Contents

Function: main

The main function is responsible for all dialog with the user and the input files, with the exception of the print function which uses the command “cout”. If the input file cannot be found, the function sets the values of the array for the default starting position. After this it prints the starting position and then goes into a loop in order to play until the game is lost. Everything needed to be done in this loop-other than receive an input-was transferred to a function called play for the sake of simplicity and readability.

Function: play

This function starts by deciding what the necessary increment is based on which way the user chose to swipe. Note that if a character outside the expected four is entered, the increment is set to 0 indicating the error. This increment assigning action later allows all four cases to be dealt with in single function, using the increment only to produce a different outcome. This makes the program very efficient in terms of storage as it eliminates the need to write a different (long) sequence of actions for each possible scenario.

The coordinate system used makes the upper left piece the 0^{th} location in the array, and it continues from left to right, incrementing the array value. When the end of a row is reached, the system continues from the row below until the (lower right) 15^{th} piece is reached. An increment of 4 moves the position down by 1 row in the grid. An increment of 1 moves the position right by 1 column in the grid. A decrement of equal value has the opposite effect.

It then calls the function “swipe”, which changes the values stored in array a, based on the increment value, to what it will look like next turn, except the random extra piece hasn’t been added yet.

The function “sameness” returns a bool indicating whether or not array a has changed at all during the last move. If it hasn’t changed this means an invalid move was entered and the array should neither added a new random piece, nor be printed. A second array, b exists for the sole purpose of comparing a to the itself from one move ago.

The function “new_piece“ adds a new 2 to a random location which is occupied by a 0. It also takes the Boolean variable “same” as an argument as it shouldn’t add a new piece if the move was invalid. It also has the additional purpose of copying the new state of a onto b as this has to be done after the new piece is added but before the next move. Because of this, it also has the first memory address of array b as an argument.

The function “print” is responsible for showing the user the current state of the array.

The final important thing to note about the function “play” is that it is intended to return a Boolean value. This is done because at the end of each turn, it has to be checked if the game is now over. “check_game_over” is the function that checks this.

Function: swipe

This function moves all the pieces to their new positions. The variable point is crucial, it indicates where on the grid we are currently checking. “go1” simply indicates if a character that wasn’t recognised is entered.

For easier analysis of the board;

If the swipe is in the down or right direction (increment is positive), the analysis starts from position 0. If the swipe is in the up or left direction (increment is negative), the analysis starts from position 15.

The variable “i” is used to move to the next line when after one line is corrected to their new values. This is done 4 times in each case since we have a square grid there are exactly 4 lines in any direction. This means that each line also holds four positions, no matter in which direction. “j” indicates our position on the current line. The function “next_non_zero” finds the next non zero term on the current line (if one exists) and moves it to the necessary position, it then moves us to the next position on the current line and modifies j to indicate that. The “point” variable is used to indicate current position and the variable “increment” shows which direction the line we are checking is going in.

After the correcting of one line is done and “j” equals 3, we are at the end of the first line. An equation is used to move to the start of the next line. An equation was chosen to be used as it handles the task more elegantly than multiple if statements.

The result of the equation “(increment%4) - 11*((increment+1)%2);” is added to our current position.

Input	Swipe direction	Increment Value	Increment Effect	Starting Position	Result of Equation	Effect of Equation
‘w’	Up	4	1 row down	0	-11	Start of 1 column right
‘a’	Left	1	1 column right	0	+1	Start of 1 row down
‘s’	Down	-4	1 row up	15	+11	Start of 1 column left
‘d’	Right	-1	1 column left	15	-1	Start of 1 row up

Function: next_non_zero

"i" is used to know when we have reached the end of a line. The first layer if statement knows when we have found the next non zero term in the line. The second layer if statements checks if our current position holds a 0. If it does, the next non zero term can move to our current position. Otherwise, we reach the third layer of if statements;

If our current position holds the same number as the next non zero, the 2 terms will merge. Otherwise, if the next non zero is directly next to our current position, nothing happens. Otherwise, the the next non zero is moved to the next position on the line.

Then, we move onto the next position and indicate doing so using "j".

If no next non zeros are found on the current line, then we move to the end of the line.

Function: sameness

The function simply checks every value of b against a, and returns false if any of them are not identical.

Function: new_piece

The function first counts the number of zeros in the grid, these represent the number of positions a new 2 can be placed. It then generates a random number between 1 and thee number of zeros. Then it finds the zero corresponding to that random number and replaces it with a 2.

The second part of this function copies array a onto b.

Function: print

The function prints out the current grid with tabs in between each column and each row on a different line.

Function: check_game_over

The function checks any piece that holds a zero or has a piece of the same value directly next to it. If no such pieces are found, it means the game is over.

Overall

I have written my program with the aim of having a small singular function to perform all operations in the game. Although this has led me to some elegant solutions, I believe my program is very versatile and ready to be modified and/or implemented elsewhere. I can already think of ways to improve the gaming experience by generalising the code even further and adding extra features. This might include letting the user select the grid size, randomly adding 4's adding with 2's to the grid and other such things. I know these can be easily added to my program and I know how to make them. Therein lies the true power of my program, it holds the potential to expand greatly. For example, with other ways of programming this game, as the grid size increased the size of their program would grow exponentially, whereas mine will only need slight adjustments. Furthermore, my program would require less processing with bigger grids and other extra features.