Georgios Solidakis (00975189)
Introduction to Computing

2048 Assignment Summary

In order to write the code for the 2048 game there are some rules that have to be followed. Before I started coding I created a list with the things that the program should and should not do.

What the program should do:

1. Load a text file called inputconf.txt containing the initial configuration.
2. Have a 2-Dimentional array with all the numbers
3. Spawn a 2-tile in a random empty space every time a move is finished.
4. Check if the move the user chose is executable and if it is not the program should not print out the same grid as before but wait for another move selection.
5. Prioritize adding depending on the move (in the case of |2|2|2|0|).
6. Add only once and not continuously (in the case of |2|2|2|2| which has to lead to|4|4|0|0| and not to |8|0|0|0|).
7. Check for Game Over when the grid is full and no move can be completed.

How I dealt with it:

The 2048 game is a 4x4 matrix so in order to represent it in C++ we needed a 2-D array. After initializing the array I needed somehow to go through all of its elements. In order to do that we need a nested for-loop going from 0 to 3 for both loops since we have 4 rows and 4 columns. I started by building the left function “a” because it seemed the easiest one as the function needed to run from left to right and my for-loops were going from 0 to 3. After building it I started doing the right function. When I finished it I realized that the only differences between the two functions were the starting point, the ending point and the step of the for-loop. This is when I decided to make a single function which does all 4 movements. This required another level of abstraction to the movement function as the indexes used in the nested for loops must dynamically be changed in accordance with the input key by the user. Since the nested for-loops iterate using different index orders (e.g. whether the “i” or the “j” goes first), pointers are needed to swap the indexes within the for-loops depending on whether the movement is horizontal or vertical.

Since we were not allowed to use “break” or “goto” statements I had to find a way to break from loops in the main function at any time depending on what was going on. This is where “array_status” takes part. “Array_status” is an integer function which can only return the values 0,1,2 depending on the circumstances. The “0” is used when there are no moves available, the “1” is used when there are moves available and the “2” is used for other exceptions such as the input being anything different to the 4 moves defined in the program.

To spawn a 2-tile in a random empty space I first came up with the idea of having two random variables which could take the values 0,1,2,3 and use them as co-ordinates in my array to find a random zero to set it equal to 2. After thinking about it and realizing that when there is only 1 zero left it will take a very long time to randomly find it like that, I came up with the idea of using a vector with all the 0-tiles and their co-ordinates in the array. This time I randomly choose one of the 0-tiles in the vector and I equate it to 2. This is done by constructing “element_position” which takes values for x and y depending on the co-ordinates of the element in the array.

To check if the move selected by the user is executable I needed to compare the array that would be printed out after the move with the previous array. If the array was not changed at all it meant that the move was not possible and the program should not print out the same array. I created a Boolean function called has_moved in order to check if anything in the array has moved. This is part of the nested for-loops so that “has_moved” is set to true every time something moves. If the for-loop was over and the if statement setting it to true wasn’t executed “has_moved” is set to false and depending on whether there are zeros or not it either terminates by setting “array_status ” = 0 or keeps running without printing out the same array by setting “array_status” = 2.

The Game Over scenario was pretty straightforward. The program should print out Game Over and terminate if the grid is full and no move can be made by the user. This is checked every time a move is made by “valid_moves” which is another Boolean function that is set to true if there are more valid moves for the user to make. If there are more valid moves “array_status” is set to 1 and if there are not it is set to 0 terminating the program and printing out “Game Over”.