes:

1- the program to be submitted for evaluation **must not use**: a) calls to the system() function; b) colored output;
2- all the code can be written in a single file; it is not necessary to use separate compilation.

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**, if existing) of the program.
    - note: the first line of each code file must contain a single line comment with the string **Txx_Gyy** as specified above (ex: **// T01_G15**)
- 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*
  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 **Tx_yy.zip** and upload that file in the FEUP Moodle's page of the course. Alternative ways of delivering the work will not be accepted.
- Deadline for submitting the work: **30/April/2021 (at 23:55h)**.