FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

MIEIC – 2012/2013 PROGRAMMING – Year 1

PROJECT Nº1

INTRODUCTION

A sliding puzzle, sliding block puzzle, or sliding tile puzzle is a puzzle that challenges a player to slide flat pieces along certain routes, usually on a flat board, to establish a certain end-configuration.

The 15-puzzle (figure 1) is the oldest type of sliding puzzle. It was invented by Noyes Chapman in the 1870s. It is played on a 4-by-4 grid with 15 square tiles, labeled 1 through 15, and a blank square. The goal is to rearrange the tiles so that they are in order. It is permitted to slide tiles horizontally or vertically into the blank square.

Figure 1 – Real 15-puzzle

The puzzle also exists in other sizes, particularly the smaller 8-puzzle, played on a 3-by-3 board. Figure 2 shows a sequence of legal moves from an initial board position (left) to the goal position (right).

1	3	6
4	2	
7	5	8

1	3	
4	2	6
7	5	8

1		3
4	2	6
7	5	8

1	2	3
4		6
7	5	8

1	2	3
4	5	6
7		8

1	2	3
4	5	6
7	8	

Figure 2 - 8-puzzle: sequence of moves to solve the puzzle on the left.

In this project, students must write a program, in C++, to generate an initial, valid configuration for the puzzle and ask players to solve it, by selecting the movements of the tiles until the end-configuration is obtained.

REQUIREMENTS

The program must have two working modes:

- training mode and
- competition mode.

In both modes, the general steps of the program are the following:

- Allow the player to select the size of the puzzle.
- Generate a new puzzle by placing the tiles on the board or load a previously saved puzzle, and present it to the player.
- Repeatedly ask the player which tile to move, until the puzzle is solved or the player decides to quit.
- Register the time the player took to solve the puzzle.

TRAINING MODE

- In this mode, the player selects the size of the puzzle he/she wants to solve, the program automatically generates the puzzle and the player is asked to solve it.
- At the end of each game the player is asked for another game.

COMPETITION MODE

- In this mode, a player may enter in competition with other players for the best time to solve a given puzzle. The set of solved puzzles, along with the information of the best players who solved each puzzle, is saved in files.
- The player starts by selecting the size of the board.
- Then he/she must select one of the options: "new puzzle" or "old puzzle".
- When the player selects "new puzzle", the program generates a new puzzle and prompts the player for solving it.
- When the player selects "old puzzle" the program shows him/her a list of the puzzles of the selected size that have been previously solved and will ask which puzzle he/she wants to solve. The player may decide not to play.
- If the player suceeds in solving the puzzle, the data file associated with that puzzle has to be created/updated. If the "new puzzle" option was selected, the program creates a new file in which it saves the initial board, the name (and other information; see below) of the first player that solved the puzzle and the time he/she took to solve it. If the "old puzzle" option was selected and the time the player took to solve the puzzle is one of the best ten, the program must update the already existing file that contains the puzzle and the list of the best players.
- The rules for naming and formatting the above referred files are indicated in the following.

DATA FILES - NAME AND STRUCTURE

- In competition mode, there are two types of data files used by the program; they must be text files.
- Each puzzle solved in competition mode must be saved in a file along with the names of the best players who solved the puzzle. The name of the file has the following format: **puzzle_NxN_SSS**, where **NxN** represents the size of the board and **SSS** represents the sequence number of the board in the set of **NxN** boards. Examples: the name of the file containg the first 3x3 board that was solved must be **puzzle_3x3_001**; the name of the nineteenth 4x4 board must be **puzzle_4x4_019**.
- The contents of these files is as follows: the first lines contain the puzzle to be solved; a blank line follows; after that, the information about the best players (up to a maximum of ten) that solved the puzzle is recorded. The information to be recorded about each player is: the player's name; his/her age and sex; and the time (in seconds) he/she took to solve the puzzle. The players must be ordered according to the time they needed to solve the puzzle (from best to worst. i.e., is ascending time order). The data must be aligned, using fixed width fields, separated by one space, as illustrated in figure 3; the widths must be respectively: 20, 3, 1 and 5.

```
1234567890123456789012345678901234567890
 6  5  2  7
 1     4  3
13 14 15  8
 9 11 10 12

           Ana Sousa  15 F    71
         Pedro Silva  13 M   105
```

Figure 3 - Possible contents of the file **puzzle_4x4_019.txt**;
only two players have solved this puzzle and
the best one was Ana Sousa, who solved it in 71 seconds.

- In order to easily determine the number of boards of each size that have already been solved in competition mode, there must be a file, named **puzzle_numbers.txt**, having the following format: each line contains a board size and the number of puzzles having that size that were solved previously. These numbers can be used to generate automatically the names of the files, described in the previous point, where the "old puzzles" are saved and to determine how many puzzles the player can choose when he/she choses the "old puzzles" option. In the first run of the program, this file does not exist and so must created. The program must determine automatically if the file exists. For example, if the contents of the **puzzle_numbers.txt** file is the one illustrated in figure 4, then the next time a "new puzzle" of size 4x4 is created, it must be saved in a file named **puzzle_4x4_193**.

```
3x3 10
4x4 192
5x5 2
```

Figure 4 - Possible contents of the file **puzzle_numbers.txt**,
indicating that the number of previously solved in competition mode was
10 3x3 puzzles, 192 4x4 puzzles and 2 5x5 puzzles.

IMPLEMENTATION DETAILS AND SUGGESTIONS FOR DEVELOPMENT

- At least three board sizes must be available: 3x3 (8-puzzle), 4x4 (15-puzzle) and 5x5 (24-puzzle). The player indicates the size through a string in the format "NxN", e.g., "4x4".
- For generating a valid, new puzzle, start with the final solution and iteratively displace some tiles to obtain an initial board. In each iteration, be careful to slide into the blank square one if its horizontal or vertical neighbours (not diagonal neighbours). Be also careful to guarantee that, after the displacements, the initial board is not the final solution. At each iteration, when there are several alternatives, choose randomly a legal move.
- During playing, a tile move can be indicated just by selecting the number of one of the close horizontal/vertical neighbours of the blank tile. The direction of the movement is not necessary and the player should not be asked for it.
- Before the puzzle is solved, when prompted for the number of the tile to be moved, a player can type letter 'q'/'Q' to quit the game.
- All user inputs must be validated and the program must not crash after an invalid input value. Some examples of invalid inputs are: a letter (different from 'q'/'Q'), instead of a tile number, when selecting the move; a number that is not a neighbour of the blank tile.
- When the player selects "old puzzle" the program shows him/her, one by one, the boards of the size selected at the beginning of the game that have been previously solved by other players, along with the names of the players who solved it, as well as the time each one took to solve it, and will ask if he/she wants to solve it. This information must be presented in the format specified in figure 3.

GRADING

The grading for this project will take into account several criteria, namely:

- Code readability in general, namely, identifier names (constants, variables, types, functions,...) , indentation, separation between functions.
- Comments: function header comments and in-line comments.
- Adequate use of data structures to represent the program data.
- Code structure and modularity. The program must follow good top-down design principles.
- Correct choice of function parameters. All function parameters must be passed using the appropriate method and 'const' qualifier must be used when justifiable.
- Limiting variable scope (avoid using global variables).
- Input validation.
- The program produces all required output; the output produced is correct and is in an acceptable format.
- All the functioning requirements are met.

SUBMISSION

- Create a folder named **TxGyy** (where **x** and **yy** represent, respectively, the number of the class/*turma* and the team number (for example, T2G07, for team 7 of class 2) and copy to the folder the source code of the program (**only the files with extensions .cpp** or **.h**).
- Compact the files into a file named **TxGyy.zip** or **TxGyy.rar** and submit the file through the link available at the course page, at Moodle/FEUP.
- **Each working group must submit only one project.** The group may resubmit the project several times, but only the last submitted version will be graded and will be used to determine if the project is late.
- **Submission must be done before 2013/04/21, 23:55.**