2048 programme

This programme consists of one main function and 13 additional functions. In the main function, the user is asked to type the name of the input file. If the file is not found, a vector of vectors of size 4x4 is created with all zeroes in it except for the very last entry, which will contain the number 2. If the file is found, then the programme creates the vector with the numbers found inside the file.

I used a vector of vectors because it is easier to initialise than an array. In the rest of my programme, several more new vectors will be created, which is why using vectors is more convenient than using an array.

The first function is of type void with the purpose of printing out the game board. There are several commands to create new lines so the printing will be clearer – it is easy to distinguish between the previous game board and the new one.

The next function is of type void and is used when the user chooses to move to the left or to the right. Each row is taken individually and is used to create a new vector which omits all the zeroes. This is done to avoid the problems of merging already merged numbers (for example, 2 2 4 0 becomes 8 0 0 0 when moving to the left) or only merging with the adjacent element (in my function, if the elements are separated by zeroes, they will not merge).

This row vector is then sent to the 'merge' functions followed by the 'move row' functions. In the process, the programme pushes back zeroes until the row vector reaches the size needed to recreate the original matrix. The function ends by replacing the old row with the newly 'merged' and 'moved' row, and repeats until all the rows are moved. To move up or down, function of type void is used which does the same thing as the **movehorizontalmatrix** function, except that it takes each column individually instead of each row.

Moving to the left and moving up require exactly the same function once the individual row or column is taken out of the matrix and operated upon independently. The same thing can be said about moving to the right and moving down.

Taking the left/up functions as an example (the indexes count in the opposite direction for the right/down functions), an integer **i** is initialised in the **mergeleftandup** function and used to access the elements of the row/column vectors (which do not contain any zeroes). It starts with the second element in the vector. If this is equal to the value of the element to the left (or on top) of it, then it will be added to the element on the left. The current element now becomes zero. **i** is incremented twice (because if this loop runs, then there is no point in checking for the next element because the element to the left of it will be zero) and the process is repeated. If the loop does not run, then **i** is only incremented once. This process repeats until **i** reaches the end of the row. Zeroes are pushed back until the row has the original number of elements again.

This row is then sent to **moveleftanduprow**. A counter **i** is initialised to count from the second element to the last element. At the beginning of each loop, counter **j** is declared to have the value of **i** and is used to check if the element to the left of it is a zero (which signifies an empty spot). If it is a zero, then the element in **j** is swapped with the zero. **j** decrements and continues the process until it no longer encounters a zero to the left of it, or it is now at the first element. The loop ends, and **i** is incremented, and **j** is declared to have the value of **i** again.

The resultant row/column vectors are sent back to the **movehorizontalmatrix** or **moveverticalmatrix** functions, and the next row/column goes through the same process until the whole board has been moved.

The next function is of type void and is used to generate a number two randomly in one of the empty spaces. A vector containing the locations of all the zeroes in the board is created. A number is randomly generated, and used as the index of the vector. This index accesses a location with a zero, and this location will be used to place the number two. I created this vector to store the locations of all the zeroes so that the function only has to generate the random number once, instead of having to check whether the random location has a zero in it or not.

A function of type bool is used to check if the game board was changed after the user entered a move. It compares the matrix before the move is entered and the matrix after the move is entered.

To check for game over, two functions of type bool are used. The main **gameover** function checks if there are any zeroes in the matrix (if there is a zero, the board can be moved) or if there are any similar numbers in adjacent locations (if yes, they can be merged and gameplay is not over even if currently, there is no zero). The latter condition is checked row by row and column by column in the **isThereAdjacent** function.