Baron Khan
bak14
baron.khan14@imperial.ac.uk

Overview

The overall design structure involves splitting the 2048 grid into separate rows and columns, and operating on each individual row/column (by storing them in a new vector and then putting the new result back into the grid) depending on the direction the player has chosen. The grid is then checked to see if there are any possible moves left.

This design structure was chosen because it would be easier to operate on each row/column individually as opposed to operating on the entire grid at the same time. However, a disadvantage to this method is that it involves a lot of extraction of rows and columns. Despite this, the separation of rows and columns means that the program structure is easier to read.

void main():
The main function loads the input file (if available) and then runs the execute_command() function. If that function returns true then the grid has changed so it should be redrawn. This occurs in a loop which terminates when no more moves can be played.

void draw_grid(const vector<int>& grid):
Takes as input the vector containing all of the elements of the grid and prints them onto the screen. Despite using 'cout', this function was separated from the main() in order to tidy up the code.

int integer_length(int n):
Returns an integer for the length of the input n. This is used in draw_grid() to align the numbers on the grid correctly (but this will only work correctly if the length of the integer is less than 8). [1]

bool execute_command(vector<int>& grid, string command):
This executes the commands depending on the input given by the user. There is a counter variable called 'executions' which counts the number of functions which have ran. If this number is greater than one it means that the 2048 grid has changed in some way. Therefore, this will be returned as true so the grid can be redrawn in the main() function (if not, then the grid did not change so there is no need to redraw it).

The commands are executed row-by-row or column-by-column. Operating on each individual column is much easier to implement as opposed to operating on the whole grid at the same time using two for loops.

bool move_row(vector<int>& grid, int row_start_index, bool moving_right):
This moves the tiles in the row defined by the starting index of the row (0,4,8,12). The row will move in the direction specified by moving_right (true for right and false for left).

Baron Khan
bak14
baron.khan14@imperial.ac.uk

void extract_row(const vector<int>& grid, int row_start_index, vector<int>& row):
This extracts the row given by the row's starting index and stores it in a temporary vector.

bool move_col(vector<int>& grid, int col_start_index, bool moving_down):
void extract_col(const vector<int>& grid, int col_start_index, vector<int>& col):
These are similar to the row functions above but apply to columns.

void move_tiles(vector<int>& tiles_list, bool moving_dir):
This is the most important function in the program. Given in input a vector containing a row/column, it will move and merge the tiles in the vector according to the direction specified by moving_dir (true for right/down and false for left/up). To make it easier for the user to determine if the tiles will move up/down/left/right, the following macro pre-processors were defined: MOVING_RIGHT (true), MOVING_LEFT (false), MOVING_UP (false) and MOVING_DOWN (true). , which would be used as the last argument to the function.

bool check_for_zeros(const vector<int>& tiles_list, bool moving_dir):
This is used in the move_tiles() function to determine whether there are any zeroes between any non-zero tiles so it is possible to move them.

void insert_tile(vector<int>& grid):
Changes one of the 0 tiles into a 2 with an equal probability. To achieve this the rand() and srand() functions from the <cstdlib> library were used. When initialising the random number generator, the seed is set to be time(NULL) from the <ctime> library. This will change the seed each time so the same pattern of placement will not occur.

bool check_grid(const vector<int>& grid):
After a command is executed and/ or a two has been inserted, this function checks whether any more moves can be made on the grid. It does this by, firstly, checking if there are any zeroes, then checking the rows and columns for adjacent pairs.