Introduction

Welcome to the Final Assignment!
In this assignment you are tasked with creating a console application version of Conway's Game of Life.

Conway's Game of Life is a game based on cellular automaton.
It is 0 player game, meaning that there is no input from the player except for defining a starting state of the game.
The game is played over a grid of cells.

Rules of the game:

1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

To get a better idea of how the game works, you may check out this web version of the game playgameoflife

Note: This assignment is meant to test your understanding of C programming concepts and techniques. It is not meant to test your knowledge in game development/engine programming, what you have learnt in the previous lessions should be sufficient for you to complete this assignment.

Project Directory

```
Final Assignment
| .vscode                   // auto generated vs code files
| tests
| | input                   // directory with all the starting states of the game
| | | beacon.txt
| | | blinker.txt
| | | glider_gun.txt
| | | glider.txt
| | | glider.txt
| | | toad.txt
| !run.bat                  // script to compile and run the game
| game.c                    // overarching game class
| game.h
| main.c                    // main entry point for the program
| map.c                     // class definition for the grid of tiles
| map.h
```

Execution

```
// !run.bat ----------------------------------- //
gcc -Wall -Werror -Wconversion -Wextra -pedantic -o GameOfLife.exe main.c game.c
map.c

// < runs "blinker" simulation model ----------- //
GameOfLife.exe 8 blinker.txt 5 5
               ^
               runs for 8 cycles
```

Instructions

Functions which you are required to edit will be highlighted in orange.
All other code must be left as is for grading purposes.

main.c

This is the entrypoint of the program.
Things worth noting:

- The program maintains 2 map buffers which hold the data for each iteration of the game
- In each game cycle the buffers are rebuilt and swapped

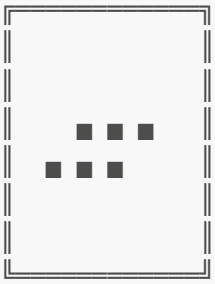
```
    Game game = CreateGame(width, height);
    game.current = &game.buffer1;
    game.previous = &game.buffer2;
```

Initialisation

- `LoadGame` is a function from the game.h library which loads the starting state of the game from a file

```
toad.txt            output

000000
000000
001110
011100
000000
000000
```



The Game Loop

A game loop is a loop that runs the cycles or "frames" of a game.
In our game loop we call the following functions:

- `system("clear")` clears the console output
- `Update` updates the game state for this iteration
- `SwapMapBuffers` swaps the map buffers `current` and `previous`
- `DisplayMap` displays the current game map stae stored in `current`
- `Sleep` adds a delay to the loop

Cleanup

- `DeleteGame` frees all memory allocated by `Game`

map.h/.c

Structures

The `Map` structure has 3 variables:

- `data` a pointer to an array of the map data
- `width` the width of the map
- `height` the height of the map

```
typedef struct _Map
{
    char* data;
    int width;
    int height;
} Map;
```

Functions

> `Map CreateMap(int w, int h);`

- Constructs a new Map object
- Takes in the width `w` and height `h` of the map to construct
- Returns the map object created
- When constructed, the data buffer in the new Map object should be zeroed using `calloc`

> `int GetValueAtPosition(Map* map, int x, int y);`

- Gets the value at a specified cell on the Map
- Takes a pointer to the Map `map` to search and the `x` and `y` position of the cell
- Returns the value of the cell

> `void SetValueAtPosition(Map* map, int x, int y, int value);`

- Sets the value at a specified cell on the Map
- Takes a pointer to the Map `map` to modify, the `x` and `y` position of the cell and the value to set

> `void DisplayMap(Map* map);`

- Displays the `data` in the Map to the console
- Takes a pointer to the Map `map` to display

> `void ClearMap(Map* map);`

- Sets all cell values in `data` to 0
- Takes a pointer to the Map `map` to clear

> `void DeleteMap(Map* map);`

- Frees the `data` buffer owned by the Map object

- Takes a pointer to the Map `map` to delete

game.h/.c

Structures

The `Game` structure has 4 variables:

- `buffer1` one of the two map buffers the game swaps between
- `buffer2` one of the two map buffers the game swaps between
- `current` a pointer to the map buffer displayed in the current cycle
- `previous` a point to the map buffer displayed in the previous cycle

```
typedef struct _Game
{
    Map buffer1;
    Map buffer2;
    Map* current;
    Map* previous;
} Game;
```

Functions

> Game CreateGame(int _w, int _h);

- Constructs a new Game object
- Takes in the width `_w` and height `_h` of the game map
- Returns the created Game object
- Creates the map buffers `buffer1` and `buffer2`

> void LoadGame(Game* game, const char* filepath);

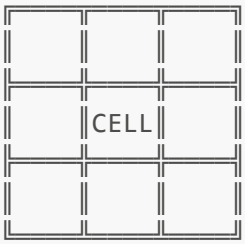
- Reads in the starting state of the Game from a file
- Takes in a pointer to the Game object `game` to populate and the path `filepath` to the test file to load
- Iterates through the characters in the file and sets the map buffer's cell values according to the characters in the file

> void DeleteGame(Game* game);

- Frees the map buffers `buffer1` and `buffer2`
- Takes in a pointer to the Game object `game` to delete

> int NeighbourCount(Map* map, int x, int y);

- Counts the number of alive neighbours of a given cell
- Takes in a pointer to the Game object `game` and the row `row` and column `col` index of the given cell
- Returns the count of alive neighbours
- The neighbouring cells are the 8 cells surrounding the given cell



> void Update(Game* game);

- Runs the game logic and updates the map buffers
- Takes in a pointer to the Game object `game` to update
- Looks through the `current` buffer and runs the game rules on it's cells, storing the result in the `previous` buffer's cells

> void SwapMapBuffers(Game* game);

- Swaps the map buffers pointed to by `current` and `previous`
- Takes in a pointer to the Game object `game` to swap buffers

Submission

- You are required to submit the modified files `game.c`, `map.c` only
- You should only modify the implementation within the functions highlighted in orange
- You are allowed to use any techniques/programming patterns previously discussed in our lessons in your implementation
- You are to ensure that there are no memory leaks in your final program submission