INFO0004 – Project 2
Power-11

Pr. Laurent Mathy

March 23, 2020

You will be writing a C++ version of Power-11, a mathematical puzzle game.

1 Game rules

Power-11 is played on a square n-by-n (n equals either 4 or 5) game board which begins with an initial configuration of two tiles, each of value 2 or 4, placed at random locations on the board (see fig. 1a).

```
Score: 0
#####################
#    |    |    |    #
#    |    |    |   4#
#   2|    |    |    #
#    |    |    |    #
#####################
```

(a) Initial setup.

```
Score: 0
#####################
#    |    |    |    #
#    |    |    |    #
#    |    |    |    #
#   2|    |   2|   4#
#####################
```

(b) Move down, creation of a new tile.

```
Score: 4
#####################
#    |    |    |    #
#    |    |   2|    #
#    |    |    |    #
#   4|   4|    |    #
#####################
```

(c) Move left, merging two tiles, creation of a new tile.

```
Score: 12
#####################
#    |    |    |    #
#   2|    |    |   2#
#    |    |    |    #
#   8|    |    |    #
#####################
```

(d) Move left, *motionless* merge, creation of a new tile.

Figure 1: Example of a game.

The player then selects to move *up*, *down*, *left*, or *right*. When a direction of movement has been selected, all the tiles present on the board slide in that direction. *Sliding* means that each tile keeps moving until it either encounters a wall (edge of the board) or another tile (see fig. 1b, resulting from selecting to move *down* from fig. 1a). Note also that, after each move, a new tile with either value 2 or value 4, is placed randomly on an empty position of the board. In fig. 1b, a tile with value 2 was (randomly) introduced on the fourth row, third column.

When 2 sliding tiles with the same value encounter each other, something interesting happens: they *merge*. When 2 tiles merge, they become a single tile whose value is the sum of the merged tiles. You can see merging in action when the user selects to move *left* from the configuration in

fig. 1b, yielding the board depicted in fig. 1c (including the randomly generated tile on the second row and third column).

Merging does not disrupt sliding. At the end of a move, there is never any *hole* left on the board between tiles, in the direction of movement (before the addition of the randomly-placed tile).

Only tiles that are already present on the board at the beginning of the round can merge.

Tiles do not need to physically move to merge: if the user chooses to move left from the position in fig. 1c, the resulting board is depicted in fig. 1d (with a randomly generated tile on the second row, fourth column).

Finally, in case there are more than 2 tiles with the same value that would encounter each other, tiles are first merged in the direction of motion, *e.g.* in a *right* move, you would first merge the 2 rightmost tiles, as illustrated in fig. 2 (with a randomly-generated tile on the second row, second column).

```
Score: 0
####################
#    |    |    |    #
#    |    |    |    #
#    |    |    |    #
#   2|   2|   2|    #
####################
```

(a) Before move *right*.

```
Score: 4
####################
#    |    |    |    #
#    |   2|    |    #
#    |    |    |    #
#    |    |   2|   4#
####################
```

(b) Move right, merging rightmost tiles, creation of a new tile.

Figure 2: Merging when more than 2 tiles are involved merges first in the direction of movement.

The game finishes when either one of the following conditions arises:

1. The board is full and there is no move that would trigger a tile merge. In this case, the user looses the game.
2. There is a tile whose value is 2^{11} on the board, in which case the user wins the game.

Throughout the game, the user accrues a score: starting at 0, the value of the tile resulting from a merge is added to the score whenever a merge occurs.

2 Game interface

Your game will be played as a console application (*i.e.* the user will interact with it textually in a terminal).

2.1 Creating the game

First, the game will start with a welcome screen, giving the users four options:

e start a new Easy (5-by-5) game;

h start a new Hard (4-by-4) game;

r Resume the last saved game;

q Quit the game.

To select an entry, the user types the corresponding key followed by `Enter`. If the user enters a wrong key (or other input), just repeat the menu.

When starting a new game, you will place randomly two tiles, each with a value of 2 or or 4 (with probability 90% and 10% respectively).

The last saved game must be in a file named `power11.saved` in the directory where the game is executed. The format of this file is:

- A line containing "Score: " followed by the current score.
- The game board stored one line per row. The board is stored in a properly framed format (with '#') and with columns delimited by '|'. In other words, the game state of fig. 1c would be saved as:

```
Score: 4
####################
#    |    |    |    #
#    |    |   2|    #
#    |    |    |    #
#   4|   4|    |    #
####################
```

2.2 Playing the game

Once a game has been started, the current state of the game is displayed as follows:

- The text "Score: " followed by the current score is printed on a line;
- The board is then printed (as described above);
- The user is given the following options:

  u move Up;

  d move Down;

  l move Left;

  r move Right;

  s Save current game state to `power11.saved`;

  q Quit current game. This option must bring the user back to the welcome menu.

If the selected direction changes the state of the game[1], a randomly chosen open tile is given a value of 2 or 4, with probability 90% and 10% respectively.

As soon as the game is finished, the game state is printed with either:

- "YOU LOST!"
- "YOU WON WITH A SCORE OF ", followed by the final score.

Any input following the end of the game must bring the user back to the welcome menu.

2.3 Bonus

For a bonus of 3 points, you can implement an *undo* facility in the game. The corresponding option should be `b` (for *Back*). This undo facility must allow the user to undo any series of move, all the way back to the start of the game.

Note that to function correctly across saved games, this facility must then ensure that a saved game in the `power11.saved` file contains the series of game states encountered in the game.

[1] A selected direction of *down* in fig. 1b would not change the state of the game, and would therefore produce no effect on the game.

3 Evaluation criteria

Your code will be evaluated based on all the criteria below. Failure to comply with any of the points mentioned in **bold face** can (and most often will) result in a zero mark!

3.1 Respect the interface

Submission must scrupulously **follow the interface** defined above. Don't rename the program, change the commands, the command keys or the board format.

You **must** provide a `Makefile` that allows to build your program. Your program **must** be named `power11`. That `Makefile` should support incremental compilation, *i.e.* after the initial build, modifying one source file should only require the recompilation of that file (or files that depend on it, if it is a header), and re-linking the program.

3.2 Robustness

Your code must be robust. The `const` keyword must be used where appropriate, sensitive variables must be correctly protected, memory must be managed appropriately, the program must run to completion **without crash**.

3.3 Warnings

Your code **must** compile without error or warning with `g++ -std=c++14 -Wall` on `ms8??` machines. However, we advise you to check your code also with `clang++ -std=c++14 -pedantic -Wall -Wextra -Wno-c++98-compat -Wno-c++98-compat-pedantic`.

3.4 Code quality

Your code must be easy to read and understand:

- Make the organisation of your code as obvious as possible. You can create as many files, objects and functions as you see fit.
- Use descriptive and consistent names.
- Complement your self-documenting code with comments, where appropriate.
- Choose a coding convention, and stick to it. *Consistency* is key.

You will also be evaluated on the usage of the STL, correctness, object-oriented design, judicious and correct use of C++ language features, code reusability, and efficiency of the solution.

4 Submission

Projects should be submitted through the submission platform before Monday, **April the 20th**, 23:59 CET. Any late submission will receive a penalty of $2^n - 1$ points (over 20), where n is the number of started days after the deadline.

You will submit a `s<ID>.tar.xz` archive, containing a `s<ID>` folder containing your code and `Makefile`, where `s<ID>` is your ULiège student ID. As mentioned above, that `Makefile` should build the `power11` executable.

The submission platform will do basic checks on your submission, and you can submit multiple times, so submit early, submit often!

Have fun...